ARTIFICIAL INTELLIGENCE AND ITS APPLICATION IN ARCHITECTURAL DESIGN

Julian E. H. Mustoe

A thesis submitted for the
degree of Doctor of Philosophy

Department of Architecture and Building Science
University of Strathclyde
Glasgow

November 1990

I would particularly like to thank my supervisor at ABACUS, Dr Alan Bridges, for his steady support of the work on this thesis. I owe much to his good humour and perspicacity.

I am also greatly in debt to Dr Karen James for her advice and help, particularly with the intricacies of the Pascal programming language.

Table of Contents

Index of Figures

Chapter 1. INTRODUCTION

There are many areas of human activity in which artificial intelligence, when defined as "that part of computer science concerned with designing intelligent computer systems, that is, systems that exhibit the characteristics we associate with intelligence in human behaviour" (Barr & Feigenbaum, 1981), could be expected to have an application. Visual perception and the ability to distinguish patterns in the world, working with numbers in such a way as to display mathematical discrimination rather than merely an ability to calculate, and the ability to process natural language at a higher level than formal symbol manipulation, are three areas of investigation that are embraced by the topic of artificial intelligence. The rate at which research in artificial intelligence has progressed has been uneven, and has varied from one topic to another. Sometimes, as in natural language processing, little has been achieved beyond revealing how difficult the problems are.

The activity of architectural design, which touches upon all three of these topics, might be thought to be another subject which is suitable for artificial intelligence research. An architect must display visual ability, he must have a grasp, perhaps somewhat rudimentary, of mathematics, and he must be verbally literate. Workers in artificial intelligence (Akin, 1978 & 1986) have made a number of efforts to investigate design. Ramesh Krishnamurti (1985) has attempted to interpret the activity of design as a kind of set-based expert system. However, the topic has so far resisted researchers from artificial intelligence, and indeed from almost all other academic specialisms including the discipline of engineering. It is instructive to inquire as to why this is so, and the first half of this thesis is an attempt to elucidate the relationship between architecture and artificial intelligence.

The purpose of this introductory chapter is, in the first place, to describe the point of view from which the investigation has been carried out. This entails a discussion of the activity of design, and the drawing of a distinction between the notions of reason and logic. Secondly, I give a brief account of the main ideas contained in the body of the work and outline the argument by which these ideas are connected. In the third place, I identify the three topics in the thesis which I claim to be contributions to knowledge.

Four Generations of Design Studies

During the course of the last 30 years a sustained effort has been made to study and understand design, including architectural design, from a positivist point of view. An area of study has grown up in which design is studied as an academic topic, and in which it is assumed, often tacitly, that design is an objective procedure which results in the creation of a definable product. Design studies is now referenced at D620.0042 in the Dewey library classification system, and the topic is complete with research programs, learned journals and shelves of specialist books. The American journal Design Methods and Theories has been published since 1976 while Design Studies has appeared in Britain at regular intervals since July 1979. Many conferences devoted to the subject have been held in Britain and North America in the years since the pioneering meeting at Imperial College in 1962 (Page, 1963). As a result of three decades of research and publication in design studies, the connotation of the word design has expanded and the notion of design has come to embrace a broad spectrum of related activities.

Brian Logan (1987) has distinguished four generations of theories about architectural design. These he entitles A Systematic Methodology, Participation in Design, The Nature of the Design Activity, and The Failure of Method.

The first generation was a search for a systematic model of the design process. The search was directed towards discovering a working method which could be used by designers, and which would enable them to improve their results. Thomas Markus (1969) and Thomas Maver (1970) advanced their development of Asimow's 1962 model of analysis-synthesis-appraisal, Christopher Alexander (1964) prescribed a severe Cartesian purgative, John Page (1964) advanced the virtues and limitations of using physical three-dimensional models in design, while Bruce Archer (1969) proposed a conceptual model "which is intended to be compatible with the neighbouring disciplines of management science and operational research." It is notable that graph theory played a prominent part in the thinking of these investigators, particularly Maver and Alexander. In Chapter 9 of this thesis I also employ some aspects of the theory of graphs in an attempt to explain why the formalisms of logic are poor ways to try to represent the activity design.

First generation models were inadequate as descriptions of design, and few designers found them to be useful. However, they were successful in revealing some of the complexities of the activity of architecture and design. The most penetrating description of the difficulties which have been brought to light by the study of design is that given by the German-American architect Horst Rittel in 1967. He characterised design problems and the process of design as "wicked". The 11 properties of a wicked problem as listed by Rittel in 1972 are:

1. Wicked problems have no definitive formulation. Any time a formulation is made, additional questions can be asked and more information requested.
2. Every formulation of the wicked problem corresponds to the formulation of the solution (and vice versa). The information needed to understand the problem is determined by one's idea or plan of a solution. In other words, whenever a wicked problem is formulated

there already must be a solution in mind. (If the lack of a desirable view is defined as a deficiency of an architectural design, a solution to that problem - the provision of the particular view - has also been stated).

3. Wicked problems have no stopping rule. Anytime a solution is formulated, it could be improved or worked on more. One can stop only because one has run out of resources, patience, etc. (An architect could keep modifying a design solution forever - he stops because he has exhausted his fee, because the building has to be finally built, or because he has exhausted some other resource.)
4. Solutions to wicked problems cannot be correct or false. They can only be good or bad. (There is no correct or false building: there can only be a "good" building or a "bad" building.)
5. In solving wicked problems there is no exhaustive list of admissible operations. Any conceivable plan, strategy or act is permissible in finding a solution and none can be prescribed as mandatory.
6. For every wicked problem there is always more than one possible explanation. The selection of an explanation depends upon the employed Weltanschauung; the explanation also determines the solution to the problem. (The high cost of construction of a building may be attributed to the "expensive" design, to the high cost of materials, to the wages demanded by unions, to high interest rates and inflation, etc.)
7. Every wicked problem is a symptom of another "higher level" problem. (If the maintenance of the residence is "too expensive" to its inhabitants, this indicates that there is a problem with the income of the inhabitants.)
8. No wicked problem and no solution to it has a definitive test. In other words, at any time any test is "successfully" passed it is still possible that the solution will fail in some other respect. (If large

windows are designed for a residence to provide the desired views, the heating of the residence may become too expensive.)

9. Every wicked problem is a "one shot" operation. There is no room for trial and error, and there is no possibility for experimentation. (A house is designed and built - there is no going back to the beginning to design and rebuild it.)
10. Every wicked problem is unique. No two problems are exactly alike and no solutions or strategies leading to solutions can readily be copied for the next problem. (Even if two residences are designed for the same family, under the same geographical conditions they will never be identical.)
11. The wicked problem solver has no right to be wrong - he is fully responsible for his action.

Every architect will recognise in this list an apt description of the type of problem to which he must address himself whenever he sits down to the drawing board or the keyboard. Property number 1, according to which no step in design is definitive, has influenced the interpretation of graph theory that I give in Chapter 9 of this text. If every formulation of a design problem can raise another question or call for more information, then each new formulation can invalidate any already-existing formulation. This is the architectural equivalent to what has become known recently as non-monotonicity in reasoning. It may be contrasted with a monotonic inference, in which the deductively logical requirement that a conclusion cannot be accepted without accepting that the premises are maintained.

The second generation of design studies, as identified by Logan, tried to meet the difficulty of the wicked problem by proposing to describe design as a dialogue rather than in terms of a model. Perhaps no useful prescriptive model of design could be found, but the usefulness and relevance

of design work might be improved if the level of design discourse could be raised. Natural language has, since the beginning of history, been a method of grappling with wicked problems, and it might be that framing design in the form of dialogue would meet the difficulty.

Rittel suggested a structure for argumentation, whereby "the artificial separation between the expert who does the work and the client (whose problem) the work is supposed to deal with" (Rittel, 1972) is closed. Design as argumentation is described by Rittel himself in his 1972 paper as a "second generation" in design studies. Alexander shifted his attention away from mapping a problem onto a solution, and towards providing good information to the designer by means of patterns for building design. The patterns which he describes are intended to provide a common ground for discussions between client, architect and other participants in the design of a building.

> "towns and buildings will not be able to become alive, unless they are made by all the people in a society, and unless these people share a common pattern language, within which to make these buildings, and unless this common pattern language is alive itself." (Alexander et al, 1977)

Second generation design methods proved to be no more successful than their predecessors because, although the complexity of design was to some extent recognised, the methods themselves gave no guidance to the designer. As noted by Geoffrey Broadbent:

> "whilst functionalist/behaviourist techniques cannot possibly work, citizen participation, advocacy planning and 'charette' cannot work either. At best they may identify a 'highest common factor' of user needs, but compounded by the existentialist designer's needs to become himself, they may mislead him into thinking other people want the same things... It is quite impossible for either of them to avoid feeding their own preconceptions and values into the solution of design problems" (Broad-

> feeding their own preconceptions and values into the solution of design problems" (Broadbent, 1979)

The third of Logan's generations was characterised by an empirical, rather than a conceptual or a discourse-structured, approach to the activity of design. The intention was to move design practice closer to the model of the scientific method, and was influenced by readings from the work of Karl Popper.

William Hillier and his colleagues draw parallels between design and Popper's principle of falsification.

> "Design proceeds by conjecture-analysis rather than by analysis-synthesis. It is argued that if research is to make an impact upon design it must influence designers at the pre-structuring and conjectural stages. The idea that research should produce knowledge in the form of packaged information, coupled to rationalised design procedures is therefore inadequate. The aim of research should be seen more in terms of providing designers with a stronger theoretical, operational and heuristic basis from which to conjecture, rather than in terms of knowledge to determine outcomes." (Hillier et al, 1972)

But this interpretation of the matter is based upon a misunderstanding of Popper's thought. Popper's intention is not to provide a methodology of science, but rather to show how scientific and non-scientific statements can be distinguished from one another.

> "Thus my proposal was, and is, that it is this [boldness of prediction], together with the readiness to look out for tests and refutations, which distinguishes 'empirical' science from non-science, and especially from pre-scientific myths and metaphysics. [This] proposal is what I still regard as the centre of my philosophy." (Popper, 1974:981)

The only claim that Popper makes for his proposal is that it has the power of demarcation between empirically scientific and metaphysical statements. He nowhere prescribes

how science is to be conducted nor does he claim to describe 'what science really is'. Notions about design which assume that Popper has invented a methodology of science, and that an analogous methodology of architecture can be formulated, are misguided. This may be clearly seen if an effort is made to apply Popper's principle of falsifiability to a problem of design. A wicked problem cannot be definitively formulated, and it will be found that in consequence it cannot be empirically falsified. We should, I think, draw a Popperian conclusion from this, and recognise that architecture and design are inherently non-scientific in character.

The same objection must be made to Jane Darke's (1979) notion of primary design generators. She adds a preliminary stage, the generator, to the would-be Popperian conjecture-analysis model. These unsatisfactory attempts to draw an analogy between Popperian philosophy and design method has lead Nigel Cross to stigmatise this area of design studies as "the bastard field of design science." (Cross, Naughton & Walker, 1981)

Logan's fourth generation differs from the previous three. The fourth generation is one of disillusion, and he entitles it 'The Failure of Method', by which he means the failure of the scientific method to function as an adequate analogy for design. Third generation notions of design have not proved to be useful to architects or to other designers. Indeed, the lack of agreement about the correct interpretation of the term 'scientific method' has lead some commentators to doubt if improved clarity in design studies can be achieved by reference to a notion that is itself cloudy.

In his account of the modern history of design theory Logan has shown, I think convincingly, that a number of otherwise well-established models cannot be made to serve as analogies for design. In their turn operations research, manage-

ment science, graph theory, advocacy planning and science have failed to do much more than to show how hard it is to understand the activity of design.

Architectural Design

Despite its allegedly dubious antecedence, the study of design has brought into play a number of phrases and general concepts. Logan distinguishes and defines five important terms.

design activity	: a global term for all actions of design
design theory	: a system of ideas describing or explaining the design activity
design methodology	: a framework within which design decision making is sequenced, the strategic level of the design activity.
design method	: a technique selected at a particular point in the design process to achieve some objective in relation to the design problem.
design problem	: the context of the design

So pervasive have these phrases become that one or another of them is apt to spring to mind unbidden whenever the term 'design' is heard. But lying behind such definitions is the central notion of the act of design. Everyone who has experience of designing a building must have been struck, and sometimes thrilled, by the mysterious way in which an idea will arrive before the conscious mind. Sometimes an idea will give the impression that it was formulated far out in space before dropping swiftly to earth and coming silently to one's attention. It will often come into the mind at some apparently inappropriate moment, when one's conscious attention is occupied with other matters.

> "When assigned a task I am in the habit of storing it in my memory, that is not allowing myself to make any sketches for months. The human brain is made in such a way that it has a

> certain independence; it is a box into which one can pour in bulk the elements of a problem and then let them float, simmer, ferment. Then, one day, a spontaneous initiative of one's inner being takes shape, something clicks, you pick up a pencil, a stick of charcoal, some coloured pencils (colour is the key to the process) and give birth onto the paper: out comes the idea ..." (Le Corbusier, undated)

This creative act, upon which the whole process of design is focussed, can be facilitated, and the ability to think creatively can be fostered, but the act itself cannot be predicted, described or explained. That is why so much of what has been written under the heading of design studies seems to be merely pushing the description of design back a few steps in the explanation. The industrial designer Christopher Jones, for example, lists (1970) four methods of searching for ideas, but devices such as brainstorming, synectics, removing mental blocks and using morphological charts are all ways of facilitating rather than explaining the creative act. Hillier and his colleagues (1972), after referring to instrumental sets, solution types, informal codes, analogy and metaphor, are finally driven to call upon "what is called inspiration" as a notion of last re sort. In short, the whole panoply of models and methods that have been proposed under the rubric of design studies may have furthered understanding of matters peripheral to the act of design, but they have little to say about the act itself. I think that the topic of design studies, for all the time and effort that has been expended upon it, is concerned with the trappings but not the substance of design.

When I employ the term 'design' in this thesis I am referring to the creative act of design and not to the concepts and notions of design studies. Similarly, I use the phrase 'architectural design' in the sense of the intuitive creation of the idea of the building. The creative act, in the course of which the substance of a design is formulated, is the activity from which all else in design springs. Its

centrality is responsible for the fact that the designer, and particularly the architect, will see the subject of design from the point of view of one whose role it is to practice creative invention. This thesis is written from that same viewpoint, and my effort is therefore focussed upon bringing artificial intelligence to bear not upon design method, but upon facilitating the act of design itself.

> "Computers can aid in the design process by generating alternative solutions that nurture the intuitive leap, and can help design development. However, they do not partake in performing the intuitive leap itself, only facilitate it." (Norman, 1987)

The unhappy history of design studies convinces me that creation is not in fact explicable, nor is the process of obtaining a novel notion capable of being imitated. I suspect, although I cannot prove, that the intuitive leap is inherently inexplicable and that it will always remain inimitable. Therefore I confine my attention in this thesis to finding ways of facilitating the work of the designer rather than trying to replicate, formalise or mechanise the central act of design itself.

Logic

It is sometimes asserted that there are three types of reasoning. In most texts these are given as deduction, induction and abduction.

> "Deduction is the basic building block of formal reasoning systems. It is generally recognised, however, that people have recourse to two other modes of reasoning: namely, induction and abduction." (Coyne et al, 1990)

However, I think that to include all three types of reasoning under a single undifferentiated heading, as if they are equivalent, is to confuse the two quite different notions of logic and of reasoning.

Reasoning is a more general term than logic, and by it is meant the cognitive activity of making inferences. The word inference has as its root the Latin verb infero, meaning to introduce or to carry in. Inference implies no more than the transference of meaning or, as the Oxford English Dictionary has it "the forming of a conclusion from premises, either by induction or deduction". Reasoning therefore allows one judgement to follow from another, but nothing is specified about the inferential method to be employed. I try in this thesis to use the terms reason and reasoning in this general sense of thinking in an orderly and accountable manner.

Logic, however, is more circumscribed and is concerned with the study of valid argument. A concise statement of the distinguishing features of the validity of a logical argument are:

> "(a) its conclusions could not be false if all its premises were true.
> (b) its conclusions contain no more content than is already provided in its premises.
> (c) the addition of further premises can neither strengthen nor weaken the argument, which is already maximally strong." (Rankin, 1988)

The effect of these requirements is that the conclusion of a logical argument must necessarily be true so long as its premises are true. A well-known example of argument by means of deductive logic takes the form of the syllogism. In a syllogism of what is known as the first figure form, a major and a minor premise result in a conclusion which differs from the premises. For example, if a major premise is that all cats are animals and a minor premise is that Orlando is a cat, then the conclusion is that Orlando is an animal upon pain of contradiction. The terms given by Aristotle to the parts of a syllogism are still in use. Argument from premises to conclusion, of which the syllo-

gism is but one type, is known as deduction and when conducted without contradiction it is logically valid of necessity.

The phrase 'inductive logic' occurs frequently in texts dealing with the methodology of science. Sometimes there is a tone of desperation about efforts to establish the logicality of induction. Bertrand Russell, for example, argues that induction is indispensible to scientific thought.

> "it seems clear that whatever is not experienced must, if known, be known by inference... If I ever have the leisure to undertake another serious investigation of a philosophical problem, I shall attempt to analyse the inferences from experience to the world of physics, assuming them capable of validity, and seeking to discover what principles of inference, if true, would make them valid. Whether these principles, when discovered are accepted as true, is a matter of temperament; what should not be a matter of temperament should be the proof that acceptance of this is necessary if solipsism is to be avoided." (Russell, 1944)

However, despite the desirability and usefulness of induction, I think that induction cannot be looked upon as a form of logic. This is because no inductive argument can be relied upon to be valid in all circumstances. An often-quoted example of this fragility is the fact that swans were known, on the basis of inductive 'logic', to be white until the exploration of western Australia. When evidence of the existence of the black swan Chenopis Altrata reached Europe in the eighteenth century the supposed fact that all swans are white had to be abandoned, and the argument upon which the supposition was based was shown to be invalid. Popper generalises this criticism of induction when he observes that,

> "I hold with Hume that there simply is no such logical entity as an inductive inference: or, that all so-called inductive inferences are logically invalid - and even inductively invalid, to put it more sharply... We have many examples of deductively valid inferences, and

> even some partial criteria of deductive validity; but no example of an inductively valid inference exists." (Popper, 1972)

Valid deduction, then, is argumentation in which the premises cannot be true while the conclusion is false, while induction can be summarised as argumentation from many particulars to one conclusion. In the early years of this century the American philosopher Charles Peirce proposed a third type of reasoning derived from Aristotle's work on the syllogism. Peirce's notion is that the minor premise of a syllogism can be derived from the major premise and the conclusion. Furthermore, according to Peirce, Aristotle himself must have thought of abduction.

> "he would have asked himself whether the minor premise of such a syllogism is not sometimes inferred from its other two propositions as data. Certainly he would not have been Aristotle to have overlooked this question." (Peirce, 1901)

To take the example of Orlando the cat, an abductive syllogism would say that if all cats are animals and Orlando is an animal, then Orlando is a cat. Clearly in this circumstance Orlando may be a cat, but the argument does not preserve him from being a dog, a horse or any other example of the class of animal. Peirce himself put forward abduction as a vehicle for discovery, but he nowhere describes it as a logic.

In everyday thought one habitually employs abduction as a method of speculation, but it is a mistake to suppose that it is a generally valid form of logic. One may, for example, suppose that today is a bank holiday from the major premise that no newspapers appear on bank holidays and the conclusion that the Correspondent cannot be bought today. However, there are clearly other possible explanations for the unavailability of the Correspondent, and such an abductive argument is not a secure one.

In fact, Peirce likens abduction not to logic but to perception.

> "abductive inference shades into perceptual judgement without any sharp line of demarcation between them; or, in other words, our first premises, the perceptual judgements, are to be regarded as an extreme case of abductive inference, from which they differ in being absolutely beyond criticism. The abductive suggestion comes to us like a flash. It is an insight, although of extremely fallible insight." (Peirce, 1903)

I conclude that the inherent fallibility of induction and abduction means that arguments based upon them can never be secure. We are left only with properly formed deductive argument as a logically valid procedure. As the American philosopher David Israel (1987) has put it "logic - deductive logic, for there is no other kind". I have therefore confined the use of the words logic and logical in this thesis strictly to deductive argument. Other forms of inference, despite the fact that they are often described elsewhere as logics, are referred to in this text as reasoning.

The Structure of the Thesis

This thesis is divided into three main sections. The first two sections are a critical examination of some aspects of artificial intelligence, and are theoretical in character. The last section is an implementation based upon some parts of the topic of knowledge engineering. It takes the form of a non-deductive expert system working with a library of photographs of buildings which is stored on an optical disk.

The first part of the main body of this thesis is concerned with a consideration of artificial intelligence in its role as a way of using computers to replicate or to study the action of the mind. This topic is variously known as ma-

chine thinking, machine intelligence or cognitive science. In order to avoid prejudging the issue, I have used the term cognitive simulation as a general designation for this aspect of artificial intelligence.

Those theories that state or imply that a computer can actually think I describe as strong cognitive simulation. A well-known example of this line of thought is the idea of scripts put forward by the American computer scientists Roger Schank and Robert Abelson in 1977. Scripts are equipped with an underlying theory of meaning referred to by its authors as conceptual dependence. I give an account of Shank and Abelson's theory of strong cognitive simulation, and of its refutation by the Berkeley philosopher John Searle.

In a less extreme version of cognitive simulation the claim that the machine is thinking is abandoned, and the object of the search becomes a machine that imitates, rather than replicates, human thought. Three areas in which this has been attempted are examined. These are machine translation of natural language, computer chess and pattern recognition. I think that the American philosopher Hubert Dreyfus shows convincingly that these activities, which I collectively refer to as weak cognitive simulation, are also impossible at any but a very elementary level.

There is a close parallel between the earlier and the later epistemology of Ludwig Wittgenstein, and the evolution of ideas about artificial intelligence which has occurred during the last 35 years. In both cases the movement has been away from logic and towards a recognition that meaning is a function of the complex nature of human thought and language. Chapters 6 and 7 are taken up with an examination of some aspects of Wittgenstein's thought, from which emerges the conclusion that if the later Wittgenstein is correct, then the earlier positivist program for artificial intelligence is impossible. I claim that establishing a

Wittgensteinian perspective upon some aspect of artificial intelligence is the first of the three contributions to knowledge that is made in this thesis.

Dreyfus successfully disposes of weak cognitive simulation, but he overstates his case when he concludes that artificial intelligence is a futile study because a computer cannot fully imitate a human mind. It is true that high quality human thinking will always rise above the best performance of a computer, but that does not, in my opinion, mean that such intelligence as a computer can simulate may not be useful. This leads me to propose a taxonomy for the whole field of artificial intelligence, from which I have chosen the topic of expert systems as suitable for further investigation. Intelligent tutoring systems are also a department of artificial intelligence in which progress could be made.

Expert systems, as they are usually constructed, are an outcome of the automation of logic. I examine through the medium of graph theory the rule-based expert systems that emerge from this type of programming, and conclude that they do not meet the needs of the architect. This is largely as a result of the incompatibility of wicked design problems and deductive logic.

However, an alternative type of expert system has been suggested by the American computer scientist Peter Frey which promises to be useful in the context of design. It is possible to see an expert system as a method of classifying solutions in terms of domain attributes. This notion opens the way to producing an expert system which can classify aspects of design with reference to the preoccupations and interests of the designer, rather than according to some inappropriate scheme of logic.

As a result of examining the coding of Frey's classification system I propose in Chapter 12 an improved algorithm

for a classification expert system. The new algorithm incorporates Frey's bit-matching technique while extending it by improving the calculation of probabilities, by providing explanations for questions and solutions, by storing the knowledge base in files and by replacing Microsoft BASIC with the Prospero implementation of Pascal. This program, called Cortex, is the second innovation to which I lay claim in this text.

The method by which an expert system is controlled is the part of its algorithm that has the greatest effect upon the performance and effectiveness of the system. Cortex incorporates a novel method of control that is applicable to any type of expert system. The control method used in Cortex is the third of the three contributions to knowledge that I claim in this thesis.

The final part of the thesis is an implementation of a prototype version of Cortex which accesses an optical disc intelligently. Sets of photographs of buildings are selected from the 10,000 images on the disc, and are displayed according to the user's conception of architecture and his attitude toward the design of buildings. The result is intended to be a system that responds to an individual architect's viewpoint, and which casts light upon his interests as a designer.

Chapter 2. ARTIFICIAL INTELLIGENCE AS MACHINE INTELLIGENCE

The two words that are juxtaposed in the phrase 'artificial intelligence' possess widely different connotations when they are used apart from one another in ordinary discourse. 'Artificial' in the sense of the Oxford English Dictionary definition of 'made by art in imitation of what is natural or real' carries with it an implication of something feigned or fictitious. A thing which is artificial is close to being an inferior substitute for that which is real and natural. On the other hand intelligence is a human quality, extended somewhat to the rest of the animal kingdom, which is universally admitted to be estimable, natural and innate. The dictionary definition of intelligence as "quickness of mental apprehension; understanding as a quality admitting of degree" is applicable, to at least some extent, to all the higher animals. It is a faculty that can be trained and developed during life, but only nature can create it. To speak of artificial intelligence is therefore to employ a syncretism with an highly tensioned internal structure. The connotation of artifice in the first term strains against the implication of naturalness in the other. Artificial intelligence is thus an inherently ambiguous piece of terminology, and it is therefore not surprising that discussions in which the phrase occurs are often confused and contradictory.

Definitions

Many attempts to formulate a specification for AI have been made since the phrase was coined in 1956. The spectrum of meaning to be found in the following 19 definitions closely reflects the compound nature of the phrase itself. The definitions are listed in chronological order.

> John McCarthy proposed that "a two-month, ten-man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature

of intelligence can in principle be so precisely described that a machine can be made to simulate it." (Charniak & McDermott, 1985)

"artificial intelligence is the science of making machines do things that would require intelligence if done by men." (Minsky, 1968)

"By 'artificial intelligence' I ... mean the use of computer programs and programming techniques to cast light on the principles of intelligence in general and human thought in particular." (Boden, 1977)

"artificial intelligence is the use of programs as tools in the study of intelligent processes" (Boden, 1977)

"Our approach to the AI problem involves identifying the intellectual mechanisms required for problem solving and describing them precisely..... General intelligence will require general models of situations changing in time, actors with goals and strategies for achieving them, and knowledge about how information can be obtained." (McCarthy, 1979)

"The ultimate AI program that we are all aiming for is one that specifies the form in which knowledge is to be input to the program, as well as the form of the rules that use that knowledge, and produces a program that effectively models that domain." (Schank, 1979)

"AI is that part of computer science concerned with designing intelligent computer systems, that is, systems that exhibit the characteristics we associate with intelligence in human behaviour - understanding language, learning, reasoning, solving problems, and so on." (Barr & Feigenbaum, 1981)

"Artificial intelligence is the study of how to make computers do things at which, at the moment, people are better." (Rich, 1983)

"Artificial intelligence is the study of techniques for solving exponentially hard problems in polynomial time by exploiting knowledge about the problem domain." (Rich, 1983)

"the discipline of Artificial Intelligence, a principle concern of which is the design of computer programs to undertake activities thought to require human intelligence." (Alty & Coombs, 1984)

"for the time being we are going to have to define intelligence in machines in the same way that Justice Potter Stewart described pornography: 'I can't define it but I know it when I see it.'" (Michie & Johnson, 1984)

"Artificial intelligence is the study of ideas that enable computers to be intelligent." (Winston, 1984)

"Artificial intelligence is the study of mental faculties through the use of computational models." (Charniak & McDermott, 1985)

"Artificial Intelligence has two different products: models of human cognition and intelligent artifacts." (Sell, 1985)

AI "covers a broad spectrum of interests loosely linked by a shared ambition to represent more of human intelligence in machines." (Bijl, 1986)

"A discipline concerned with the building of computer programs that require intelligence when done by humans." (Illingworth, Glaser & Pyle, 1986)

"The real goal of AI, after all, is to design or understand systems that can reason about the world, not themselves." (Smith, 1986)

"As an attempt to sum up the various definitions of AI, I would like to categorise sophisticated programming techniques (the so-called 'smart programs') as syntactical approaches, and the search for 'principles of intelligence' as a semantic approach. A further development would then be a pragmatical approach which I would like to consider as a new paradigm (or working philosophy)." (Born, 1987)

"As engineering, AI is concerned with the concepts, theory, and practice of building intelligent machines... As science, AI is developing concepts and vocabulary to help us to understand intelligent behaviour in people and in other animals." (Genesereth & Nilsson, 1987)

Most of these quotations define artificial intelligence as programming a computer to simulate, which is to say to assume the appearance of, cognition. These definitions in effect equate AI with cognitive simulation, which I shall

hereafter abbreviate to CS. The artificiality of artificial intelligence is emphasised, and the intelligence that is looked for is of an operational, not an ontological, type. I shall discuss later the sense in which artificial intelligence can be said to simulate human intelligence under the heading of weak CS.

In the meantime, however, I propose to discuss the very different conception of artificial intelligence which is implied by the two definitions of artificial intelligence produced by McCarthy and by Schank in 1979. Artificial intelligence is, according to these authors, a matter of creating the reality rather than the appearance of intelligence. The emphasis in McCarthy and Schank is upon intelligence rather than artifice. The intelligence displayed by the computer, in this view of the matter, would be manufactured by man but would possess a real existence. I refer to this interpretation of artificial intelligence as strong CS.

The ambition to create strong CS programs propelled much of the early work on artificial intelligence, particularly during the 1950's and 60's, and echoes of this idea can still be heard. For example, the 1986 paper by Smith from which one of the definitions in the above list is taken. Several influential authors, Herbert Simon (1977) and Marvin Minsky (1966) in particular, have written optimistically on the prospects for the achievement of strong CS. Furthermore, the vision of a machine with its own independent intelligence is widespread in the popular imagination and it is a favourite topic with, for instance, television journalists (Vaux, 1988) and film makers (Austin, 1968). It is important, therefore, for both theoretical and historical reasons to establish clearly what is meant by the term intelligence and the sense, if any, in which a machine can correctly be said to possess it.

It is easier to anthropomorphise the computer than any other machine. One may, perhaps, speak of a motor car as 'tired' or a sailing yacht as 'gentle'. Sometimes, when typing mistakes occur frequently, a typewriter may seen to have acquired 'a mind of its own'. These phrases, of course, are never meant to be taken in other than an ironic or metaphorical sense. But a computer is a symbol manipulating machine which is able to accept natural language input, and which can produce output to which meaning may be attributed. The process by which these symbolic transformations take place will generally be so complicated that no human brain can comprehend it in its totality. A certain mysteriousness adheres to the inner workings of a computer, even for those who are accustomed to using them. Moreover, output can follow input amazingly quickly. When faced with the speed, complexity and dependability of a working computer an observer can allow himself to suppose that the machine is thinking, and consequently he may be prone to attribute to it at least some of our own cognitive faculties.

Furthermore, the use of phrases such as 'meaningful output' in the previous paragraph can be taken to signify that meaning is a property which resides in the output material itself. I shall shortly give some reasons as to why this single-term definition of 'meaning' is mistaken. But it remains that our language has not yet developed ways of describing accurately the new aspects of the relationship between man and machine which the invention of the computer has brought about. There is, for example, no accepted linguistic distinction between, on the one hand, the way in which information is processed by a computer and, on the other, the sense in which human beings make use of information. Indeed, some investigators allow themselves to assume that because the same word is used the meaning must be the same in the two cases. The confusion that characterises Minsky's editorial introduction to Semantic Information Processing (1968), for example, is principally due to a

failure to observe this distinction between two or more different uses of such words as 'learn', 'understand' and 'intelligent'.

It seems to me that for the present we must live with a situation in which the appearance of thinking possessed by a functioning computer is compounded in our minds with a deeply-seated verbal ambiguity about the nature of computer operations. With the publication of his paper entitled 'Minds, Brains and Programs' in 1980 John Searle has tried to dispel some of the confusion which results when familiar concepts are used unguardedly in the novel context of computing. I believe that Searle has, in large part, made his case and the next chapter of my thesis is occupied with an account of why I think that he has been successful.

Scripts and Conceptual Dependency

In 1977, three years before Searle's paper appeared Roger, Schank and Robert Abelson published their 'Scripts, Plans, Goals and Understanding'. These two authors were working at that time at the Department of Computer Science of Yale University, and their book has since become well known in artificial intelligence circles. 'Scripts, Plans, Goals and Understanding' rehearses the author's notion of conceptual dependency and it proposes a novel method of representing knowledge under the name of scripts. The notion of scripts, and the claims made for this method of representation by Schank and Abelson, is the specific target of Searle's critique. In his paper Searle attacks the idea of strong CS as exemplified by scripts, and it is therefore necessary at this point to devote some space to an account of Schank and Abelson's proposal.

Two models of human memory are currently favoured by psychologists. According to the semantic conception of memory we possess in our minds, or from the physicalist perspective our brains, a permanent store of knowledge the items of which are related to one another according to their

meanings. Meaning is then attributed to words or other objects of experience by a mental process of searching a tree-like structure of semantic concepts. For example, 'claws' are related to 'tigers', 'tigers' to 'cats', 'cats' to 'carnivores' and 'carnivores' to 'animals' by means of a hierarchy organised on semantic principles. The structure of concepts, once learned, remains with us through life and we access it as required during cognitive activity. The semantic model of memory assumes, in effect, that our minds are analogous to a library and that we make use of something equivalent to the Dewey Decimal Classification for the purpose of locating and attributing meaning.

Schank and Abelson adopt the alternative view of memory, according to which we accumulate a store of personal experiences rather than semantic concepts, and that the mind accesses memory according to a scale of time. This is referred to in the literature of psychology as the episodic model of memory.

"The over-all organisation of memory is a sequence of episodes organised roughly along the time line of one's life. If we ask a man 'Who was your girlfriend in 1968?' and ask him to report his strategy for the answer, his reply is roughly: 'First I thought about where I was and what I was doing in 1968. Then I remembered who I used to go out with then.' In other words, it really isn't possible to answer such a question by a direct look-up. Lists of 'past girlfriends' do not exist in memory. Such a list must be constructed. The process by which that list is constructed is a search through episodes organised around times and location in memory." (Schank & Abelson, 1977:19)

It is a consequence of this view of memory that the mind must be able to work with an assembly of items whose meanings bear no intrinsic relationship to one another. Memory episodes occur as a result of the chances of life, and their structure and relationship in the mind reflects the fortuitousness of events. The question therefore arises as to how the mind relates one episode to another in such a

way as to acquire general knowledge or recognise repeating occurrences? Some mechanism must be at work to place the episodes into an organised and comprehensible form.

> "If memory is organised around personal experiences then one of the principal components of memory must be a procedure for recognising repeated or similar sequences. When a standard repeated sequence is recognised, it is helpful in 'filling in the blanks' in understanding. Furthermore much of the language generation behaviour of people can be explained in this stereotyped way." (Schank & Abelson, 1977:18)

This requirement for a principle upon which memory episodes can be ordered provides Schank & Abelson with the clue which leads them to their notion of scripts.

> "Some episodes are reminiscent of others. As an economy measure in the storage of episodes, when enough of them are alike they are remembered in terms of a standardised generalised episode which we will call a script. Thus, rather than list the details of what happened in a restaurant for each visit to a restaurant, memory simply lists a pointer (link) to what we call the restaurant script and stores the items in this particular episode that are significantly different from the standard script as the only items specifically in the description of that episode. This economy of storage has a side effect of poor memory for detail. But such a side effect, we shall argue, is the price of having people able to remember anything at all. Script based memory is what will enable computers to understand without having their memories filled up with so much that search time is horrendously long." (Schank & Abelson, 1977:19)

Schank and Abelson have developed the notion of scripts into a structured formalism by means of what they refer to as the theory of conceptual dependence. According to this theory, which in point of fact is no more than an assertion, there exists beneath language a foundation of meaning which can be precisely described and to which any sentence in any language can be reduced.

"Conceptual Dependency (henceforth CD) is a theory of the representation of the meaning of sentences. The basic axiom of the theory is:

A For any two sentences that are identical in meaning, regardless of language, there should be only one representation.

The above axiom has an important corollary that derives from it.

B Any information in a sentence that is implicit must be made explicit in the representation of the meaning of that sentence." (Schank & Abelson, 1977:11)

They proceed to list the 11 primitive actions which they claim can, when qualified by means of a numerical scale running from -10 to 10, serve to represent the meaning of every conceivable sentence. The primitive actions are presented as;

ATRANS The transfer of an abstract relationship such as possession, ownership or control.
PTRANS The transfer of the physical location of an object.
PROPEL The application of physical force to an object.
MOVE The movement of a body part of an animal by that animal.
GRASP The grasping of an object by an actor.
INGEST The taking of an object by an animal to the inside of that animal.
EXPEL The expulsion of an object from the body of an animal into the physical world.
MTRANS The transfer of mental information between animals or within an animal.
MBUILD The construction by an animal of new information from old information.
SPEAK The action of producing sounds.
ATTEND The action of attending or focusing a sense organ towards a stimulus.

It is then asserted that separate sentences expressed in this notation can be assembled into a complete text by means of inferentially connected causal chains.

"not any action can result in any state, and not any state can enable any action. Thus, for every primitive action, there is associated with it the set of states which it can affect as well as the states that are necessary in order to effect it." (Schank & Abelson, 1977:25)

The theory of conceptual dependency is made up, then, of two components. Firstly, the idea that all discourse can be described by reference to a definable set of semantic concepts, and secondly the notion that these concepts can be related to one another in a causal manner. A full description of conceptual dependency, including examples of the notation, is given in Schank (1975).

The concept of a script reaches its complete formulation when the problem of generality is addressed. A separate script to describe each story would defeat the purpose of scripts, which is to provide an economical way of representing meaning. Some generalising mechanism is called for. Schank and Abelson propose to achieve this by uniting a script, representing the structure of a type of situation, with a knowledge base containing the events that are characteristic of a particular state of affairs. The new composite and flexible representation they call a knowledge structure.

> "we are establishing a level of representation different from Conceptual Dependency. The primitive ACTS and causal links of Conceptual Dependency are used to describe real world events, while script names make reference to the knowledge structures that motivate or underlie real world events. These levels of representation are connected by what we will call the Script link. The representation that we used above (with $SCRIPTNAME and its various roles) is this higher Knowledge Structure level. It is connected by a Script link to the Conceptual Dependency structure that instantiated it.
> What we are proposing then is that there be both a knowledge structure (KS) and a Conceptual Dependency (CD) representation for any given text. Some texts will not actually impart information about both, but it is to be expected that in most texts there will be enough complexity to necessitate that both levels be represented, with links between them." (Schank & Abelson, 1977:152)

A script is thus a stereotyped representation of a sequence of events occurring in a particular context. It has proved

to be a serviceable idea when the events and their context permit the two axioms of conceptual dependency to be adhered to. That is to say, scripts are found to be adequate in straightforward contexts where a single representation is capable of embracing the meaning of two or more sentences, and in which the meaning is simple enough to be represented explicitly. Under these circumstances a text can be expressed fairly adequately in the form of the conceptual dependency notation. A restaurant is the context which crops up most often in the literature of scripts, and the events are such things as ordering a meal, eating it and paying the bill. In such circumstances the meaning of a sentence is unambiguous and the causal chain is reasonably clear.

The authors have derived the notion of scripts from the episodic theory of memory, and they go on to claim that the script concept can in its turn throw light on the psychology of cognition. Scripts, they say, are a pattern for the way in which we understand the world.

> "By subscribing to a script-based theory of understanding we are making some strong claims about the nature of the understanding process. In order to understand the actions that are going on in a given situation, a person must have been in that situation before. That is, understanding is knowledge based. The actions of others make sense only in so far as they are part of a stored pattern of actions that have been previously experienced." (Schank & Abelson, 1977:67)

But circumstances in life are rarely so clear-cut. While it is true that previous experience is necessary for understanding, it is also true that most real-life situations are too complicated to be explicable by reference to any one script or set of scripts. This is because as events occur they have the effect of redefining the significance of those events that have previously occurred. A diner's conduct in a restaurant, to take Schank and Abelson's favourite context, will be affected by, among other things,

what he has been told that he should expect of the particular restaurant. What he makes of the food and the service, and how he responds to them, may then reflect back upon his judgement upon the restaurant itself or upon his opinion of his informant, or upon both. Scripts are dependent upon understanding, as well as understanding being to some extent dependent upon scripts.

That is why most texts, or stories to adopt the terminology of scripts, are to some degree metaphorical and allusive. In everyday discourse understanding is achieved by metaphorical transference of meaning and accepted patterns of knowledge, for which one may read scripts, are modified by a process of allusion. To take an extreme and therefore illustrative example, no 'script-based theory of understanding' could represent adequately the meaning, or rather the meanings, of the following passage from The Third Policeman.

> "The reader will be familiar with the storms that have raged over this most tantalising of holograph survivals. The 'Codex' (first so called by Bassett in his monumental De Selby Compendium) is a collection of some two thousand sheets of foolscap closely handwritten on both sides. The signal distinction of the manuscript is that not one word of the writing is legible. Attempts made by different commentators to decipher certain passages which look less formidable than others have been characterised by fantastic divergences, not in the meaning of the passages (of which there is no question) but in the brand of nonsense which is evolved. One passage described by Bassett as being 'a penetrating treatise on old age' is referred to by Henderson (biographer of Bassett) as 'a not unbeautiful description of lambing operations on an unspecified farm'. Such disagreement, it must be confessed, does little to enhance the reputation of either writer." (O'Brien, 1967)

O'Brien's paragraph is meant to be read in at least four distinct ways. It functions as a part of the plot of his novel, it is an ironic comment on the difficulty of reading

accurately, it takes a swipe at self-important editors, and it implies that the only response to really intractable problems is laughter. The four levels of meaning interact with one another, and they alter their relative importance according to the presuppositions which each particular reader brings to the text. Consequently, there is no one representation into which even a single sentence of The Third Policeman could be mapped. It is therefore impossible, because of axiom A, to apply the technique of Schanki-an scripts to a text which is as multi-layered and expressive as is the work of Flan O'Brien.

However, it is sufficient for the present purpose to note two things about the notion of scripts. Firstly, it follows from axiom B above that a formal system of symbols would suffice to describe a script fully and completely. Scripts are therefore by definition computable. Secondly, Schank and Abelson make some large claims for their work.

> "SAM (Script Applier Mechanism) is a program running at Yale that was designed to understand stories that rely heavily on scripts....
> SAM understands these stories and others like them. By 'understand' we mean SAM can create a linked causal chain of conceptualizations that represent what took place in each story. SAM parses the story into conceptualizations using Reisbeck's analyser (Reisbeck, 1975). These are then fed to a program that looks for script applicability (Cullingford, 1976). When a script seems to be applicable, it is used by the script applier to make inferences about events that must have occurred between events specifically mentioned.
> The final representation is a gigantic Conceptual Dependency network. We could claim that this output indicates understanding, but as no one can read it (and for the more obvious reasons) we have developed programs that operate on the output of the understanding program." (Schank & Abelson, 1977:177)

This amounts to saying that they have instantiated strong CS in the computer centre at Yale. The fact that they draw close parallels between human understanding and computer

processing of scripts demonstrates that their use of the word understanding in the phrase 'this output indicates understanding' is to be taken literally rather than figuratively. The computer understands, they say, in the same way as a human being and strong CS has, for them, become a reality. However, I believe that Schank and Abelson are deluded when they claim to have reproduced cognition. John Searle's paper of 1980 shows, I think, why it is that they are mistaken.

Chapter 3. THE CRITIQUE OF JOHN SEARLE

It falls to few philosophers to describe a Gedankenexperiment that becomes famous. It is true that Christian Huygens' use of symmetry arguments in the seventeenth century to derive the conservation laws for momentum and energy appear frequently in modern textbooks of mechanics (Layzer, 1984). The paradox of Schrodinger's cat (Schrodinger, 1935), which raises as-yet unanswered questions about quantum theory, is referred to regularly in recent scientific discussions. But Searle's Chinese room argument has appeared at least twice before the general public (Searle, 1984, Vaux, 1988) as well as attracting the attention of innumerable philosophers, linguists and workers in artificial intelligence since its publication only ten years ago. It has, in fact, come into use as a standard shorthand description of a certain type of critical comment on artificial intelligence.

The Chinese Room

Searle describes his thought experiment in the following way.

> "Suppose that I'm locked in a room and given a large batch of Chinese writing. Suppose furthermore (as is indeed the case) that I know no Chinese, either written or spoken, and that I'm not even confident that I could recognise Chinese writing as Chinese writing as distinct from, say, Japanese writing or meaningless squiggles. To me, Chinese writing is just so many meaningless squiggles. Now suppose further that after this first batch of Chinese writing I am given a second batch of Chinese script together with a set of rules for correlating the second batch with the first batch. The rules are in English, and I understand these rules as well as any other native speaker of English. They enable me to correlate one set of formal symbols with another set of formal symbols, and all that 'formal' means here is that I can identify the symbols entirely by their shapes. Now suppose that I am given a third batch of Chinese symbols together with some instructions, again in English, that

enable me to correlate elements of this third batch with the first two batches, and these rules instruct me how to give back certain Chinese symbols with certain sorts of shapes in response to certain sorts of shapes given me in the third batch. Unknown to me, the people who
1 are giving me all these symbols call the first batch 'a script', they call the second batch a 'story', and they call the third batch 'questions'. Furthermore, they call the symbols I give them back in response to the third batch 'answers to the questions', and the set of rules in English that they gave me, they call 'the program'. (Searle, 1980a:417-418)

The events that take place in the Chinese room form an almost exact parallel with the highly anthropomorphic account of running the Script Applier Mechanism which is given by Schank and Abelson in section 8.2 of Scripts, Plans, Goals and Understanding. In a run of SAM, one or several scripts is read into computer memory, next a second
2 input is made in the form of the story, then questions about the story are input, and lastly answers are computed and printed out. The function of SAM is to organise and control these processes. The only difference, and it is a critical difference, between the procedures at Yale and operations in the Chinese room is that SAM is processed on a computer while Searle's Chinese symbols are processed in a human brain.

Searle's point is that when the Chinese room is regarded as
3 an input/output system it is in fact simulating, and not replicating, understanding.

> "it seems to me to be quite obvious in the example that I do not understand a word of the Chinese stories. I have inputs and outputs that are indistinguishable from those of the native Chinese speaker, and I can have any formal program you like, but I still understand nothing. For the same reasons Schank's computer understands nothing of the stories, whether in Chinese, English or whatever since in the Chinese case the computer is me, and in cases where the computer is not me, the computer has nothing more than I have in the case where I understand nothing." (Searle, 1980a:418)

I think that it is no more than the simple truth to say that Searle, locked in his room and performing his role of an English speaking manipulator of Chinese symbols, does not understand the Chinese language. Searle's understanding of English suffices for him to follow the English program, and by following it to apply a Chinese script to a Chinese story. Similarly his knowledge of English enables him to use the English language program to supply answers in Chinese to questions about the story which are presented to him in Chinese. But the distinction between, on the one hand manipulating a system of formal symbols, as he is doing, and on the other hand understanding a story, remains clear and incontrovertible. It follows that if Searle does not understand the story then neither does a functionally analogous CPU in a computer which is occupied with, for example, a run of SAM. Claims made for strong CS thus emerge as the result of confusing symbol manipulation with understanding.

Searle's refutation of the case for strong CS is important because he offers a conceptual, not an empirical, argument.

> "It is an empirical question whether any given machine has causal powers equivalent to the brain. My argument against strong AI is that instantiating a program is not enough to guarantee that it has those causal powers." (Searle, 1980b:452)

A better argument or a more accurate piece of logic might perhaps prove him wrong, but no experimental finding could, I believe, overturn his conclusion. Most commentators, including Danto (1980), Eccles (1980) Libet (1980), Maxwell (1980), Natsoulas (1980), Obermeier (1983), Puccetti (1980) and Ringle (1980) agree with this conclusion. However, there are those such as Dennett (1980), Minsky (1980) and Moor (1988) who adopt a naive empiricism, and claim that the progress of science may one day show how a machine can acquire intentionality. But they do not explain how it is

that an experiment could establish the truth of a faulty argument.

The last part of Searle's paper is taken up with his attempt to answer the question of why the claims of strong CS, or strong AI as he phrases it, must necessarily be mistaken. He appeals to the concept of intentionality, and he puts forward in support of his position a "monist-interactionist" (1980b) view of cognition. He explains the impossibility of strong CS by trying to demonstrate that an unbridgeable gulf exists between the operation of a machine and the functioning of the brain. These are experimental rather than conceptual matters, of course, and Searle's critics have not failed to try to undermine his explanations by reference to empirical data.

Conclusion

Most hostile comments upon the Chinese room argument attack it not directly but by criticising Searle's attempt to justify his conclusion. Searle's critics raise objections based upon the nature of cognition, about which there has been and is much controversy, and their arguments bear upon weak as much as upon strong CS. The attack which has been made upon both strong and weak CS by Hubert Dreyfus is the subject of the next chapter. Dreyfus approaches the subject of artificial intelligence from a phenomenological point of view, and his critique casts a great deal of light upon the nature of human thinking and its relationship to machine computation. I shall therefore postpone a discussion of Searle's views on cognition, which underpin his Chinese room argument, until Chapter 5 of this text.

Chapter 4. THE CRITIQUE OF HUBERT DREYFUS

Alfred North Whitehead, who was co-author with Bertrand Russell of 'Principia Mathematica', has observed in his 'Adventures of Ideas' that:

> "(Plato's) later dialogues circle round seven notions, namely - The Ideas, The Physical Elements, The Psyche, The Eros, The Harmony, The Mathematical Relations, The Receptacle. I mention them because I hold that all philosophy is in fact an endeavour to obtain a consistent system out of some modification of these notions." (Whitehead, 1933:354)

That is as much as to say that more than 2000 years of Western philosophy amounts to little other than a series of footnotes on Plato.

The work of Hubert Dreyfus goes some way to corroborate Whitehead's remark, for Dreyfus has much to say about Plato in his influential book 'What Computers Can't Do' first published in 1972. Dreyfus is interested in the legacy of Plato not on account of the metaphysical doctrines it contains, but rather as a foil and counter-example to his own epistemological point of view. Furthermore, he is a critical rather than an admiring commentator. Plato, for Dreyfus, is the originator of the view that thought can only be described as knowledge if it can be stated explicitly.

> "Since the Greeks invented logic and geometry, the idea that all reasoning might be reduced to some kind of calculation - so that all arguments could be settled once and for all - has fascinated most of the Western tradition's rigorous thinkers. Socrates was the first to give voice to this vision. The story of artificial intelligence might well begin around 450 BC when (according to Plato) Socrates demanded of Euthyphro, a fellow Athenian who, in the name of piety, is about to turn in his own father for murder: 'I want to know what is characteristic of piety which makes all actions pious... that I may have it to turn to, and to use as a standard whereby to judge your actions

and those of other men." Socrates is asking Euthyphro for what a modern computer theorist would call an 'effective procedure', 'a set of rules which tell us, from moment to moment, precisely how to behave'.
Plato generalised this demand for moral certainty into an epistemological demand. According to Plato, all knowledge must be stateable in explicit definitions which anyone could apply. If one could not state his know-how in terms of such explicit instructions - if his knowing how could not be converted into knowing that - it was not knowledge but mere belief."
(Dreyfus, 1979:67-68)

Dreyfus goes on to trace the progress of the interpretation of knowledge as something necessarily explicit through the work of Galileo, Hobbs and Leibniz to Boole and Babbage. The development of computers in the 1940's brought this strand of Western thought to its culmination.

> "For, since a digital computer operates with abstract symbols which can stand for anything, and logical operations which can relate anything to anything, any digital computer (unlike an analogue computer) is a universal machine. First, as Turing puts it, it can simulate any other digital computer....Second, and philosophically more significant, any process which can be formalised so that it can be represented as a series of instructions for the manipulation of discrete elements, can, at least in principle, be reproduced by such a machine.
> But such machines might have remained overgrown adding machines had not Plato's vision, refined by two thousand years of metaphysics, found in them its fulfilment. At last here was a machine which operated according to syntactic rules on bits of data. Moreover, the rules were built into the circuits of the machine. Once the machine was programmed there was no need for interpretation; no appeal to human intuition and judgement. This was just what Hobbs and Leibniz had ordered, and Martin Heideggar appropriately saw in cybernetics the culmination of the philosophical tradition." (Dreyfus, 1979:72)

But if the power and generality of the new machine was to be realised in practice then some "technique for converting any practical activity such as playing chess or learning a language into a set of instructions" (Dreyfus, 1979:74) was

needed. Some of the early artificial intelligence programs appeared to be examples of such a technique.

> "With digital computers solving such problems as how to get three cannibals and three missionaries across a river without the cannibals eating the missionaries, it seemed that finally philosophical ambition had found the necessary technology: that the universal high-speed computer had been given the rules for converting reasoning into reckoning.
> The field of research, dedicated to using digital computers to simulate intelligent behaviour, soon became known as 'artificial intelligence'." (Dreyfus, 1979:77)

We have seen earlier that John Searle defines strong AI as the reproduction of human intelligence by means of a computer. He proceeds to prove, I think convincingly, that what he calls strong AI is impossible. For Dreyfus, however, artificial intelligence means not the reproduction of intelligence but the simulation of intelligent behaviour. This is the undertaking which Searle refers to as weak AI. I think that the two activities are more usefully described as strong and weak CS respectively. But regardless of terminology, it is clear that Searle and Dreyfus are addressing themselves to different, if related, topics.

Dreyfus's criticism of artificial intelligence is more far-reaching than Searle's. For Dreyfus, the point to be refuted is not that computers can think, but that thought itself is a species of computation. For, he argues, if reasoning could be converted into reckoning by becoming mechanised, then far-reaching consequences for our ways of seeing everything will ensue.

> "Aristotle defined man as a rational animal, and since then reason has been held to be of the essence of man. If we are on the threshold of creating artificial intelligence we are about to see the triumph of a very special conception of reason. Indeed, if reason can be programmed into a computer, this will confirm an understanding of man as an object, which Western thinkers have been groping toward for

two thousand years but which they only now have the tools to express and implement. The incarnation of this intuition will drastically change our understanding of ourselves. If, on the other hand, artificial intelligence should turn out to be impossible, then we will have to distinguish human from artificial reason, and this too will radically change our view of ourselves. Thus the moment has come either to face the truth of the tradition's deepest intuition or to abandon the mechanical account of man's nature which has been gradually developing over the past two thousand years."
(Dreyfus, 1979:78-79)

Dreyfus's proclaimed intention to refute the notion that calculation is the same thing as thinking is therefore much more than an attempt to undermine artificial intelligence. It is, for him, the final battle in the war between man as a rational mechanism and man as a free intellect. And the casus belli is artificial intelligence.

Dreyfus divides his task into three distinct stages. He begins by claiming that the promise of artificial intelligence remains unrealised, and will never in fact be fulfilled, because of intractable methodological difficulties. He takes as examples of unrealised promise the early of attempts at machine translation of natural language, problem solving and pattern recognition.

Unfulfilled Promise - Machine Translation

One of the early machine translation programs was developed by a team at the National Physical Laboratory in Teddington lead by A[ ] Szanser. Work began in 1959 and by 1966 it had achieved an assessment of "slightly less than good" (Szanser, 1967) The NPL program, like others which appeared during the late 50's and early 60's, was based upon a much oversimplified procedure which can be illustrated diagrammatically as;

Source language ------> Target language.

The NPL program made use of a large dictionary of technical

terms in English and Russian which had been compiled at Harvard University (Oettinger, 1955). The hope was that by selecting matched English and Russian words, and then ordering them according to syntactic rules, a high quality machine translation in the direction of either language would result, at least of technical texts. But, as was suspected at the time and is known now, the complexity of natural languages will quickly overwhelm such a primitive scheme. There are four main defects in the simple NPL algorithm and other similar machine translation programs.

In the first place, the source language may employ a single word with two meanings while the target language expresses each meaning with a separate word. For example, the two meanings of the English word 'pen', used of an implement for writing with ink and also to denote a small enclosure, is represented in French by 'plume' and 'parc' respectively. Therefore, in order to select the correct French word when translating an English text, and vice versa, it is necessary to know the meaning as well as the syntactical description of the words in a translation program's dictionary. This difficulty in making a translation is referred to as lexical ambiguity.

A corresponding difficulty, known as grammatical ambiguity, results from the fact that an ambiguous sentence in the source language may be represented by several different grammatical structures in the target language. To say in English that 'He follows Darrida' could be correctly translated into French either as 'Il suit Darrida' or 'Il souscrire au Darridism'. The intransitive English verb in the first case is translated into a French intransitive verb, while in the second case a transitive verb is required in the translation. As with the lexical ambiguity of individual words, the meaning of the English sentence must be disambiguated before a corresponding French construction can be found.

When a connective or a pronoun serves the grammatical purpose of pointing back to something which has been said, it is described as anaphoric. But a difficulty, in the context of machine translation, is that it may well be capable of referring to more than one preceding word or clause. For instance, the pronoun 'it' in the last sentence may refer to the nouns 'connective' or 'pronoun' in the sentence before, or to the noun phrase 'a difficulty for machine translation' in the same sentence. References to objects which occur later than the pronoun or which lie outside the text altogether are known as cataphoric and exophoric respectively (Halliday & Hasan, 1976). The problem of referential cohesion occurs frequently in linguistic analysis. Texts containing these types of reference can only be translated correctly if the exact referent is known, and this is of course a question of semantics rather than syntax.

A particularly intractable difficulty in all language translation, both for a human linguist and a computer, is dealing with idioms. An idiom is a linguistic construction approved by usage whose significance differs from its grammatical meaning. Elaine Rich (1983) observes that;

> "An idiom in the source language must be recognized and not [mechanically] translated directly into the target language. A classic example of the failure to do this is illustrated by the following pair of sentences. The first was translated into Russian, and the result was then translated back to English, giving the second sentence:
>
> 1. The spirit is willing but the flesh is weak.
> 2. The vodka is good but the meat is rotten.

It is evident that a third element must be added to the simple diagrammatic representation of machine translation if it is to become a method capable of overcoming the lexical, grammatical, referential and idiomatic obstacles.

The diagram must be expanded into the form;

Source language ---> Semantic encoding ---> Target language

Dreyfus agrees with the need for a semantic component in a workable automatic machine translation algorithm. He goes on to argue that fully automatic high quality translation, often abbreviated to FAHQT, is impossible because meaning cannot in fact be represented by any formal symbolic system. Furthermore, he applies his comments on semantic processing to problem solving and pattern recognition programs as well as attempts at machine translation. His reasons for this assertion, which makes up the second part of his analysis of weak CS, are interesting and, I think, cogent.

Unfulfilled Promise - Computer Chess

It is possible, in principle, to solve many problems by enumerating every possibility and then attempting to select the best solution from the list. The problem to which this exhaustive process of 'counting out' has most often been applied is the game of chess. The conduct of a chess player is completely rule bound but the immense number of possible board states, some 10^{120}, means that playing and winning the game is not a simple matter. The combination of strict formality and enormous extent has made chess a favourite vehicle for experiments in problem solving by computer.

The algorithm upon which all modern chess playing programs are based is the 'lookahead-evaluate-minimax' model that was first proposed by Claude Shannon in 1950. Using this method the machine proceeds by searching ahead from the current position along a branching tree of possible moves. The result of each possibility is recorded on a numerical scale and the move with the highest score is selected and made. It has been found that in chess there is an average of about 35 lookahead branches for each board state. Other things being equal, therefore, the lookahead tree would

grow by powers of 35. Advances in the design of chess playing programs have taken the form of finding ways to prune the lookahead tree, and the recent Cray Blitz program (Hyatt et al, 1986) grows by a power of only about 8 for each ply.

By 1967 the MacHACK chess program had achieved the standard of an average club player (Greenblatt, Eastlake & Crocker, 1967). More recently the Cray Blitz program has beaten players of National Master standard and is rated at about 2300 on the United States Chess Federation scale (Hyatt et al, 1986). Although progress in the techniques available for pruning the lookahead tree have helped to reach what is, by the standard of ordinary mortals, a very high rating the most important factor has been an enormous increase in the power of computers. MacHACK ran on a DEC PDP-6 capable of 2×10^5 arithmetic operations per second, while the Cray X-MP two processor machine upon which the 1983 version of Cray Blitz was implemented has a speed of 10^{12} arithmetic operations per second, a five million-fold increase in speed of processing.

Chess playing programs show clearly the effects of a fundamental difficulty facing any problem solving routine which relies on exhaustively counting out the possibilities, which is the fact that the necessary number of computations increases exponentially with the size of the problem. This phenomenon is known in computer jargon as 'the combinatorial explosion'.

> "Chess, however, although decidable in principle by counting out all possible moves and responses, presents the problem inevitably connected with choice mazes: exponential growth. Alternative paths multiply so rapidly that we cannot even run through all the branching possibilities far enough to form a reliable judgement as to whether a given branch is sufficiently promising to merit further exploration." (Dreyfus, 1979:101)

Few people would disagree with Dreyfus when he claims that an attempt to simulate cognition, even using a very large computer to count out all the possibilities, is doomed to fail in the case of any but trivially small problems. The processing power of the human brain, which is about 10^{18} arithmetic operations per second, is some million times faster than a supercomputer. However, despite the remarkable power of the chemical computer housed within his skull, a human player is no more able to count out all possible moves in a game of chess than is Cray Blitz. The number of possibilities remains much too large for the brain as it is for the supercomputer.

Unfulfilled Promise - Pattern Recognition

Nearly everything that we do involves, if it does not actually follow from, an act of perception. The word 'perceive' derives ultimately from the Latin 'capere' meaning 'to lay hold of', and the modern usage similarly implies the active acquisition of information or knowledge. The enterprise of "making machines do things that would require intelligence if done by men" therefore leads naturally to experiments in computer perception. A sub-division of this undertaking is the attempt to program a computer to recognise patterns.

One might hope that pattern recognition would be easier to do than perception by computer because the pre-existence of the pattern to be recognised restricts the scope of the problem. Perception in the general sense leaves open the question of what it is that is to be perceived. But it quickly emerges that pattern recognition is more difficult than it at first seems. In his role of the gadfly of the artificial intelligence community, Hubert Dreyfus has not been slow to point out some of these difficulties and their consequences.

> "A computer must recognise all patterns in terms of specific traits. This raises problems of exponential growth which human beings are

able to avoid by proceeding in a different way. Simulating recognition of even simple patterns may thus require recourse to each of the fundamental forms of human 'information processing' discussed this far. And even if in these simple cases artificial intelligence workers have been able to make some headway with mechanical techniques, patterns as complex as artistic styles and the human face reveal a loose sort of resemblance which seems to require a special combination of insight, fringe consciousness, and ambiguity tolerance beyond the reach of digital machines. It is no wonder, then, that work in pattern recognition has had a late start and an early stagnation." (Dreyfus, 1979:120)

I think that much of what Dreyfus says here is perfectly true. It is impossible to see how a computer, which has no human or personal history nor is possessed of intentionality, can ever be programmed to recognise a Tinoretto, to distinguish between a string quartet by Hayden and another by Mozart, or to pick out a particular face in a crowd. He could also have included in his list of impossibilities the perception of significant patterns in a game of chess. But the passage that I have quoted above displays the characteristic weakness, as well as the cogency, of Dreyfus's line of argument.

He correctly points out the impossibility of doing something difficult by imitating human methods of thought and he goes on to imply, erroneously I think, that simpler tasks of a similar type are therefore also impossible to carry out. This follows from his failure to distinguish between the imitation and the mere simulation of cognition. Just as he was lead in 1965 to dismiss the possibility of expert computer chess by the fact that:

"situations will always occur in which the machine cannot pursue the chain of moves which contains the winning combination; thus, there will always be games that people can win and machines cannot." (Dreyfus, 1965)

So he concludes that all pattern recognition research is in a state of stagnation because no computer could recognise a painting in a museum. But more humble lines of pattern recognition research are not at a halt. The last twenty years has, for example, seen a great deal of progress in the automatic recognition of printed and typewritten characters. The topic of character recognition is often abbreviated as CR.

The most simple type of CR machine makes use of magnetic printing ink. Serial numbers on bank cheques, for example, when read using a magnetic scanner, produce a characteristic waveform. The waveform can be compared very easily with a stored bank of waveforms representing the elements of the font used in printing the cheque forms, and the result output to a computer screen or file. Magnetic CR machines work very fast, but little intelligence, real or artificial, is required of them.

The reverse is true of programs written to recognise handwriting. The great variety of graphical forms which appears in handwritten texts is often sufficient to confuse the human eye. But despite great differences in size, regularity, shape and connectedness in cursive script it is possible to achieve recognition rates as high as 97% (Davis & Lyall, 1986). This is done by extracting the elements of which a character is composed and comparing them with a database containing the set of possible graphical strokes (Eden, 1968). The advantage of this method is that a stroke which does not precisely match the pattern, a crossing of a 't' which is not accurately horizontal for example, can be recognised as a rotated crossing stroke rather than being rejected as outside the set. Furthermore, a character can be recognised correctly even if one of its elements is missing provided that the combination of characters that it does exhibit is possessed by no other character. Cursive script recognition, or CRS, programs have decipherment

capabilities which, were they exhibited by human readers, would be considered to demonstrate intelligence.

An intermediate position on the artificial IQ scale is occupied by the optical character reading, or OCR, machines. The type now frequently seen on office desks can usually read only a restricted number of printed character fonts. The DEST PC machine, for instance, can read a piece of text provided that it is printed on one of 12 daisywheel typefaces or 9 produced on a dot-matrix printer. Recognition is carried out by comparing the character as it is read with a prepared database of standardised character forms. These machines are accurate within their design limits, but they are somewhat restricted in their capabilities. The DEST PC will fail to recognise text that has, for example, been enlarged or reduced in a photocopier. This machine is a long way from displaying the "special combination of insight, fringe consciousness and ambiguity tolerance" which Dreyfus supposed to be indispensable, but it is nevertheless a very useful device to anyone who has to handle text.

However, the type of OCR machine that possesses a database which is trainable rather than standardised can perform at a level that is a convincing imitation of intelligence. The KDEM system (Hockey & Scott, 1981) employs a vertical slit optical reader by means of which an enlarged image of a character is sent to the screen, together with the system's guess as to which alphanumeric character it is. The operator confirms or corrects the system's judgement, whereupon the character is entered into the database. The key to the effectiveness of the system is that a character is stored not as a single complete image, but rather as an assembly of graphical features. This enables subsequently read characters to be identified even if they differ in some ways from any of the database records. An 'i' without a dot, for example, will be read correctly because the program has been supplied with the fact that no other charac-

ter in the Roman alphabet possesses only a short perpendicular stroke which springs from the baseline. The result of the process of training is that the system soon ceases to call for confirmation of the identity of a character, and it is able to go ahead and read the rest of the document without help from the operator. If the ability to accept training is held to be a feature of intelligence, then the KDEM system can lay claim to the artificial variety of that faculty.

The Human Situation

It appears, on Dreyfus's assessment, that a computer will never be able to translate text, play chess or recognise patterns. He points to our human ability to recognise a Degas, or to translate Dante, and one must concede in his favour that a grandmaster can still defeat even a program as powerful as the Cray Blitz. Wherein, then, lies the difference between computing machines on the one hand and, on the other, our own selves as cognitive beings? This question brings us to the last part of Dreyfus's argument against weak CS, which he sees as the final incarnation of the Platonic tradition.

The last stage of Dreyfus's attempt to rebut the claims of researchers in artificial intelligence consists of giving an alternative account of how it is that humans display actual intelligence. This he does from a phenomenological point of view.

Dreyfus the phenomenologist approaches the question of human performance by examining what a chess player thinks he is doing while he is conducting his game. The American chess master Eliot Hearst attributes the skill of the human player not to the number of moves he can foresee, but to his ability to judge the significance of the pattern and structure which a game displays.

> "Apparently the master perceives the setup in large units, such as pawn structure or cooperating pieces, and can even decide which side has the advantage. When he does make an error, it is very often one of putting a piece on a very desirable square for that type of position." (Hearst, 1967:35)

That is to say, the player begins by seizing upon the overall pattern of the game. He identifies the places in which he and his opponent is strong or weak, he recalls previous games in which he has faced a similar situation, and he exploits what he knows about his opponent's style of play. Only then does he count out the possible moves. This type of problem solving Dreyfus calls 'zeroing-in'. Zeroing-in is based upon intuition and interpretation, not upon calculation, and it works from the general to the particular. It is, in fact, the converse of 'counting out'.

Zeroing-in works for us because we can use the context of our situation to judge the significance of things. A piece is vulnerable in the context of a particular state of the board, and a move is made because of later moves that it may facilitate. But the context of the board is influenced by the context of experience of the two players, and that in its turn is partly a function of the state of chess culture. And the culture of chess is a part of general culture, which exists in history. It is impossible to provide a computer with all that is needed to zero in on a problem because the sequence of the layers of context forms an infinite regress.

> "Thus, for example, to pick out two dots in a picture as eyes one must have already recognised this context as a face. To recognise this context as a face one must have distinguished its relevant features such as shape and hair from the shadow as and highlights, and these, in turn, can be picked out as relevant only in a broader context, for example, a domestic situation in which the program can expect to find faces. This context too will have to be recognised by its relevant features, as social rather than, say, meteorological, so that the program selects as significant the people

> rather than the clouds. But if each context can be recognised only in terms of features selected as relevant and interpreted in terms of broader context, the AI worker is faced with a regress of context." (Dreyfus, 1979:289)

Human beings, as sentient creatures, can cut short the regress of context because they have a personal point of view from which to decide what aspects of the context are relevant. As Ludwig Wittgenstein puts the matter, "What has to be accepted, the given, is - so one could say - forms of life." (PI II, 226). But a computing machine, which can do no more than manipulate a formalism of symbols, would need an infinite amount of information if it were to be able to arrest the infinite regress of the problem context.

Conclusion

In the previous chapter I have given an account of Searle's Chinese Room experiment in which, in my opinion, he disposes of the notion that a computer can be said to be thinking just because it is able to manipulate symbols. Dreyfus's attack on artificial intelligence research has been described in this chapter. He dismisses artificial intelligence not just because it cannot instantiate thinking but because it cannot, he asserts, simulate thinking either. But there are many parallels between the two analyses offered by Searle and Dreyfus. In the next chapter I attempt to compare their arguments in such a way as to come to a conclusion about the status of cognitive simulation as a sub-division of the topic of artificial intelligence.

Chapter 5. SEARLE, DREYFUS AND THE SIMULATION OF COGNITION

Some conscious states of mind occur without direct reference to the outer world. For example, voluntary movement of the body, the sensation of a painful tooth or the exercise of memory are states of mind that are directed inwards upon oneself. They are complete without reference to external reality, and are in a sense intransitive. But other states imply the relevance of something in the outside world. One may believe that something is not so, one may wish something to be so, or one may be afraid of something. These states, which are directed at an external object or set of independent circumstances, are known as intentional states.

Intentional States

The fact that intentional states are internal mental phenomena rather than percepts is shown by the fact that an intentional state can be directed at an unknown or fictitious object.

> "for a large number of Intentional states I can have the state without the object or state of affairs that the Intentional state is directed at even existing at all: I can believe that the king of France is bald even if, unknown to me, there is no king of France; and I can hope that it will rain even if it doesn't rain." (Searle, 1979:74)

Searle has capitalised the word Intentional in the text of his paper in order to distinguish the philosophical use of the term from its meaning in ordinary usage of "done on purpose". In the absence of intentional states, in this technical sense of the word, the mind would be isolated from its environment and thinking would be impossible. However, the essential point, in the context of artificial intelligence, about the notion of intentionality is that it involves a two-term relationship. There has to be the object or circumstance which is referred to, and there must be a conscious being capable of directing attention to

those things. Intentionality is the name for the logical link which connects the two sides of the relation.

However, in the notion of meaning as proposed by Schank (1975) as part of his theory of conceptual dependency there is no room, nor any need, for intentional states.

> "We define an interlingua as a representation of meaning of natural language that does not involve any of the words of the language. This representation of meaning should be extractable from any language and capable of being generated into any other language.
> In order to try to develop an interlingual representation it is necessary to reject the idea that thought does not exist independent of language. We thus propose that language has words which name thoughts and that thoughts can be separated. Thus we assume that any language can be translated into any other language." (Schank, 1975:8)

That is to say, meaning resides in the words themselves, independently of the linguistic relation of an observer to those words.

The semantic autonomy of the object is, no doubt, required if semantic processing by computer is to become a reality. But I think that Searle has placed his finger accurately upon the feature that most clearly distinguishes thought from computation. He explains the result of the Chinese room experiment as following from the necessarily intentional character of thought, and that this intentionality is possessed by the programmers, not the computer.

> "formal symbol manipulations by themselves don't have any intentionality; they are quite meaningless; they aren't even symbol manipulations, since the symbols don't symbolise anything. In the linguistic jargon, they have only a syntax but no semantics. Such intentionality as computers appear to have is solely in the minds of those who program them and those who use them, those who send in the input and those who interpret the output." (Searle, 1980a:422)

In short, intentionality, which is indispensable for thought, is a two-term relationship between a conscious being and the external world. Conceptual dependency, on the other hand, is based upon single terms each of which is held to characterise something in the world. In short, conceptual dependency attempts to replace a two-term intentional relationship with a single-term property attribution. A computer may indeed be programmed to manipulate the components of a Schankian script, but Searle's analysis of intentionality shows why we should not mistake this for thought.

Although Searle claims (1980b:454) that "I am concerned....only incidentally with the 'mind-brain problem'.", he does in fact devote much of the space in his paper to what he calls the causal powers of the human brain. He has shown that intentionality is indispensable to thought, and he wants to go on to demonstrate that only brains can display intentionality. Thus he is led, despite a disclaimer, to say something about the vexed problem of the relationship between the brain and the mind.

It is possible to argue, as did Rene Descartes, that the brain as a part of the body is a physical object while thought is immaterial and takes place in a sphere remote from space and time where the laws of physics do not apply. A person's bodily life, from this point of view, takes place in the physical world and is external to him, while those things that occupy his mind constitute his internal life. There are a number of difficulties with Cartesian dualism, the most intractable of which is the problem of accounting for the interactions of the mind and the body. David Hume was the first thinker to acknowledge this completely, and he found himself in consequence driven to embrace a completely solipsistic view of the world. If one's mind is indeed remote in space and time from one's brain, then the only thing of which the possessor of a mind can have knowledge is that same mind.

Most thinkers other than Hume have felt it necessary to try to modify the doctrine of Cartesian dualism so as to evade solipsism. Gilbert Ryle in his 'The Concept of Mind' (1949) made a comprehensive attack of what he called "the ghost in the machine", by which he meant the idea that our machine-like bodies are inhabited by an immaterial ghost-like mind. He advanced a monist conception of the relation of brain and the mind. According to Ryle, to speak of a mind is to discuss someone's propensity to do things.

> "To talk of a person's mind is not to talk of a repository which is permitted to house objects that something called 'the physical world' is forbidden to house; it is to talk of the person's abilities, liabilities, and inclinations to do and undergo certain sorts of things, and of the doing and undergoing of these things in the ordinary world. Indeed, it makes no sense to speak as if there could be two or eleven worlds. Nothing but confusion is achieved by labelling worlds after particular avocations. Even the solemn phrase 'the physical world' is as philosophically pointless as would be the phrase 'the numismatic world', 'the haberdashery world', or 'the botanical world." (Ryle, 1949:190)

To suppose otherwise, he says is to make the "category mistake" of inventing a thing for whose reality the only evidence is the existence of a word. Ryle's critique of dualism is, like Descarte's proposal, conceptual, and neither thinker makes any appeal to empirical evidence to support his position.

For his part, Searle produces in his paper a sketch of a position on the brain-mind problem which rests a little awkwardly between the views of Ryle and Descartes. He claims that there is a necessary biological relationship between cognition and the kind of creatures we are.

> "It is not because I am the instantiation of a computer program that I am able to understand English and have other forms of intentionality (I am, I suppose, the instantiation of any

> number of computer programs), but as far as we know it is because I am a certain sort of organism with certain biological (i.e. chemical and physical) structure, and this structure, under certain conditions, is causally capable of producing perception, action, understanding, learning, and other intentional phenomena. And part of the present argument is that only something that had those causal powers could have that intentionality." (Searle, 1980a:422)

It seems to me that Searle has quite unnecessarily muddied the water by appealing to what he claims is accepted empirical knowledge. We have in recent years learned quite a lot about how nerve impulses in the brain are transmitted, and we know where in the brain some cognitive processes are centred. But the manner in which thoughts and feelings are related to, or result from, the physiological working of the brain remains a deep and fascinating mystery. Neurobiologists have been able to push back the frontier of our understanding of how the brain works, but the mind-brain problem has retreated in step with the advance of science. One therefore stands on very shaky ground when one tries to put neurobiology to use in epistemological discussions. Searle would have saved himself some inconclusive skirmishes with empirically oriented critics such as Fodor (1980) Hofstadter (1980) and Minsky (1980) had he recognised this. But, it seems to me, Searle's case stands by its logical coherence rather than by virtue of any empirical buttressing. One may accept the conclusion of his argument while regretting his ill-judged excursion into the biology of the brain.

The fact that all formal symbol manipulation systems suffer from a complementary pair of related limitations, which are the combinatorial explosion in the direction of counting out and infinite regress in the opposite direction of contextual assessment, is Dreyfus's central insight and it is the thought that underpins the whole of his book. I think he is correct to conclude from this discovery that a computer can never simulate thinking, and that the project

of weak CS is impossible. A computer that is instructed to count out all possibilities will never complete the task, while if it is to zero-in upon the problem it will require an infinite amount of contextual information.

Searle has shown that strong CS, in the sense of the reproduction of cognition by computer, is also impossible. But he does not attempt to extend his disproof to weak CS, or to artificial intelligence in general.

> "He (Schank) thinks I want 'to call into question the enterprise of AI.' That is not true. I am in favour of weak AI, at least as a research program." (Searle, 1980b:453)

Phenomenology

Dreyfus, however, extends his argument against weak CS to embrace any attempt to use computers in cognitive studies, and indeed he takes it so far as to reject the entire Western analytic attitude towards personal experience.

> "We have seen that what counts as 'a complete description' or an explanation is determined by the very tradition to which we are seeking an alternative. We will not have understood an ability, such as the human mastery of natural language, until we have found a theory, a formal system of rules, for describing this competence. We will not have understood behaviour, such as the use of language, until we can specify that behaviour in terms of unique and precisely definable reactions to precisely defined objects in universally defined situations. Thus, Western thought has already committed itself to what would count as an explanation of human behaviour. It must be a theory of practice, which treats man as a device, an object responding to the influence of other objects, according to universal laws or rules. But it is just this sort of theory, which, after two thousand years of refinement, has become sufficiently problematic to be rejected by philosophers both in the Anglo-American tradition and on the Continent. It is just this theory which has run up against a stone wall in research in artificial intelligence. It is not some specific explanation, then, that has failed, but the whole conceptual framework which assumes that an explanation of human

behaviour can and must take the Platonic form, successful in physical explanation; that situations can be treated like physical states; that the human world can be treated like the physical universe. If this whole approach has failed, then in proposing an alternative account we shall have to propose a different sort of explanation, a different sort of answer to the question 'How does man produce intelligent behaviour?' or even a different sort of question, for the notion of 'producing' behaviour instead of simply exhibiting it is already coloured by the tradition. For a product must be produced in some way; and if it isn't produced in some definite way, the only alternative seems to be that it is produced magically." (Dreyfus, 1979:232)

The point of view from which Dreyfus conducts his critique of artificial intelligence is that of the phenomenological school of thought. This many-syllabled word denotes a method of philosophical enquiry whose twentieth century form was initiated by Edmund Husserl with the publication in 1913 of his 'Ideen zu einer reinen Phanomenologie und phanomenologishen Philosophie'. The central idea upon which phenomenology is founded is Husserl's assertion that a study of meaning must rest upon insight rather than, as empiricists would have it, upon generalisations from experience. Husserl maintains that no distinction can be made between perception and what is perceived, and that objects are correlated with states of mind. For Husserl consciousness was all. Husserl's ideas were developed principally by Martin Heidegger with his 'Sein und Zeit' of 1927 and by the French philosopher Maurice Merleau-Ponty whose 'Phenomenologie de Perception' appeared in 1945. The later thought of Ludwig Wittgenstein has many parallels with, and some important differences from, the phenomenological mainstream.

It is at once apparent that phenomenology is in conflict with the entire Western tradition of analytic philosophy, and in particular with the scientific view of reality. The scientific viewpoint is based upon the assumption that there exists an external physical reality and that human

thought occurs with reference to it. It is precisely this attitude to which Husserl attributes all our troubles and confusion.

> "Husserl contends that in striving to build up an objective picture of reality, scientific practice has progressively cut off subjective experience from the life-world to such an extent that Western man is in a permanent state of crisis, i.e. he feels that science is his only source of facts and loses consequently his lived relations to the historical and social reality of life. In brief, Western man is deprived of the immediate evidence of his world considered as the realm of significant relations to objects and to his fellow men, and is condemned to rely on intermediate abstract constructs: the life-world is concealed by the transcendental act of scientific elaboration." (Thines, 1987:327)

Few people whose reflective capacities are not completely atrophied can fail to see the force of Husserl's contention. A purely calculative attitude, characterised in French by the adjective 'Cartesian', dominates much of modern life and it brings alienation as often as enlightenment in its train. The world does indeed turn to stone when it is subjected to the stare of an exclusively scientific Minerva. But only an over-riding impulse to be completely consistent in all of one's thoughts can drive one to place personal experience in diametric opposition to scientific knowledge. Fortunately, scientific rationality and human experience are only obliquely in conflict and one need not, I think, be either inflexibly scientific nor steadily introspective about everything.

Hubert Dreyfus came to the study of artificial intelligence from a phenomenological background, and he has a characteristically wary attitude to science.

> "if my favourite thinkers (who might be called antiphilosophers) were right, the new computer approach should not work, based as it was on using programs or rules to impart 'knowledge' to machines. So I confidently continued to

teach Merleau-Ponty's claim that perception and understanding are based in our capacity for picking up not rules, but flexible styles of behaviour. For example, someone who knows how to drive a car with a shift on the steering column can easily transfer the skill to a shift on the floor, even though the rule describing the sequence of movements required would be very different. Explaining Heideggar, I continued to assert that we are able to understand what a chair or a chair or a hammer is only because it fits into a whole set of cultural practices in which we grow up and with which we gradually become familiar. As I taught I wondered more and more how computers, which have no bodies, no childhood, and no cultural practices, but are disembodied, fully formed, nonsocial, purely analytic engines, could be intelligent at all. Clearly, if the word I was getting from the robot factory was right, then the antiphilosophers I was teaching were wrong. I realised that if I was to go on teaching those antiphilosophers to skeptical students, whom I now thought of as the heirs of Plato, Kant and Husserl, I had better find out just how intelligent computers were and how intelligent they were likely to become." (Dreyfus & Dreyfus, 1986:5)

When Dreyfus turns his eye upon artificial intelligence he sees in it the culmination of the analytic de-humanisation of the Western world view. While this may seem to be rather a heavy burden to be borne by a mere sub-division of computer science, I think that it is important to take note of what he says. Dreyfus speaks as a philosophical historian, as a well informed critic of artificial intelligence and as a lucid spokesman for an anxiety about computers and computation that is widespread in the educated public.

> "During the past two thousand years the importance of objectivity; the belief that actions are governed by fixed values; the notion that skills can be formalised; and in general that one can have a theory of practical activity, have gradually exerted their influence in psychology and in social science. People have begun to think of themselves as objects able to fit into the inflexible calculations of disembodied machines: machine for which the human form-of-life must be analysed into meaningless facts, rather than a field of concern organised by sensory-motor skills. Our risk is not the

> advent of superintelligent computers, but of subintelligent human beings." (Dreyfus, 1979:280)

The Limits of Phenomenology

I am not equipped to adjudicate upon Dreyfus's attempt to overturn nearly everything that has been thought in the West since the time of Plato. However, I do feel able to say that, so far as artificial intelligence is concerned, his argument against weak CS is sound, and that there is indeed a wide ontological gulf lying between our minds and our machines. But I also think that Dreyfus underestimates the scope and complexity of "the enterprise of AI", and that he fails to recognise that artificial intelligence is not an exclusively conceptual undertaking. He is on weak ground when, embarking upon the last section of his task, he attempts to consign all parts of the subject of artificial intelligence to the paper shredding machine.

There is a discontinuity between the analysis that Dreyfus offers of machine thinking and the predictions he makes about the future of artificial intelligence. He has identified, I think correctly, the reason why FAHQT is impossible, which is the problem of the infinite regress of context. He then goes on to say,

> "The foregoing considerations concerning the essential role of context awareness and ambiguity tolerance in the use of a natural language should suggest why, after the success of the mechanical dictionary, progress has come to a halt in the translating field. Moreover, since, as we have seen, the ability to learn a language presupposes the same complex combination of human forms of 'information processing' needed to understand a language, it is hard to see how an appeal to learning can be used to bypass the problems this area must confront." (Dreyfus, 1979:111)

It follows from the impossibility of FAHQT, Dreyfus maintains, that every language translation program must be useless. But to say this is to needlessly circumscribe the topic artificial intelligence. Peter Sell's definition of

the subject which is given in Chapter 2 points out that "intelligent artifacts" and "models of human cognition" are equally important in artificial intelligence.

Dreyfus is so preoccupied with what he thinks are faulty computer models of cognition that he fails to attach any importance to intelligent artifacts, in the field of machine translation or elsewhere. But the development of translators' assistant programs (NLP, 1984), which facilitate rather than conduct the process of translation, would not have occurred had Dreyfus's condemnations encompassed the whole truth about artificial intelligence. His assessment of the future of computer chess is similar to his predictions about machine translation. The human chess player, Dreyfus says,

> "sees that his opponent looks vulnerable in a certain area (just as one familiar with houses in general and with a certain house sees it as having a certain sort of back), and zeroing in on this area he discovers the unprotected Rook. This move is seen as one step in a developing pattern.
> There is no chess program which even tries to use the past experience of a particular game in this way. Rather, each move is taken up anew as if it were an isolated chess problem found in a book. This technique is forced upon programmers, since a program which carried along information on the past position of each piece would rapidly sink under the accumulating data. What is needed is a program which selectively carries over from the past just those features which were significant in the light of its present strategy and the strategy attributed to its opponent. But present programs embody no long-range strategy at all." (Dreyfus, 1979:105)

This is all perfectly true, and it is indeed hard to see how it will ever be possible for a computer to be programmed to assess the state of the board in the same way as a human player. But despite this, computer programs can be formidable and effective opponents for even highly skilled players. Dreyfus himself discovered this when, despite his

early claim that "Still no chess program can play even amateur chess" (1965), he was beaten by MacHACK (Hayes & Levy, 1976:6).

It has been pointed out by the Canadian psychologist Zenon Pylyshyn that Dreyfus's phenomenological point of view may be responsible for his blindness to pragmatic matters.

> "Clearly then the 'information' which Dreyfus is concerned to have represented involves that of which we have 'experiential evidence' including such subjective phenomena as the feeling of 'zeroing-in' and our 'sense of oddness'.... It would not be enough to describe the function but one would have to simulate the appearances. But this amounts to a request that we reproduce the phenomena rather than simulate them.
> This can only reveal a basic misunderstanding as to the function of scientific understanding. As Einstein is said to have remarked, it is not the function of science to produce the taste in the soup! The scientist's task is not to duplicate phenomena but to make them accessible to the intellect. In contemporary Western science this can mean only one thing: The scientist must substitute for the 'real thing' a system built on principles which he can understand." (Pylyshyn, 1974:65)

Dreyfus, like a good phenomenologist, wishes to emphasise the over-riding importance of authentic human experience and to marginalise those abstract and disembodied cerebrations which collectively go by the name of science. This viewpoint has enabled him to furnish a penetrating critique of that part of artificial intelligence which is closest to direct human experience, which is CS, but it also blinds him to the importance and usefulness of the other near-scientific topics which go to make up the subject of artificial intelligence. Dreyfus's thought exhibits the strengths but also the weaknesses of those who adhere to the cause of insight in the ancient contest between abstract rationality and human intuition.

Cognitive simulation may well be, as Dreyfus claims, a tainted study, but he takes no account of those artificial intelligence topics for which a mere correspondence, rather than an actual or conceptual identity, between thinking and computing is sufficient. Most of the traditional artificial intelligence topics other than CS are of this type, and I think that these survive his strictures intact. Dreyfus may be said to have carried out a valuable piece of surgery on the body of artificial intelligence, but far from expiring upon the operating table the patient has recovered successfully and is now more healthy than before.

The great changes that have occurred in artificial intelligence research during the last decade and a half are partly the result of the strictures of writers such as Searle and Dreyfus. After 15 years of publishing, during which it grew from 300 pages to 1000 pages a year, the journal Artificial Intelligence published a long article entitled 'Artificial Intelligence - Where Are We?' (Bobrow & Hayes, 1985). The purpose was "to ask some of the people who have been in, or observers of, the field during these years to comment on where we have been, where we are and what the future might hold." The article is a stocktaking of the artificial intelligence workshop.

Many replies were of the 'its early days yet' kind. Donald Michie, for example, commented that;

> "A historical analogy is with the first synthesis of an organic compound in 1828, when a trace of urea was made, previously believed impossible except by participation of living cells."

Most contributors reflected that 'its tougher than we thought it would be'. Roger Schank observes that;

> "The most significant advance in the last decade has been the appreciation of just how complex the nature of thinking is. We have come to understand how complex the issues are."

But the usually tacit thought that informed most of the respondents' comments was that a fundamental change has come over all parts of the field of artificial intelligence studies. The nature of the change was most clearly described by Terry Winograd, who wrote;

> "My own work underwent a major change, as I moved away from the assumption that the way to make better and more useful computers (and interfaces) was to get them to be intelligent and use natural language. I recognised the depth of the difficulties in getting a machine to understand language in any but a superficial and misleading way, and am convinced that people will be much better served by machines that do well-defined and understandable things than those that appear to be like a person until something goes wrong (which won't take long), at which point there is only confusion."

However, the magazine editors did not ask their contributors to give their reasons for their assessment of the direction in which artificial intelligence is evolving. No answers were sought or provided on the exact nature of the difficulties that have been experienced, nor why it is that many aspects of artificial intelligence have turned out to be so much harder than was once supposed.

Conclusion

I think that much light can be shed on these more fundamental questions by making a comparison between the development of research in artificial intelligence and the evolution of the thought of Ludwig Wittgenstein.

In the second quarter of this century Wittgenstein evolved two very different philosophies. His first attempt upon the problems of meaning was close in spirit to that of the early workers on machine cognition. The cast of Wittgenstein's later thinking has many parallels with the more mature attitudes towards AI that are exhibited in the 1985 Artificial Intelligence article from which the preceding quotations have been taken. In the next two chapters I

shall try to contrast early and late notions of artificial intelligence by correlating them to the development of Wittgenstein's thought. The early Wittgenstein has little to offer the architect. However, the interpretation which Wittgenstein gives in his later work to the notion of meaning throws much light upon the process of architectural design.

Chapter 6. WITTGENSTEIN AND ARTIFICIAL INTELLIGENCE

During his lifetime Ludwig Wittgenstein published only one book, his Tractatus Logico-Philosophicus, the German edition of which appeared in 1921. The translation into English by Charles Ogden and Frank Ramsey, which was published in the following year with an introduction by Bertrand Russell, has been superseded by the 1961 translation of David Pears and Brian McGuiness, and it is the later version of Wittgenstein's text that I use in this thesis. The Tractatus is a young man's book - iconoclastic, rigorous and concise to the point of terseness. But Wittgenstein, despite the difficulty of the Tractatus, and even though he took no part in public life and shunned all publicity, is "the most influential philosopher of the 20th century" (Block, 1987). He is the only philosopher in modern times to have fathered not one but two distinct schools of thought.

The Vienna Circle

The Vienna Circle, a group whose best-known members were Moritz Schlick, Rudolph Carnap, Kurt Godel, Otto Neurath and Friedrich Waismann, were the originators of logical positivism. Their purpose in philosophy was to develop the empirical tradition of John Locke, David Hume and Ernst Mach by applying to it the techniques of symbolic logic. The modern study of logic was begun by Gottlob Frege with the publication in 1884 of his Die Grundlagen der Arithmetik and continued by Russell and Whitehead with their Principia Mathematica of 1913. But Wittgenstein in the Tractatus transformed the discoveries of these pioneers into a lucid and internally coherent logical system. The importance of his achievement was recognised immediately. Russell in his 1922 introduction to the first English edition said of the Tractatus,

> "whether or not it proves to give the ultimate truth on the matters with which it deals, [the Tractatus] certainly deserves, by its breadth

> and scope and profundity, to be considered an important event in the philosophical world."

The members of the Vienna Circle, whose collaboration spanned most of the 1920s and 1930s, were greatly indebted to Wittgenstein's ideas. One of the founder members of the group, and a central figure in the development of logical positivism, Moritz Schlick, has recorded his assessment of the Tractatus.

> "This book, which in my firm conviction is the most significant philosophical work of our day, cannot be assigned to any particular 'tendency', but it contends for the fundamental truth on which all empiricism is founded.....the inestimable significance of Wittgenstein's work lies precisely in this, that in it this nature of the logical is completely elucidated and established for all time to come. This happens in that, for the first time, an entirely clear and rigorous concept of 'form' is provided, which banishes at a stroke those difficult problems of logic which have lately given so much trouble to serious investigators." (Schlick, 1928)

It is ironic that Schlick's high hopes of the Tractatus were to be undermined by the subsequent work of Wittgenstein himself.

After his return to Cambridge and to philosophy in 1929, Wittgenstein's ideas evolved away from the pure and crystalline world of the Tractatus. He found reasons to doubt the status of logic as the irreducible structure of language, and from these doubts there emerged a fresh conception of language as a type of game whose meaning was inseparably bound up with usage. Wittgenstein in his work of the 1930s and 1940s telescoped Schlick's "for all time to come" into barely more than a quarter of a century. His later work was published posthumously, and collectively it constitutes the core texts of the Oxford school of natural language philosophy.

> "At Oxford Wittgenstein's ideas entered a very different philosophical atmosphere from that

which prevailed at Cambridge. Oxford philosophers, for the most part, have learnt their philosophy as a part of a course of study which is based upon classical scholarship: in particular, the influence of Aristotle has been strong at Oxford as it has never been at Cambridge, where so far as any classical philosopher has been influential it is Plato, not Aristotle.... At Oxford, then, Wittgenstein's ideas were grafted onto an Aristotelian-philological stock; the stock has influenced the resultant fruits which, amongst other things, are considerably drier and cooler than their Cambridge counterparts." (Passmore 1957)

But Wittgenstein's association with the Vienna Circle was not entirely forgotten in the ebb of philosophical fashion and the flow of events. One of the Wittgensteins's 14 posthumously published works, 'Ludwig Wittgenstein and the Vienna Circle', is a transcription and translation, published in English in 1979, of conversations with Wittgenstein recorded in shorthand by Waismann between 1929 and 1932.

Although Continental European thinkers of the late twentieth century continue to be interested mainly in questions of logic and structure, the line of thought that derives from the Oxford school retains its philosophical dominance in the English speaking countries to this day. So great was Wittgenstein's intellectual fertility that the doctrines of the Oxford philosophers, particularly their notion of meaning, effectively refute the earlier logical positivist school of thought. His unique achievement was to father two influential schools of thought, the second of which is a refutation of the first.

Wittgenstein's death in 1951 occurred only five years before the first recorded use of the term 'artificial intelligence' (in Charniak & McDermott, 1985). The character of the new discipline of artificial intelligence that emerged in the 1950s had much in common with the conceptions that lay behind the Tractatus. There was the same preoccupation with logic, a similar drive towards calcula-

bility and a shared assumption that meaning is synonymous with the truth function of a proposition. I think that the early workers in artificial intelligence were misguided in their approach to their subject for the same reasons that the assessment by Schlick and the Vienna Circle of the Tractatus was mistaken. The shortcomings in the Tractatus were elucidated by Wittgenstein himself in his later work, and they are summed up in the book published in 1953 as his Philosophical Investigations.

Wittgenstein did not arrange his texts into the normal pattern of sentence, paragraph, page and chapter. The questioning nature of his thoughts could not be reconciled with a flowing and connected prose style. He therefore adopted the practice of extracting material from his notebooks and editing it together into groups of short entries arranged according to topic. His literary executors have followed the same principles when preparing his posthumously published works for the press.

The Tractatus, for instance, consists of 526 continuously printed paragraphs which vary in length from a short sentence to nearly a whole page of text. In this book the paragraph numbering system follows an hierarchical classification system and serves to guide the reader by grouping entries together under topic. Philosophical Investigations is divided into only two sections. Part II contains only 14 long entries. Perhaps the editors despaired of subdividing and classifying such complex material more finely. The entries in Part I, however, are nearly as short and pithy as those of the Tractatus, and as in the earlier work they are roughly gathered into paragraphs according to topic. The editors of his notebooks and his conversations with Waismann, however, have provided no numbers to the paragraphs.

Because Wittgenstein's style is so compressed, and his prose so pregnant with meaning, the customary method of

annotating a commentary is too clumsy to be applied to his work. When referring to his texts one needs to be able to identify a smaller unit than the complete page. In this section of my thesis I have therefore abandoned the Harvard convention of referencing. Following the practice of other commentators, my references to the Tractatus and Philosophical Investigations are to the titles of these works, abbreviated to capital letters, followed by the paragraph number. The numbers following the abbreviations of the Notebooks and his conversations with Waismann are, however, to page numbers only. I refer to the Notebooks as NB, the Tractatus as TLP, to Ludwig Wittgenstein and the Vienna Circle as WVC, and to the two parts of Philosophical Investigations as PI I and PI II. Full descriptions of all four books appear in the list of references.

That an observable phenomenon must necessarily have an abstract theoretical cause is a conviction that comes easily to those of us who inherit the tradition of Western thought. A heated gas expands according to Boyle's law, an ice skater spins like a top because of the principle of the conservation of angular momentum, and the motion of an atomic particle cannot be described completely because of Heisenberg's uncertainty principle. These laws of nature are the culmination of 2500 years of intellectual effort and are in some ways the summit of our cultural achievement.

The profundity and durability, and the respect in which scientific generalisations are held, has lead many inferior writers to try to cloak speculation in the trappings of abstract principle. The 'laws' of Marx and the 'systems' beloved of sociologists come to mind. But despite these abuses I think that most people would agree that necessary principles are more illuminating than contingent facts, and it is therefore not surprising that early workers in the field of artificial intelligence should begin by assuming

that explanation of observations made in the new discipline would follow from the discovery of fundamental principles.

> "The field of artificial intelligence is full of intellectual optimists who love powerful abstractions and who strive to develop all-embracing formalisms. (Schank & Abelson, 1977)

I shall now try to show that the early investigators, when trying to deduce the "all-embracing formalisms" that would be appropriate to artificial intelligence, adopted a viewpoint very similar to that of the author of the Tractatus.

The Search for a Conceptual Base

A clear statement of the epistemological expectations of early workers in artificial intelligence is given in Roger Schank's contribution to a collection of papers that he and Kenneth Colby edited in 1973. This is the paper which introduced the idea of scripts to artificial intelligence. In it he says,

> "One basic assumption presented in this work is that since it is true that people can understand natural language, it should be possible to imitate the human understanding process on a computer, if it is possible to state those processes explicitly. Basically, the view of language understanding expressed here is that there exists a conceptual base into which utterances in natural language are mapped during understanding. Furthermore, it is assumed that this conceptual base is well-defined enough such that an initial input into the conceptual base can make possible the prediction of the kind of conceptual information that is likely to follow the initial input.
> Thus, we will be primarily concerned with the nature of the conceptual base and the nature of the mapping rules that can be employed to extract what we shall call the conceptualisations underlying a linguistic expression." (Schank, 1973:187)

Wittgenstein had a very similar notion when he wrote the Tractatus. In this work he gave to logic the role of "conceptual base". Logic must, as he says, "look after itself" because logic is prior to all experience.

> "To give the essence of a proposition means to give the essence of all description, and thus the essence of the world." (TLP 5.4711)
> "The description of the most general propositional form is the description of the one and only general primitive sign in logic." (TLP 5.472)
> "Logic must look after itself." (TLP 5.473)

An unqualified, and as I think ill-founded, faith in the
2 power of abstraction is characteristic of both early artificial intelligence theory and of the Tractatus.

Calculi and Computability

The machine orientation of artificial intelligence carries
3 with it a requirement for computability. Nearly 20 years ago Allen Newell summarised this aspect of artificial intelligence research from a Carnegie-Mellon point of view.

> "I should be explicit about the meaning of the term mechanism for me (and for the field of computer science, I might add). A mechanism is any determinate physical process. An abstract process constitutes a mechanism if, in principle, there are ways to realise it by a physical process. Thus, any program for a digital computer constitutes a mechanism. Similarly, a rule for which we can build a physical device that can realise its application is a mechanism (or represents one, if we want to be fussy).
> 4 This idea can be formalised in the notion of effective procedure, Turing Machine, Markov Algorithm, Post Production System. Or we can start with the formal system as the primitive (ideal) notion of mechanism, and work back toward physical processes. But it all comes to the same thing. Extension of usage to stochastic, statistical, or probabilistic mechanism is straightforward, going from the abstract notions of probability to physical processes that obey these formal models." (Newell, 1973:4)

A researcher who is of necessity concerned with issues of
5 computability will be attracted to a conception of knowledge that has the characteristics of a calculus. The Trac-

tatus is built round a method of calculation based upon symbolic logic, and it was later referred to by Wittgenstein as a calculus.

> "For there is not a mere analogy between our way of using words in a language and a calculus; I can actually construe the concept of a calculus in such a way that the use of words will fall under it." (WVC, 168)

The term 'mechanism' as used by Newell is virtually identical to Wittgenstein's conception of a 'calculus'.

Independence of Atomic Facts

By analogy with scientific reductionism, it is possible to hope that the bedrock of philosophy can be reached by means of analysis. If philosophical concepts are divided and subdivided sufficiently, and with enough rigour, then one will eventually get down to the irreducible pellets, or atoms, of thought whose existence serves to support all cognitive processes. This line of investigation took Russell to his empirical version of logical atomism, a title of his own creation, in which the atoms are indivisibly simple sense impressions (Russell, 1918). For Wittgenstein, however, the atoms of interest were of a logical rather than an experiential character. Wittgenstein, it should be noted, never applied the term 'logical atomism' to his own work.

> "Every statement about complexes can be resolved into a statement about their constituents and into the propositions that describe the complexes completely." (TLP 2.0201)

Wittgenstein's enquiries lead him to believe that the world can be described by the logical structuring of the names of irreducible things.

> "The world is the totality of facts, not things." (TLP 1.1)
> "The facts in logical space are the world." (TLP 1.13)

One consequence of this conception of reality is that logical facts are independent of one another. If two or more facts were found to be in any way dependent upon each other, then they would be composite rather than simple in nature and they would for this reason have to forfeit their atomic character. "Each item can be the case or not the case while everything else remains the same." (TLP 1.21) Wittgenstein constructs the whole edifice of the Tractatus upon the simplicity and combinability of facts, and the logical innovations made in his work could not be based upon on any other supposition.

But atomic simplicity and unfettered combinability are also attractive features in the symbols to be used in a computing environment. Symbols that are independent of one another can then be manipulated freely, and the patterns that emerge from a computation would reflect the rules of combination rather than the status of the symbol. Furthermore, a group of symbols can be added to or subtracted from without disturbing the structure of the set. These desirable features were not overlooked by early workers in the field of artificial intelligence. Terry Winograd, for instance, in a discussion of the problems involved in natural language processing, remarked that;

> "We can view production systems as a programming language in which all interaction is forced through a very narrow channel..... The temporal interaction [of individual productions] is completely determined by the data in this STM [short term memory], and a uniform ordering regime for deciding which productions will be activated in cases where more than one might apply.... Of course it is possible to use the STM to pass arbitrarily complex messages which embody any degree of interaction we want. But the spirit of the venture is very much opposed to this, and the formalism is interesting to the degree that complex processes can be completely described without resort to such kludgery, maintaining the clear modularity between the pieces of knowledge and the global process which uses them." (Winograd, 1975)

In fact, Winograd's paper proposes to construct an entire automatic natural language processing system upon the pattern of a production system.

The claim that facts were independent of one another was the first doctrine of the Tractatus to be abandoned by Wittgenstein when he took up the study of philosophy again in 1929. Speaking in December of that year to Waismann about the Tractatus he said,

> "I thought that all inference was based on tautological form. At that time I had not yet seen that an inference can also have the form: This man is 2m tall, therefore he is not 3m tall. This is connected with the fact that I believed that elementary propositions must be independent of one another, that you could not infer the non-existence of one state of affairs from the existence of another. But if my present conception of a system of propositions is correct, it will actually be the rule that from the existence of one state of affairs the non-existence of all other states of affairs described by this system of propositions can be inferred." (WVC 64)

Wittgenstein is saying here that some propositions are mutually exclusive in such a way that no amount of analysis will make them otherwise. In this he is, I think, correct. One must conclude, therefore, that Winograd's ambition to be able to add or subtract rules from a production system as and when convenient, while maintaining its integrity as a semantic system, is impossible. The meaning of the individual rules will establish 'kludgery' connections between them despite their formal independence. Furthermore, these connections will be invisible to a machine which is engaged in simply manipulating symbols according to a program.

The Logic of a Double Negative

The Tractatus interested Russell, and later the members of the Vienna Circle, for technical as well as philosophical reasons. In the central third of the Tractatus, occupied by the paragraphs beginning with 4. and 5., Wittgenstein is

concerned with the nature and status of logical operations. He devotes such a large section of his text to this topic because of a dissatisfaction with the earlier methods adopted by Russell and Whitehead in their Principia.

The logical notation adopted by Russell and Whitehead (1913, Vol I:6) is based upon that invented by Frege. Propositions are to be related to one another by means of five connectives, which can be set out in a list.

Contradictory	(negation)	~
Logical Sum	(either....or)	v
Logical Product	(and)	.
Implicative	(if....then)	⊃
Equivalence	(equivalent)	≡

Wittgenstein in paragraph 5.101 gives the 16 truth-functions which can be derived from two propositions by using these connectives in all possible meaningful combinations. But he is dissatisfied with the notation because to make use of it at all is to imply that the two propositions are unrelated until they are brought together into one of these 16 expressions.

Wittgenstein's whole case in the Tractatus is that propositions are related to one another on account of their internal logical nature, not the contingent fact that they have been juxtaposed in an expression.

> "If the truth of one proposition follows from the truth of others, this finds expression in relations in which the forms of the propositions stand to one another: nor is it necessary for us to set up these relations between them, by combining them with one another in a single proposition; on the contrary, the relations are internal, and their existence is an immediate result of the existence of the propositions." (TLP 5.131)

He is therefore driven to seek another notation which recognises the pre-existing nature of the relationship between one proposition and another. This he immediately

does in the next paragraph when he brings into his argument the notational device known as the Sheffer stroke, | (Sheffer, 1913). The Sheffer stroke means 'neither...nor', so that 'p|q' means 'neither p nor q'. By employing this notation, an expression which would have been written 'p v q' by Russell and Whitehead can be stated as 'p|q.|.p|q', and furthermore the double negative expression '~(~p.~q)' can be reduced to the single Sheffer connective. Wittgenstein describes the idea as follows;

> "When we infer q from p v q and ~p, the relation between the propositional forms of 'p v q' and '~p' is masked, in this case by our mode of signifying. But if instead of 'p v q' we write, for example, 'p|q.|.p|q', and instead of '~p', 'p|p' (p|q = neither p nor q), then the inner connexion becomes obvious." (TLP 5.1311)

Wittgenstein is saying that the fact that the five connectives used by Russell and Whitehead can be replaced by a single symbol proves his central point. This is, that a proposition can be inferred from others not on account of the connectives that we choose to place between them, but by the fact that when they are brought into relation by means of a connective their sense becomes immediately obvious from their nature.

Wittgenstein has now established the two fundamental notions upon which his explication of language in the Tractatus are based. These are the independence of atomic facts and the derivation of propositions from previous propositions by means of a single all-sufficient operation in logic. In paragraphs 5.2 to 5.52 Wittgenstein expands this argument, by means of a negative recursive procedure, into what he refers to as "the general propositional form" (TLP 5.54). His description of general propositional form, omitting the long and difficult argument by which it was arrived at, is given in paragraph 5.3 of the Tractatus.

"All propositions are the results of truth-operations on elementary propositions.
A truth-operation is the way in which a truth-function is produced out of elementary propositions.
It is the essence of truth-operations that, just as elementary propositions yield a truth-function of themselves, so too in the same way truth-functions yield a further truth-function.
When a truth-operation is applied to truth-functions of elementary propositions, it always generates another truth-function of elementary propositions, another proposition. When a truth operation is applied to the results of truth-operations on elementary propositions, there is always a single operation on elementary propositions that has the same result.
Every proposition is the result of truth-operations on elementary propositions."

Wittgenstein's expansion of Sheffer's discovery into a form in which it can be applied to the derivation of general propositions is the aspect of his work that most impressed his contemporaries. It is what is referred to by Russell as "an amazing simplification of the theory of inference" in his introduction to the Tractatus.

I have thought it worthwhile to describe Wittgenstein's conception of the general form of a proposition at some length because it entails the use of the logical device of the double negative. In logic, but not in language, a double negative is equivalent to a positive. The whole crystalline structure of the Tractatus would shatter without the support of this notion.

"The sense of a truth-function of p is a function of the sense of p.
Negation, logical addition, logical multiplication, etc. etc. are operations.
(Negation reverses the sense of a proposition.)" (TLP 5.2341)
"An operation can vanish (e.g. negation in '~~p': ~~p = p). (TLP 5.254)

The same conventional equivalence of the double negative and the positive is used in the logic of computer programming. It functions as part of both propositional and predi-

cate logic, where it is sometimes known as the generalised form of De Morgan's law, as well as doing duty in normal mathematical notation.

> "Theorem 1.4 (Generalised De Morgan's law)
> For an arbitrary proposition A constructed using only the connectives ~.

A	~A	~~A
F	T	F
T	F	T

> Fig. 1.15. Truth table showing the equivalence of A and ~~A." (Dowsing, Rayward-Smith & Walter, 1986:20)

It is clear that all computer operations, including programs written for the purpose of artificial intelligence, are as dependent upon the equivalence of the double negative and the positive as is the Tractatus itself.

Rules and Truth-Functions

One of the participants in the 1956 Dartmouth Summer Project on Artificial Intelligence, for which John McCarthy coined the phrase 'artificial intelligence', was Marvin Minsky. His contribution to the proceedings was published later in an amplified form as 'Steps Toward Artificial Intelligence'. Most of his paper is taken up with describing the domain of AI and with speculating about likely avenues of advance. But in his concluding remarks Minsky gives an account of the epistemological assumptions that he believed must necessarily underlie the new discipline.

> "Suppose that we want a machine which, when embedded for a time in a complex environment or 'universe', will essay to produce a description of that world - to discover its regularities or laws of nature. We might ask it to predict what will happen next. We might ask it to predict what would be the likely consequences of a certain action or experiment. Or we might ask it to formulate the laws governing some class of events. In any case, our task is to equip the machine with inductive ability - with methods which it can use to construct general

> statements about events beyond its recorded experience. Now, there can be no system for inductive inference that will work well in all possible universes. But given a universe, or an ensemble of universes, and a criterion for success, this (epistemological) problem for machines becomes technical rather than philosophical." (Minsky, 1961:27)

Minsky's "method to construct general statements" is very close to Wittgenstein's law of the projection of inference, which he illustrates in the Tractatus by means of a musical analogy. Wittgenstein, however, shares Popper's distrust of induction, (TLP 5.135, 6.363), and would see no point in giving a machine inductive ability.

> "There is a general rule by means of which the musician can obtain the symphony from the score, and which makes it possible to derive the symphony from the groove on the gramophone record, and, using the rule, to derive the score again. That is what constitutes the inner similarity between these things which seem to be constructed in such entirely different ways. And that rule is the law of projection which projects the symphony into the language of musical notation. It is the rule for translating this language into the language of gramophone records." (TLP 4.0141)

Similarly, Wittgenstein's notion of the truth function of a proposition, which he derived from Frege, Russell and Whitehead, is a close parallel with Minsky's putative "criterion for success".

> "Like Frege and Russell I construe a proposition as a function of the expressions contained in it." (TLP 3.318)
> "To understand a proposition means to know what is the case if it is true.
> (One can understand it, therefore, without knowing whether it is true.)
> It is understood by anyone who understands its constituents." (TLP 4.024)

But, as we have seen, there is no "criterion for success" in artificial intelligence, and no method for mechanically constructing general statements about events has been forthcoming. I think that the explanation for this is that

the epistemology of early artificial intelligence was of the same, as it now seems over-optimistic, type as that which is set forth in the Tractatus.

Conclusion

If I am correct in this assessment, then Wittgenstein's later criticisms of the Tractatus can be applied very closely to the methods and assumptions of early artificial intelligence, and to do so will serve to show why some of its initial ambitions are impossible to realise. I therefore propose in the next section of my text to discuss, from a late Wittgensteinian point of view, the attitudes to knowledge implicit in the work of some of the pioneers of artificial intelligence. I hope that this will lead to a more correct and mature understanding of the subject.

Chapter 7. AI AND THE LATER WITTGENSTEIN

In the 30 years that elapsed between the publication of the Tractatus and Wittgenstein's death in Cambridge his ideas about the relationship between logic and language changed greatly. One of the reasons for his change of attitude was that he realised that language as it is used, as opposed to how it may be structured, does not necessarily equate a positive and a double negative.

In Philosophical Investigations Wittgenstein asked questions about language that in his Tractatus days would have seemed meaningless.

> "Imagine a language with two different words for negation, 'X' and 'Y'. Doubling 'X' yields an affirmative, doubling 'Y' a strengthened negative. For the rest the words are used alike.- Now have 'X' and 'Y' the same meaning in sentences where they occur without being repeated?- We could give various answers to this.
> (a) The two words have different uses. So they have different meanings. But sentences in which they occur without being repeated and which for the rest are the same make the same sense.
> (b) The two words have the same function in language-games, except for this one difference, which is just a trivial convention. The use of the two words is taught in the same way, by means of the same actions, gestures, pictures and so on; and in the explanations of the words the differences in the ways they are used is appended as something incidental, as one of the capricious features of the language. For this reason we shall say that 'X' and 'Y' have the same meaning.
> (c) We connect different images with the two nega tives. 'X' as it were turns the sense through 180^{o}. And that is why two such negatives restore the sense to its former position. 'Y' is like the shake of the head. And just as one does not annul a shake of the head by shaking it again, so one doesn't cancel one 'Y' by a second one. And so even if, practically speaking, sentences with two signs of negation come to the same thing, still 'X' and 'Y' express different ideas." (PI I, 556)

Wittgenstein, in paragraph (c) above, has put his finger upon one of the main difficulties inherent in semantic information processing. The expression '-2 x -4 = 8' and the sentence 'I never, never drink spirits' are formally identical, but they mean different sorts of things. This kind of distinction is opaque to a symbol manipulating machine such as a computer, and can only be overcome if the computer is as conversant with language as is an educated human being. It is impossible to imagine a program that supplies a computer with all that can be known about language and its use, for this is an infinite amount of information, and there is no way of doing such a thing. For the present, then, we must take Wittgenstein's point to heart, and not expect from a computer what it cannot deliver.

Logic and Semantic Nets

Wittgenstein's objective in writing the Tractatus was to discover the foundations upon which language and our understanding of the world must rest. He believed that he had found his version of the holy grail in logic, and particularly in his own refined version of symbolic logic.

> "A logical picture of facts is a thought." (TLP 3)
> "The totality of true thoughts is a picture of the world." (TLP 3.01)
> "Thought can never be of anything illogical, since, if it were, we should have to think illogically." (TLP 3.03)

Logic, for him, was necessary to and independent of experience. "for Wittgenstein, there was an absolute distinction between the empirical and the logical, such that the latter would never depend upon the former." (Mounce, 1981:9)

In his paper of 1968, in which he introduces the idea of semantic nets, Ross Quillian describes the logical structure that he believes is needed in order to achieve understanding by a machine.

> "It further seems likely that if one could manage to get even a few word meanings adequately encoded and stored in a computer memory and a workable set of combination rules formalised as a computer program, we could then bootstrap this store of encoded word meanings by having the computer itself 'understand' sentences that he has written to constitute the definitions of other single words. That is, whenever a new, as yet uncoded, word could be defined by a sentence using only words whose meanings had already been encoded, then the representation of this sentence's meaning, which the machine could build up by using its previous knowledge together with its combination rules, would be the appropriate representation to add to its memory as the meaning of the new word. Unfortunately, two years of work on this problem led to the conclusion that the task is much too difficult to execute at our present state of knowledge. The process that goes on in a person's head when he 'understands' a sentence and incorporates its meaning into his memory is very large indeed, practically all of it being done without his conscious knowledge." (Quillian, 1968:246)

Quillian confesses the difficulties he is experiencing, but while doing so he implies that success can be expected when more work has been done and the state of our knowledge has improved. However, I think that his difficulties are inherent rather than contingent, and that his ambition of mechanising the process of meaning can never be realised. The reasons that lead to this conclusion were identified by Wittgenstein in his later work.

Much of Philosophical Investigations is taken up with the development of a theory of meaning as use, and with the exploration of the idea of language games. In the Tractatus Wittgenstein tried to show that the truth of a statement was a function of a well-formed proposition, and he assumed that matters of meaning were psychological rather than philosophical in character. "the relation of the constituents of the thought and of the pictured fact is irrelevant. It would be a matter for psychology to find out." (NB 129) But in Philosophical Investigations he gave weight to the

notion that one cannot comprehend the truth or falsity of a statement without knowledge of the use of the terms that are employed.

> "we might say..... that a sign 'R' or 'B', etc. may be sometimes a word and sometimes a proposition. But whether it 'is a word or a proposition' depends on the situation in which it is uttered or written.....For naming and describing do not stand on the same level: naming is a preparation for description. Naming is so far not a move in the language game - any more than putting a piece in its place on the board is a move in chess. We may say: nothing has so far been done, when a thing has been named. It has not even got a name except in the language game." (PI I, 49)

For example, a sign consisting of the letters 'speech' will acquire a partly different meaning according to whether it is used in a political, elocutional or ethnographic situation, the logical structure of the language notwithstanding.

> "It is interesting to compare the multiplicity of the tools in language and of the ways they are used, the multiplicity of the kinds of word and sentence, with what logicians have said about the structure of language. (Including the author of the Tractatus Logico-Philosophicus.)" (PI I, 23)

I think that Wittgenstein's later thoughts are correct on this point, and that Quillian will never acquire a 'store of encoded word meanings' nor a 'set of combination rules formalised as a computer program'. Words will always shift their meaning according to the use to which they are put, and the rules for their combination will vary from one linguistic situation to another. No logic can define even one word in a permanent and unambiguous fashion, and I therefore think that Quillian's notion of semantic net is unsound. When he proposes semantic nets he is looking to logic for something that it cannot provide.

Picture and Frame Theories of Representation

Following Frege, Wittgenstein makes a distinction in the Tractatus between the sense and the truth of a proposition.

> "It is clear that we understand propositions without knowing whether they are true or false. But we can only know the meaning of a proposition when we know if it is true or false. What we understand is the sense of the proposition." (NB 94)

That is to say, a proposition must have sense if it is to be a proposition at all, but it will only possess meaning if it is true. But how are we to know when a proposition is true or false? Wittgenstein introduces his famous picture theory of meaning in an attempt to answer this question. The sense of a picture, he says, is to be found in the arrangement of its constituent parts rather than in its relation to external facts. This must be so, for otherwise it would be impossible to have a picture of a non-existent object. Its sense, then, is a property internal to the picture. In the same way, he says, a proposition possesses sense when it has an internal logical structure. But only when the structure of a proposition corresponds to the structure of some aspect of external reality can we say that it possesses meaning as well as sense.

> "The essence of a propositional sign is very clearly seen if we imagine one composed of spatial objects (such as tables, chairs and books) instead of written signs. Then the spatial arrangements of these things will express the sense of the proposition." (TLP 3.1431)

In the next paragraph he expresses this thought in a more general and abstract way.

> "Instead of, 'The complex sign "aRb" says that a stands to b in the relation R', we ought to put 'That a stands to b in a certain relation says that aRb.' (TLP 3.1432)

That is to say, it is the fact of the relation between a and b that gives meaning to the proposition aRb, and not the reverse. So it transpires that a proposition will have sense if it is logical, and meaning if it pictorially corresponds to external reality. I think that it is worth discussing these rather difficult aspects of the Tractatus in a thesis about artificial intelligence because Wittgenstein's concept of the proposition as picture is so closely parallelled by Minsky's notion of frames.

Minsky claims a fellow feeling with Schank, Abelson and Norman when he finds himself "moving away from the traditional attempts by behaviouristic psychologists and by logic-oriented students of artificial intelligence in attempts to represent knowledge as collections of separate, simple fragments." (Minsky, 1975:211). In an attempt to supply "a unified, coherent theory" he proposes that:

> "We can think of a frame as a network of nodes and relations. The 'top levels' of a frame are fixed, and represent things that are always true about the supposed situation. The lower levels have many terminals - slots that must be filled by specific instances or data. Each terminal can specify conditions its assignments must meet." (Minsky, 1975:212)

His fixed 'top level' corresponds almost exactly to the logical sense of a Wittgensteinian proposition, while the 'terminals' reflect the situation from which the frame obtains its meaning. If the Tractatus is right about this interpretation of meaning then Minsky and his idea of frames is also right. But Wittgenstein himself, in his later work, has cut the ground from under both the picture and the frame theory of meaning.

It is possible to maintain that a Wittgensteinian picture must be complete when the mood of a proposition is indicative.

> "I can describe a state of affairs which consists in there being a circle of specific size at a specific point of the square. This is a complete picture. For to what follows it does not matter what description I choose, whether I use co-ordinates for example; what matters is only this, that the form of description has the right multiplicity." (WVC, 39)

But a difficulty arises when the mood of the phrase is conditional - when, for instance, the size of the circle is conditional upon a numerical qualifier. Then the picture may or may not be complete depending upon how its definition is qualified.

> "Thus when numbers occur in the sentence and indicate where the circle is and how large it is, it may happen that I replace the numbers by variables or perhaps only by intervals, e.g.{6-7, 8-9}, and then I shall get an incomplete picture. Imagine a portrait in which I have left our the mouth, then this can mean two things; first, the mouth is white like the blank paper: second, the picture is always correct, whatever the mouth is like." (WVC, 39)

For these reasons Wittgenstein was driven to the conclusion that, despite what he had written in the Tractatus, a picture may have sense but nevertheless be incomplete. From this it follows that a picture, because it may be incomplete, can have no necessary truth-function and with this admission the whole structure of meaning as set out in the Tractatus falls to the ground. Wittgenstein recognised this conclusion when he wrote;

> "We see that what we call 'sentence' and 'language' has not the formal unity that I imagined, but is the family of structures more or less related to one another..... The preconceived idea of crystalline purity can only be removed by turning our whole examination round. (One might say: the axis of reference of our examination must be rotated, but about the fixed point of our real need.)" (PI I, 108)

I shall describe what I believe are the consequences for AI of Wittgenstein's later views shortly. In the meantime one must conclude that Minsky's frames, like Wittgenstein's

pictures, will be incomplete and misleading when they incorporate quantifiers. But a frame must if it is to be useful incorporate quantifiers, described by Minsky as 'terminals', and a Minskyan frame can therefore never be complete. I conclude that frames cannot fulfil Minsky's purpose of providing "a unified, coherent theory" of knowledge. Nevertheless, frames may be useful tools.

Understanding as Mapping or Language-Game

Any discussion of the nature of artificial intelligence must attempt to come to terms with the notion of understanding. In her book Artificial Intelligence Elaine Rich makes this attempt in Chapter 9, where she says that;

> "understanding is the process of mapping a statement from its original form to a more useful one." (Rich, 1983:298)

The mapping analogy of meaning appears frequently in the literature of AI (Schank & Abelson 1977, Simon 1977, Nilsson 1980, Akman, ten Hagen & Tomiyama, 1990). An example from the writings of Schank has been quoted earlier in this chapter. The notion that meaning is something to be searched for, and that it can be found by employing the right procedure, is also implicit in Wittgenstein's picture theory of meaning.

> "The pictorial relationship consists of the correlations of the picture's elements with things." (TLP 2.1514)

Mapping procedures lend themselves to description by rules - Quillian's semantic nets are an effort to do just this - and they will therefore be attractive to workers in artificial intelligence who are trying to program a computer to, in a certain sense, understand. Some successes have been achieved by artificial intelligence programs which rely upon automatic mapping methods. Students have, for example, been helped to diagnose faults in electronic circuit designs by using SOPHIE (Brown et al, 1982). SOPHIE contains

a sub-routine, called the "referee" by the authors, that maps student answers onto a library of known fault conditions, and in this sense the program 'understands' the student-machine dialogue. I think that these methods can be very useful in artificial intelligence provided that one remembers the restricted sense in which they display the quality of understanding. Mapping procedures transform symbols, but they do not possess knowledge.

Wittgenstein begins his Philosophical Investigations with a quotation from Saint Augustine's Confessions. The saint recounts how as a child he learnt to map the words that he heard used by his elders onto the objects that these words denoted. This is, in fact, a very elementary theory of language, and Wittgenstein's purpose in beginning with so simple an example is to show by contrast how complex language really is.

> "That [Augustine's] philosophical concept of meaning has its place in a primitive idea of the way language functions. But one can also say that it is the idea of a language more primitive than ours." (PI I, 2)

Learning the name of something is just one example of a language-game.

> "This [asking something's name], with its correlate, ostensive definition, is, we might say, a language-game of its own. That is really to say: we are brought up, trained, to ask: 'What is that called?'- upon which the name is given. And there is also the language-game of inventing a name for something, and hence of saying, 'This is ....' and then using the new name." (PI I, 27)

By the use of the term language-game Wittgenstein means to draw an analogy between the use of language and playing a game. In both there is a set of rules and conventions which determine which moves are permissible, and a given move can only be judged according to the rules of the game to which it belongs. Thus, kicking the ball is a legitimate - which

is to say, meaningful - move in soccer or rugby, but not in cricket. By analogy, a word will have one meaning when functioning in an interrogative and another in a jocular language-game. Wittgenstein lists 24 examples of language-games in paragraph 23 of Philosophical Investigations, and countless others occur throughout the body of the book. But he nowhere gives rules for identifying or defining the entire set of language-rules. This omission is for the very good reason that it would be impossible to do this without describing the entire structure and content of the language itself.

> "We remain unconscious of the prodigious diversity of all the everyday language-games because the clothing of our language makes everything alike.
> Something new (spontaneous, 'specific') is always a language-game." (PI II, 124)

I think Wittgenstein is right when he characterises language as made up of an infinite number of language-games. Dreyfus, in his concern with context, is in effect restating Wittgenstein's thesis in Philosophical Investigations. It follows that, for true understanding to take place in a computing environment, an infinite number of representations are required in order that, as Rich says, a correlation can take place between the original form and a more useful one. But this is a task that is quite beyond us. No program can embrace the "prodigious diversity" of language games, let alone incorporate an infinite set of representations. I think, therefore, that we must leave the question of computer understanding in abeyance, at least for the present and probably forever, as insoluble.

Conclusion

I am driven to conclude that there is no solution, nor is there ever likely to be a solution, to some of the problems which have during the last 30 years been investigated under the auspices of artificial intelligence. The case advanced by Searle against the ambition to get a computer to under-

stand is, it seems to me, irrefutable. No program can make up for the fact that a machine cannot acquire a point of view. Dreyfus has shown that the notion of information processing by computer is based upon a confusion between the two senses, the weak and the strong, of the idea of information. Thirdly, a Wittgensteinian analysis shows that no system of rules can encompass the vast number of procedures that go to make up natural language. In fact, the insoluble problems in artificial intelligence are just those which involve machine understanding of natural language. This is the reason why linguistic philosophy is able to illuminate the study of artificial intelligence.

If this conclusion is correct then we must be content to classify any artificial intelligence topic that embraces natural language, representing common-sense knowledge or understanding language for example, as a long-term research project. It follows that, for the present, we should forgo any claim to be able to make use of computers as processors of natural language.

But there are many other aspects of artificial intelligence that take the computer for what it really is, which is simply a fast and tireless symbol manipulator. Traditional artificial intelligence subjects which lend themselves to computer processing, and in which progress can be made, are pattern recognition, voice recognition, robotics, games playing, problem solving, intelligent tutoring systems and expert systems. These are the topics that make up weak artificial intelligence, in the sense in which this term was used in Chapter 3 by Searle. The next section of my text will attempt to bring cognitive simulation, general artificial intelligence and intelligent artifacts together into a coherent taxonomy of artificial intelligence.

Chapter 8. THE TAXONOMY OF ARTIFICIAL INTELLIGENCE

Cognitive simulation continues to occupy a prominent place in the lay conception of artificial intelligence. Although it is an interesting topic, the study of which throws much light upon epistemology, cognitive simulation is only one small department of the enterprise of artificial intelligence. Artificial intelligence programs that are likely to be useful to the architect will emerge from other parts of the subject.

The Encyclopedia of Artificial Intelligence (Shapiro, Eckroth & Vallasi, 1987) contains several hundred separately entitled subject entries. They range in specificity from 'The Nature of Logic' to short entries on lesser-known language parsers. However, there are a few fundamental topics upon which the large number of particular subjects that are embraced by artificial intelligence depend. Natural language, visual perception, machine learning, search, control, and solving problems are the underlying techniques of artificial intelligence. These, together with cognitive simulation, intelligent tutoring systems and expert systems are the headings of the taxonomy of artificial intelligence which is attempted in this chapter.

In one of the definitions of artificial intelligence that was quoted in Chapter 2 Peter Sell divided the topic into two parts, "models of human cognition and intelligent artifacts." Efforts to create programs of both these have been central to the enterprise of artificial intelligence since its beginning, but Sell's definition leaves out most of what might be called mainstream artificial intelligence. Many topics in artificial intelligence are attempts to produce useful programs by advancing our understanding of thought processes. Mainstream artificial intelligence lies between Sell's two categories, and in effect tries to unite them.

Natural Language Processing

Perhaps the most characteristic manifestation of natural intelligence is our use of language. By far the larger part of human language use, and all of the simple language-like activities of animals, takes the form of spoken language. In speech the representation of thought is by means of words, and the words are themselves represented by sounds. But in speech the spoken words are modified and qualified by tone of voice, facial expression, body language, timing and volume. Spoken language is thus much more than a simple verbal phenomenon, and it is not susceptible to purely linguistic dissection.

So great is the complexity and subtlety of language that some theorists have been driven to the conclusion that linguistic ability is innate. How otherwise can a three-year-old child, years away from playing chess or understanding calculus, use language sufficiently well to hold a meaningful conversation? Some commentators have come close to asserting the existence of a human 'language organ', on the analogy of a hand or a foot, which one comes to use as a result of experience and education (Miller & Chomsky, 1963). Other researchers, notably the Swiss psychologist Jean Piaget (1936), take the view that the ability to use language emerges as part of the general intellectual development of the human personality. The theories of both Chomsky and Piaget imply, to a greater or lesser extent, the existence of some kind of inherited propensity to speak. Perhaps we shall understand the nature of this feature of our minds one day. In the meantime, however, the lack of understanding of how we acquire and use language hampers efforts to write language processing computer programs.

The slowness of progress in creating speech processing programs is the result of a poor understanding of the phenomenon of language, and of the fact that spoken language is only a component part of our apparatus of everyday

communication. The written word is, however, a somewhat more describable phenomenon. When words are represented not by fleeting sounds but by written symbols they acquire a certain stability, and also a set of rules for their use. A written sentence is distinguished as a declaration or a question by rules of grammar, and not by means of the intonation of the voice as in speech. These rules give a handle on written text, and furnish a means by which text processing algorithms may be devised. This is the reason why research in natural language programming is largely concerned with the written word.

Commercial natural language processing programs are usually intended to facilitate access to databases for the non-technical user. Some are available for use with desktop machines, but the most long-established system is INTELLECT (Harris, 1977) which runs on IBM mainframes. It functions by matching input strings with a lexicon of words and concepts. The lexicon can be extended by the user to cover the words applicable to a particular domain. All goes well provided that the user follows the rules that reside in the system, but errors occur when the input is too natural, that is to say, when the user fails to frame his enquiry in a full and complete style. INTELLECT, unlike a human interlocutor, cannot understand when an inquirer takes part of the answer for granted.

Programs such as INTELLECT are interesting, but their value lies at least as much in showing what the problem is as in their functionality. They can work only in a very circumscribed domain, and for the moment one must conclude that there is no language processing program which would be useful to the architect. Furthermore, it is hard for the reasons advanced in Chapter 4 to see how progress towards a useful system can be made.

Visual Perception

Architecture is by its nature centred upon visual experience and any artificial intelligence program that displayed visual capability would be close to the architects sphere of interest. Conventional computer-aided drawing programs display no more intelligence than a drafting pen. However, it is possible to imagine a program which could do very much more. For example, a program could be devised which could read a set of photographs or a video tape of a building and output drawings of its plan and elevations. Such a program would be both an interesting and a useful tool in the design studio.

A good deal of progress has been made in the optical aspects of automatic visual perception. A photograph or a video image consists of an arrangement of tones and, if a colour image, of hues. But it is difficult to analyse them because tones merge into one another, making it hard to distinguish the edges between tones that denote the edges of the objects that are being represented. The reality of a visual edge is shown in Figure 8.1(a), in contrast to the computationally desirable shape of the step in 8.1(b).

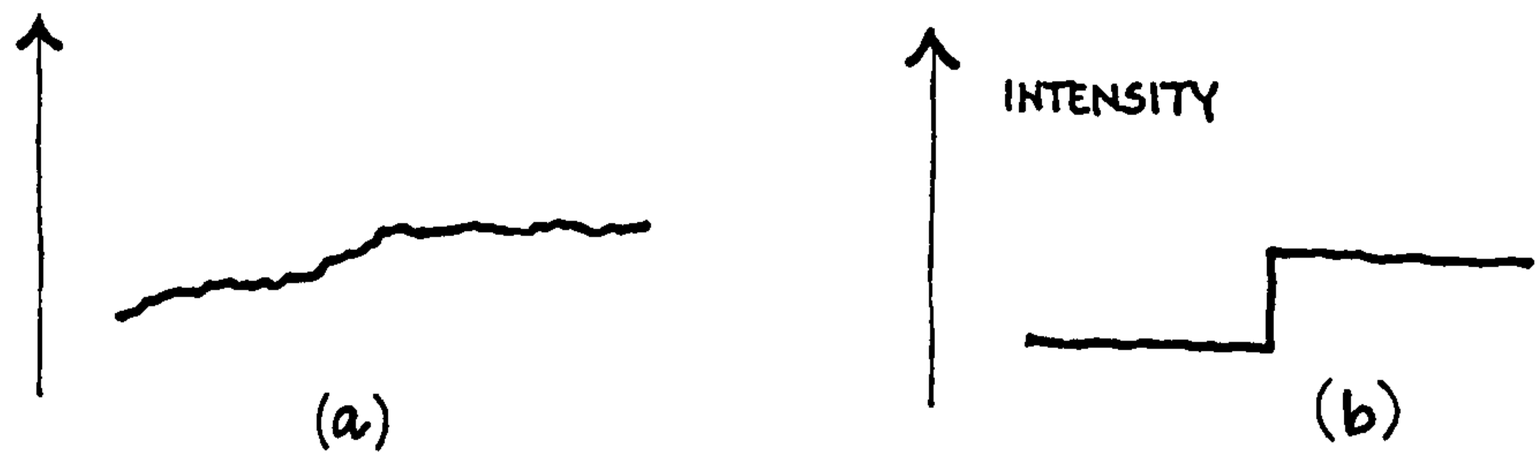

Figure 8.1
Visual Edges.
(Redrawn from Winston, 1984)

If an image is divided in a grid-like fashion then the average tone of the whole is the average of the individual cells. Edges occur where the tone of adjacent cells are on, above or below the plane of the tonal average. It is not difficult to calculate the position of the edges by summing

the degree to which each point in the image contributes to forming an intersection with the plane of the average intensity. The procedure is illustrated diagrammatically in Figure 8.2.

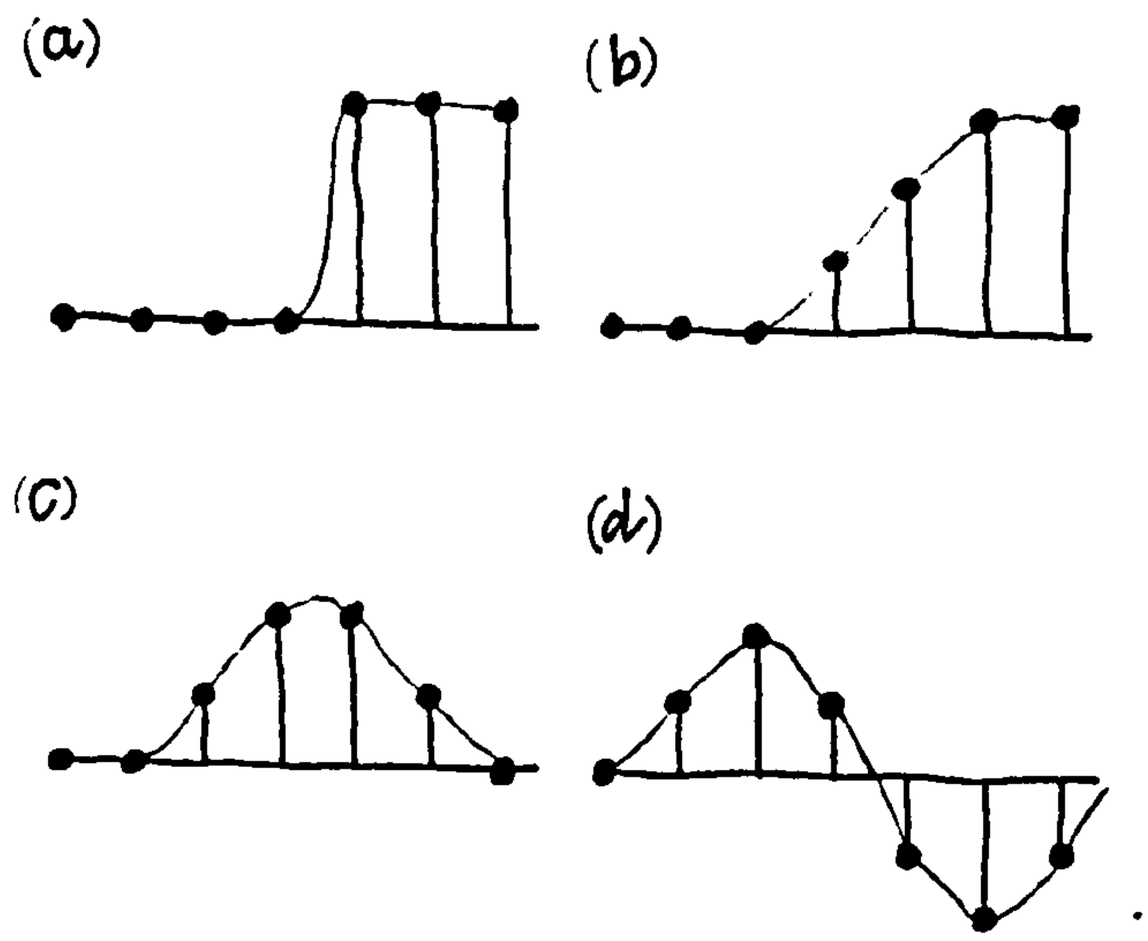

Figure 8.2.
Filtering a Point.
(a) brightness change to be analysed. (b) each point averaged with respect to its neighbours. (c) averaged difference between points in (b). (d) averaged differences in (c) step is localised to the point at which the line crosses the x-axis.
(Redrawn from Winston, 1984)

The result of analysing a tone image is shown in Figure 8.3. Most of the important edges in the photograph have been detected, but the output is far from accurate. This is because some dark tone is shadow while other is shade. The algorithm cannot distinguish between the two types of darkness, and the wayward edges in Figure 8.3(b) reflect this unresolved ambiguity. Nevertheless, the broad picture emerges correctly.

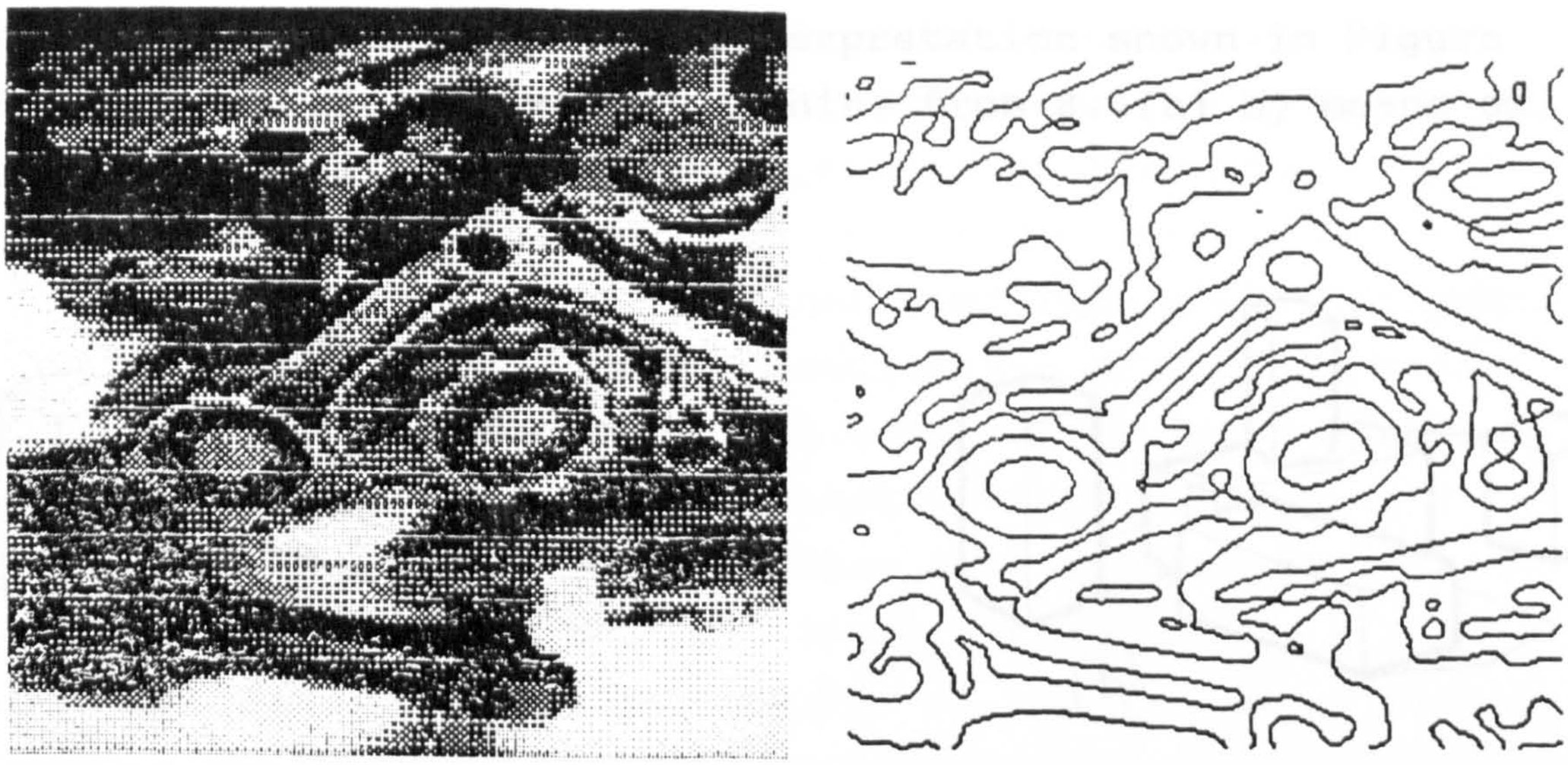

Figure 8.3
Example of Edge Detection
(a) machinery part in coarse-grained half-tone (b) computer derived edge locations of (a)
(from Zucher, 1987)

The planes of which every six-sided polyhedron, such as a cube or a rectangular block, is made up meet at 12 arises and at eight vertices. Each vertex in such an object is the meeting point of three planes. Most buildings, and many other objects such as paving materials, books and containers of all kinds, are made up of six-sided polyhedra, and a computer system that can interpret them correctly would have many applications in architecture and elsewhere.

In 1975 the American computer scientist David Waltz published a paper in which he was able to show that a scene made up of blocks, when represented by a line drawing, is composed of a surprisingly small number of line junctions. A glance at Figure 8.4(a) might give the impression that the lines join in a myriad ways, but Waltz demonstrated that the vertices at which three planes meet can only belong to a set of as few as 18 types. The set consists of four T's, three arrows, five forks and six L's. When this fact is systematically applied as a constraint to a scene of blocks it is possible to resolve the ambiguities of a two-dimensional drawing and output the three-dimensional

nature of the objects. The interpretation shown in Figure 8.4(b) has been derived by machine from 8.4(a) by means of Waltz's algorithm.

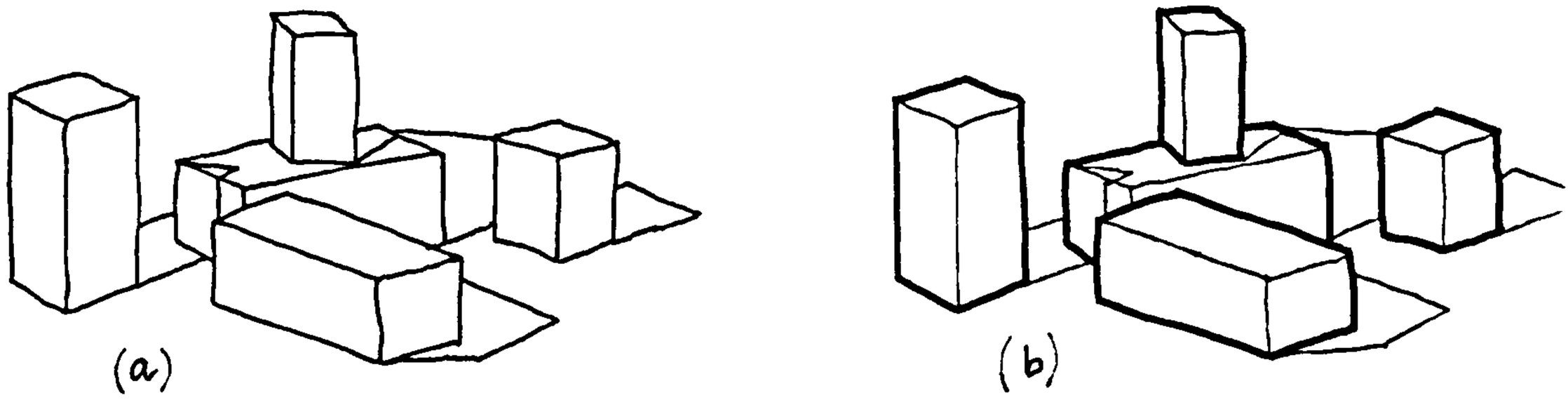

Figure 8.4
Scene Analysis by The Waltz Algorithm
(adapted from Waltz, 1975)

A system that could filter the edges from a scene and then resolve the volumetric nature of the objects of which the scene is composed could be a powerful tool in visual perception and a very useful aid to architects and other designers working on the built environment. There are no insurmountable obstacles standing in the way of constructing such a system, and it is likely that artificial intelligence will put a functioning scene analysis system into the hands of the architect in due course.

Machine Learning

To learn is to acquire knowledge. The activity of acquiring mere information is a lesser thing, and is usually known by such terms as rote learning or memorisation. The critique that Hubert Dreyfus directs at artificial intelligence is based upon this distinction, and the difference between the two things must assume importance in any discussion of machine learning. I think that Dreyfus's analysis is correct, and that knowledge is inaccessible to a computer for the want of a machine point of view. It is not fruitful, therefore, to discuss machine learning in terms of epistemology. However, a feature of human learning is that the behaviour of the learner is modified by what he has

learned. It is useful, then, to take a behaviourist approach to the matter and to consider to what extent a computer can be programmed to simulate the process of learning.

At the most elementary level, the performance of a computer can be changed by the simple acquisition of data. A data-base that, for example, outputs a different total when a new record is added fails the test of intelligence, for such a task calls for no more than straightforward reckoning. The relationship between input and output is a straight line, and the performance of the system is linear. However, a program, if it is to imitate the characteristics of human thinking, must display a non-linear correlation between data acquired and system performance.

The generalised programming technique whereby a computer can be set up to imitate human understanding is known as concept learning. In contrast to merely memorising data, the process of learning involves the acquisition of structures of information, or concepts. A person's behaviour is more likely to be altered by becoming appraised of a concept than just learning a fact. It is, for example, a correct datum that there are 100 pence in one pound sterling, but the notion of money as a medium of exchange and a measure of value is a concept.

A program procedure can be set up so that the receipt of one or more parameters has the effect of assigning values to a number of related variables. The group of variables constitute a concept, whose characteristics will vary according to the value of the parameter. A price list can be thought of as a simple type of concept, by which a group of objects are classified according to their monetary value. Input of object description, price or both will determine the nature of the output. The internal structure of a large programming concept, however, can be extremely complicated and its imitation of learning may be very life-like.

A concept may imitate learning by logical steps, as in a production system, by analogy, or by means of classification procedures. Logic is employed in the 'memo functions' algorithm proposed by Donald Michie in 1968. Semantic nets are proposed as a method of analogical learning by Patrick Winston (1980). The expert system shell Cortex that I describe in Chapter 13 enables complex concepts to be assembled by a classification algorithm, and its output mimics a process of learning from the input.

The evolution of living things over time can be seen as a kind of collective learning process. Evolutionary learning occurs in two stages. In the first stage the population reproduces in such a way as to produce new individuals whose characteristics differ from the parental stock. In the second stage environmental pressures favour the better adapted individuals by killing off those who are less well adapted. The new parental stock is thus composed of those who have had the greatest success in the race to survive. The population is, in effect, continuously winnowed by an environmental wind which disperses the chaff and conserves the grain for the next cycle of reproduction. The surviving population has learnt from the reproductive mistakes of its forebears by a process of involuntary adaption.

The American computer scientist John Holland has worked for some 15 years upon the notion that a computer program can be devised which will be able to adapt, or 'learn', by means of a kind of Darwinian selection process acting on its data. An illustration such as Figure 8.5 makes it hard not to regard a chromosome as a one-dimensional array, with each chromatid standing in the place of an array element.

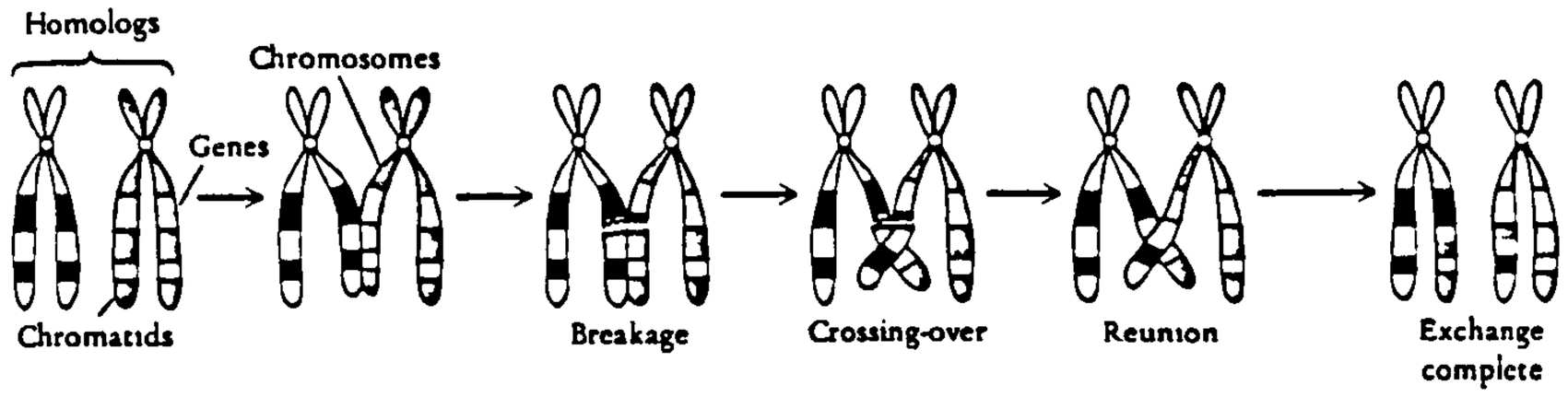

Figure 8.5
Chromosome Cross-over During Cell Division
(from Russell, 1986)

Holland's (1986) idea is to represent information by means of bit strings, and to form new concepts by recombining blocks of data on the analogy of chromosome cross-over. The new information structures can then be selected according to the success that they achieve when applied to problem solving. The method that he proposes to sift the new information structures is akin to Michie's memo-function procedure, and is referred to by Holland as a bucket-brigade algorithm. In effect, the most well-adapted structures are promoted, while those that fail the test of utility are demoted and eventually dropped.

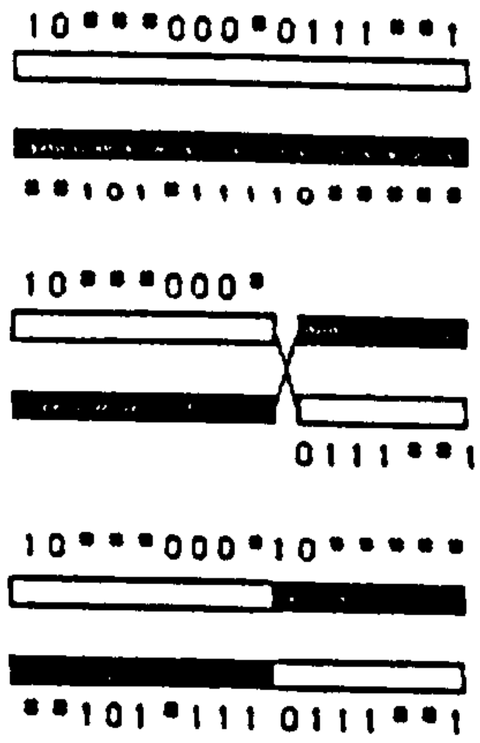

Figure 8.6
Bit-string Cross-over
(Booker, Holland & Goldberg, 1989)

The success of a genetic-learning algorithm hinges upon solving the problem of control. Holland proposed to exploit

the fact, taken over from genetics, that widely separated data will be crossed-over more frequently than items lying close together on the string. This feature can be utilised in a control scheme.

> "Contiguity of constituents, and the building blocks constructed from them, are significant under the cross-over operator. Close constituents tend to be exchanged together. Operators for rearranging the atomic constituents defining the rules, such as the genetic operation inversion, can bias the rule-generation process towards the use of certain kinds of building blocks." (Booker, Holland & Goldberg, 1989)

The most complicated and intractable of all learning concepts is the process of design. For the reasons put forward in Chapter 1 I think that design itself will always be uncomputable, but one can envisage design support programs that are capable of learning and which would be very useful to the architect. A technical analysis program that is set up to deduce a suitable energy conservation strategy from input about the design of the building, or another that is able to identify likely legal constraints from the configuration of the building would be of great value in architecture. Both programs would function as examples of machine learning, and there is no reason why artificial intelligence cannot furnish design aids of this type.

Search

In some situations the process of learning takes the form of a search. A subtle form of search process is involved in doing a crossword puzzle, for instance. But search is also an integral part of other areas of artificial intelligence. Solving problems, translating natural language, game playing, and theorem proving may all entail a search for a solution.

In principle any problem can be solved by examining all possible solutions as a preliminary to choosing that which is the best fit. However, exponentiation sees to it that

only the solutions to small problems can be found in this way. Noughts and crosses can be played successfully with a brute force algorithm, but that is a reflection of the fact that every schoolboy gets the hang of the game at an early age. Problems that are of interest in real life must be structured in some way if the combinatorial explosion is to be evaded and a computer is to be able to search for and find a solution.

The steps that must occur in a search for a solution to a problem can be represented as a tree, as a network, or as a classification scheme. What appear to be other ways of structuring the search space all resolve themselves into one of these three types. Games playing algorithms are usually structured in the form of a tree. A network is the structure which underlies the notion of scripts which was discussed in Chapter 2, and the same can be said of Marvin Minsky's suggestion of frames. Production systems, discussed in Chapter 8 also conform to the pattern of a network. Classification structures have been studied but little in artificial intelligence. This is surprising, for the notion of classification is the most general of all available concepts by which a search space can be structured. Indeed, at a sufficiently abstract level both trees and networks can be regarded as methods of classification. The expert system shell Cortex, which is described in detail in Chapter 13, is based upon classification principles.

It is very difficult to structure the search space of a large problem adequately. A thorough understanding of the problem in hand is needed, and a good deal of native inventiveness is also called for. But it is possible, when an effective structure has been found, to apply a number of well-established techniques to the task of conducting the search.

In the index to his textbook Winston (1984) lists 16 search methods, and there are few trees or networks that will not yield to one of them. However, the scoring system used by Peter Frey (1986a) in his expert system House.Bas is the only method of searching a classification scheme of which I am aware in the literature of artificial intelligence. In the description of Cortex I present another, and I believe better, way of searching a classification structure. Cortex proceeds by means of repeated cycles of counting, comparison and exclusion.

Search is not so much an artificial intelligence research topic in itself as a study that lies behind and which supports investigation into all the other areas of the subject. Structuring and traversing the solution space occurs in almost every artificial intelligence endeavour. Perhaps it is the artificial intelligence manifestation of the long-held belief that 'Nothing's so hard, but search will find it out.' (Herrick, 1648)

Control

In the decisions which form so large a part of everyday life we are guided by what are the known facts of a situation, but also by what we know about those facts. One may confidently cross a street when the pedestrian's traffic light turns green. However, if the light is already green when first seen the crossing may be postponed because one knows that the light will not remain green for long. Our human methods of decision are semantic, in the sense that what one does is a function of what are the facts and what one understands about the facts. But our human methods of arriving at a decision cannot be implemented upon a computer because the machine has no point of view and it is impossible to furnish it with understanding. It is necessary, when programming a computer, to use formal rather than semantic techniques.

A symbol, such as the spoken or written word 'cat' is interpreted in the English language to mean a feline animal. Other symbols, such as 'gatto' or 'chat' are interpreted in such a way as to possess the same meaning in Italian and French respectively. Natural languages are all interpreted systems, in which the symbol has meaning to an informed human user. This remains true of restricted languages such as Basic English or computer programming languages. It is not true, however, of other symbolic systems.

The variables 'p' and 'q' in a logical statement to the effect that 'p → q' are uninterpreted and carry no meaning. The symbol '→' is a connective which states that in all circumstances 'a' implies 'q'. It is a feature of formal languages, of which mathematics is the best known and the most pervasive, that the state of the system is exclusively
2 a result of the transformation rules acting upon the symbols of the system. For example, it is a rule of elementary arithmetic that a pair of the symbols '2' are transformed to the symbol '4' when the connectives are '+' and '='. The application of the formalisms of arithmetic to practical matters such as measurement, forecasting or resource estimation take place outside the formalism, and are semantic rather than formal in character.

Procedural programming, which is exclusively concerned with logical or mathematical operations, relies upon the well-established formation rules of these subjects. A procedural
3 program will solve the bracketed parts of a complex expression first, for instance, while every programming language makes use of the logical principle of modus ponens in the form of the IF-THEN formalism.

However, these methods are not adequate for the type of
4 declarative programming called for in artificial intelligence. The decisions that must be taken are too complicated, and fresh rules have to be devised to suit the nature of the programming declaration. A blackboard system, for

example, will attempt to choose from several the solution method best suited to the problem. But how is the best method to be decided upon?

> "The control problem is fundamental to all cognitive processes and intelligent systems. In solving the control problem, a system decides, either implicitly or explicitly, what problems it will attempt to solve, what knowledge it will bring to bear, and what problem-solving methods and strategies it will apply. It decides how it will evaluate alternative problem solutions, how it will know when specific problems are solved, and under what circumstances it will interrupt its attention to selected problems or sub-problems. Thus, in solving the control problem, a system determines its own cognitive behavior." (Hayes-Roth, 1985)

In a rule-based expert system a solution is found by working through a network of production rules. But in what order are the rules to be used? An intelligent tutoring system will try to furnish answers to questions according to the knowledge level of the user, which raises the problem of how the user's knowledge is to be assessed. This topic is known as the problem of control, and it makes itself felt in all aspects of artificial intelligence. Control, as it manifests itself in expert systems, is discussed in greater detail in Chapter 12 of this text.

Problem Solving

Some of the early workers in artificial intelligence attempted to develop a general-purpose problem solving program which would exhibit a capacity to learn. They hoped that a search method could be devised whose capacity to learn about its environment would enable it to be applied to a wide variety of cognitive problems.

It is observable that a feature of many types of problem is that a gap exists between the existing and a desired situation. In some cases a procedure, or operator, can be found which narrows or closes the gap. This is the simple notion

that underlies what is known in artificial intelligence as means-end analysis. However, complications arise as soon as the idea comes to be applied, beginning with the recognition that an operator may be able to close only part of the gap. If the remaining gap exists either before or after the operator it will need to work with other operators, while it may itself require another operator to close a gap in its own capability. In the first case the operators must be chained, while in the second they will need to be nested.

The best-known solver program was General Problem Solver, or GPS, written at Carnegie-Mellon University and the Rand Corporation under the direction of Alan Newell, Cliff Shaw and Herbert Simon. A distinction was made in GPS between problem-dependent parts of the system and those which are independent of the particular problem. This enabled the problem-independent parts to be treated formally, and they could in consequence be made computable. The formal method upon which the authors decided to rely was a 'table-of-connections' with which to select the appropriate operator. In practice GPS worked well with clearly definable puzzles such as the Towers of Hanoi or Cannibals and Missionaries, or with theorem proving tasks where the constraints can be completely described. GPS failed, despite its name, when it was applied to problems of a general type.

> "In the construction of a general problem solver, employing a fixed set of problem-solving techniques, the internal representation is critical; that is, it must be general so that tasks can be expressed in it; yet the structure must be simple enough for the problem solving techniques to be applicable. Since the techniques require that certain information be extracted from the internal representation, they are applicable only if processes that abstract the necessary information from the internal representation are feasible. Thus the difficulty of constructing a general problem solver is determined primarily by the variety and complexity of its problem-solving techniques." (Ernst & Newell, 1969)

This is a rather obscure way of saying that they found it impossible to design a table of operators appropriate to a problem which is only partially formulated. In fact, GPS was brought to a halt by problems of control.

It has, in practice, proved to be impossible to discover a set of control rules for a search method which is valid generally. The response of workers in artificial intelligence to this difficulty has been to restrict the scope of the problem to which a program is applied. When the domain of the problem is sufficiently circumscribed it is possible to devise a method of control whereby the program will work effectively. This is the reason, for instance, for the attention that is now given to expert systems rather than to general problem solvers. In Chapter 13 of this thesis I have attempted to contribute a solution to the type of control problem that occurs in the design of expert system shells.

Intelligent Artifacts

The topics proposed so far as making up the taxonomy of artificial intelligence are subjects of active research programs. This work opens the way to programs that exhibit increasingly intelligent performance, as well as contributing to more long-standing investigations. Research into classification systems and methods of search have contributed to mathematical combinatorics (Holland, 1986) while machine learning programs have thrown light upon educational psychology. Not all artificial intelligence research has a positive outcome. Work on language processing, for example, has done little but bring into sharper relief the profound nature and baffling complexity of natural language. Optimistic researchers in this field are now rare. (Bobrow & Hayes, 1985:382)

Nevertheless, programs which are examples of Peter Sell's 'intelligent artifacts' are emerging from 30 years of

theoretical work. These have taken the form of intelligent tutoring systems, and expert systems.

Intelligent Tutoring Systems

A textbook can, in artificial intelligence terms, be thought of as a knowledge base together with a problem-solving algorithm bound between cardboard covers. The author assembles the information he wants to communicate, and presents calculations, arguments and other routines such as indexes which are designed to make the topic accessible and useful to the reader. Some interaction with the reader may take place in the form of exercises, answers to questions and suggestions for further reading. However, the form of a book is fixed, and as a tutorial system it is essentially static. Nevertheless, books have served mankind well for some 3000 years and the future of civilisation, like its past, seems to be intimately involved with books and their use.

Every functional feature of a book can be reproduced on a computer, with the screen taking the place of the printed page. But a static program, reflecting the fixed nature of the book, would fail to take advantage of the possibilities of the computer.

All but the very simplest of computer programs are to some degree interactive. A program will solve an expression differently according to the input value of a variable. It is true that every reader obtains from a book something unique to himself, since reading is a mentally active process, but no book can modify its text according to information that it has received from the reader. The fact that a computer can be programmed to do just this, and to alter its performance in the light of response from the user, has lead to the development of entirely new forms of teaching and learning systems.

Early examples of teaching programs, such as the National Development Programme (Hooper, 1977), were hardly more dynamic than a book. The interactiveness of NDP was confined to question answering, and such static programs are now best referred to as examples of computer assisted instruction, or CAI.

The first intelligent teaching system, or ITS, was SCHOLAR written by Jaime Carbonell while a research student at the Massachusetts Institute of Technology. Carbonell's notion was to use artificial intelligence techniques in such a way as to obtain a response that varied according to the nature of a user's inquiry (Carbonell, 1970). This was done by using Ross Quillian's suggestion of semantic nets. In SCHOLAR the user's answer to a question prompted the output of the information at a network node and also invoked output from other connected nodes. Many routes would be available through a large network, and the particular route that a user found himself taking was partly selected by the character of his own input. SCHOLAR was in this sense interactive, and this is the basis of the statement that it was the first intelligent tutoring system, or ITS. However, the program suffered from the inflexibility of semantic nets, and is ill-adapted to represent the changing nature of the inter-relationships between concepts that is characteristic of matters in the real world. Furthermore, the program made no attempt to assess a student's learning needs.

If every student possessed the same style of mental apprehension and readiness of comprehension then the task of the educator, human or machine, would be greatly simplified. But the very ability to understand is a function of the possession of an individual point of view, and the learning performance of every student is therefore unique. A skilled human tutor will be able to communicate expert knowledge about a subject, but to be effective he will also need to know how to teach according to the state of understanding

of the individual student. When the tutor is a computer a comparable interaction is called for between student and machine.

Some of the features to be sought in an effective ITS have been listed by Johnson and Keravnou.

> "The characteristic features of a tutoring system are that it has:
>
> 1. The ability to evaluate the student's hypotheses and, in the light of the hypotheses, to criticise requests for additional information.
> 2. The ability to communicate (explain) to the student its strategies for attacking problems and demonstrate application of the strategies to concrete problems (probably problems formulated by the student).
> 3. The ability to answer (in terms understandable to the student) any relevant questions raised by the student.
>
> Additional, and very welcome,features of a tutoring system are:
>
> 4. Provision for unconstrained initiative on the part of the student.
> 5. Provision for a 'natural language' interface mechanism." (Johnson & Keravnou, 1988)

Between 1973 and 1982 the computer scientists John Brown, Richard Burton and Johan de Kleer developed their SOPHIE programs while working at Bolt, Beranek and Newman in Cambridge, Massachusetts. SOPHIE was designed to perform interactively with an individual student rather than to merely instruct an inquirer.

The performance of electronic circuits is a complicated area of technology which has been intensively studied. Brown and Burton chose the diagnosis of faults in electronic equipment as the domain for their ITS. Electronic troubleshooting relies upon a detailed knowledge of the technology, but it is not itself programmable. The operation of Kirchoff's current and voltage laws, for example, are well understood, but no algorithm is available as to how these

laws can explain a specific equipment failure. The effective application of knowledge in electronic troubleshooting is, as in so many other domains, a matter of skill. SOPHIE incorporates a large amount of information about electronics in its knowledge base, and attempts to use it in such a way as to engender this skill in its human students.

In its developed form of SOPHIE III Brown and Burton's program consists of three main components. These are the electronic expert, the troubleshooter and the coach. Information about the specific circuit as well as general knowledge of electronics is stored in the electronic expert, and this part of SOPHIE is therefore domain specific. Output of the expert is made in the form of deductions about the state of the circuit and about the values which measurements upon the circuit would produce.

The troubleshooter and the coach are both self-contained and independent of the particular circuit which is under consideration. The function of the coach is to examine the deductions of the electronic expert and decide whether or not to interrupt or to advise the student. The troubleshooter monitors the measurement values produced by the electronics expert and chooses the most informative one. These two components of SOPHIE worked well but the electronic expert, despite its name, raised some difficulties to which the authors could find no answers.

Two of the unsolved difficulties were caused by assumptions about component performance without which the program could not work. In the first place, the program assumes that a malfunction is the result of only one fault.

> "The single-fault presupposition is the most pervasive. Almost every deduction employed by SOPHIE III relies upon it. Without it a corroboration cannot logically be used to verify the underlying components nor can a conflict be used to verify the components that aren't mentioned in the underlying assumptions that

> lead to the conflict. Much of the behaviour of the reasoning mechanism of the behaviour-tree is no longer valid. (Brown, Burton & de Kleer, 1982)

Secondly, the program assumes that all possible faults are known. This may not in fact be so, for all possible modes of failure are impossible to foresee even in a field as well-studied as electronics. The third assumption listed by the authors of SOPHIE is the most subtle and gives rise to the greatest difficulty.

> "The third important presupposition, that a circuit symptom is a direct consequence of some component behaving symptomatically, is only true for circuits which do not have some kind of "memory". Suppose a device has a circuit breaker on its input which blew every time the power supply was plugged in. The power supply is manifesting a symptom, but every component is functioning correctly:... The problem is, of course, that the circuit breaker "remembers" that some component was behaving symptomatically, even though the component might not be doing so at present. This type of fault is notoriously hard to find since the trouble-shooter does not get the opportunity to see the faulted component manifest its symptom." (Brown, Burton & de Kleer, 1982)

The authors of SOPHIE conclude that although they suffered from the limited memory of their computer,

> "The issues concerning the need for a theory of human understanding of complex systems, in particular circuits, was clearly the more challenging one. Indeed, much of our recent research has been directed at attacking this problem. It quickly became clear to us that the work that went into SOPHIE II and III on explanation put the cart before the horse. We have no adequate theory of what it meant to understand a circuit and hence no well defined "target" model of what we wanted the student to learn. As a consequence no real theory of explanation was forthcoming." (Brown, Burton & de Kleer, 1982)

In recent years de Kleer and Brown have endeavoured to form a theory of understanding which will, they hope, correct the shortcomings of their ITS. Their notion has become known by the title of 'qualitative physics'.

> "qualitative physics yields qualitative descriptions of behaviour based upon qualitative descriptions of the physical situation and physical laws. The key contribution that makes qualitative physics useful and possible is that moving to the qualitative level preserves the important behavioural distinctions. For example, important concepts and distinctions underlying behaviour are state, cause, law, equilibrium, oscillation, momentum, quasistatic approximation, contact force, feedback, etc. These terms are qualitative and can be intuitively understood." (de Kleer, 1987)

However, this list of the features of a fundamental structure to perceived events is open to the same objections as is Roger Schank's notion of scripts. That is, the supposed units of qualitative physics themselves need interpretation if they are to be understood. I fear that this avenue of inquiry will lead the researcher into another infinite regress of meaning. Many observers confess themselves to be sceptical about the likely outcome of this line of investigation.

> "It is only fair to remark that the research programme advocated has been pursued by some of the best minds in AI for the last 20 years, but with very little to show for the effort so far. This lack of results has recently become the subject of some debate in the AI literature; codification of commonsense reasoning (based upon a naive understanding of physics) is generally regarded as one of the hardest unsolved problems in AI research today." (Akman, ten Hagen & Tomiyama, 1990)

Expert Systems

Expert systems are specialised problem-solving programs which are designed to apply to particular circumscribed problems. They are the result of an attempt to use a computer to penetrate a problem deeply on a narrow front,

rather than to merely scratch at the edges of a broad problem. Interest in expert systems is pragmatic.

> "We have already remarked that a shift took place in AI research over the past two decades. It was a shift from a search for broad, general laws of thinking towards an appreciation of specific knowledge - facts, experiential knowledge, and how to use that knowledge - as the central issue of intelligent behaviour. This shift came not as a consequence of irrefutable arguments that immediately persuaded all researchers by their cogency and correctness. Rather, the shift came about because demonstration projects that used large amounts of knowledge simply worked." (Feigenbaum & McCorduck, 1983)

The technical reason as to why the demonstration programs worked was not simple, but followed from the fact that it becomes possible to control a large amount of information when it is concerned with a single homogeneous topic. The restricted and defined topic within which an expert system is designed to work is known as its domain.

The distinction, first made in the design of GPS, between the problem-specific and the problem-independent parts of the program are a prominent feature of expert systems. In the jargon, the domain-dependent component is known as the knowledge base while the independent part of the program, which operates upon the knowledge base, is referred to as the inference engine. The distinction is a useful one, and it is maintained in the design of Cortex, because it distinguishes as clearly as possible the data with which the program operates. In procedural programming the difference between data and procedure is blurred, and much of the method of the program is encoded tacitly rather than explicitly. In Cortex, the knowledge base consists of files separate from the procedural code and whose informational content can be read in plain English.

Furthermore, the inference engine can, in principle, be used to operate upon more than one knowledge base and so be

applicable to several domains. A disembodied inference engine, complete with its control mechanism, is known as a shell. A shell is an expert system without knowledge. Cortex is a prototype shell, and it is implemented in Chapter 14 in the domain of an optical disc reader and the University College Dublin architectural video disc (Hastings, 1986).

If a layman consults a human expert, the solution that he receives to his problem will be better understood and is more likely to be believed if the expert can explain his solution. A patient will be more likely to improve his diet if a doctor explains that too much weight strains the heart, and a motorist will pay more attention to the choke if a mechanic explains that a rich petrol/air mixture damages the valves of an engine. Similarly, an expert system should be designed to give the reasons for proposing a particular solution to a problem. A computer program that baldly outputs a solution without an explanation will lack credibility.

Conclusion

I think that artificial intelligence can most usefully be classified under three main headings. These are cognitive simulation, mainstream topics, and intelligent artifacts.

Cognitive simulation, in either its strong or its weak form, is very unlikely to be successful. Knowledge and understanding are human facilities and are shared to some degree by all higher animals. The cognitive faculties of the knower or the understander are dependent upon the possession of a point of view, and therefore can only be exhibited by an entity which is not only conscious but self-conscious. A human knows something because he knows that he knows it. No computer, however powerful, can acquire a point of view, and therefore cannot escape an infinite regress of meaning. Searle and Dreyfus discuss these issues because they are interesting and fruitful, and

I think that their conclusions about the impossibility of cognitive simulation are correct.

The topics of mainstream artificial intelligence do not suffer from inherent contradictions, although the difficulties involved in natural language processing by computer are extremely daunting. Mainstream artificial intelligence is an active area of research, and will I think produce both understanding and useful programs in due course.

In the meantime, I have chosen to try to devise an intelligent program which will be useful in the field of architecture. My intention is to exploit and expand upon already-known techniques, rather to embark upon a task that cannot be completed without the results of future research work. Expert systems are at present the most highly evolved artificial intelligence artifact, and my effort in this thesis is to try to advance the design of expert systems to a further stage of development. The majority of working expert systems are written according to a formalism known as a production system. Production systems are logical in the strict sense of the word. Conclusions that are arrived at by employing such a system are valid if the premises are valid. However, production systems cannot, in my opinion, represent the process of design effectively and the conventional type of expert system that is based upon them is not suitable for use in architectural design. The next two chapters attempt to show why this is so.

Chapter 9. GRAPHS

In the opinion of some authors the whole of human life can best be described as an exercise in solving problems.

> "When we wish to explain the behaviour of human problem solvers (or computers, for that matter), we discover that their flexibility - their programmability - is the key to understanding them. Their variability depends upon their being able to behave adaptably in a wide range of environments." (Newell & Simon, 1972)

If the interpretation of the word 'problem' is made sufficiently wide, then everything can indeed be said to be a matter of solving one of an infinite array of problems. But a definition, when it is so all-embracing, loses its explanatory power and is apt to decline into little more than a truism. It does not seem to be particularly useful, for example, to be told by Matthew Arnold (1865) that criticism is "a disinterested endeavour to learn and propagate the best that is known and thought in the world." In a similar way, I think that Newell and Simon's characterisation of human beings as problem solvers is so broad that it explains almost nothing. I believe with Feigenbaum that, to be useful in the practice of artificial intelligence, the notion of problem solving should be restricted to discrete problems which are definable and which are amenable to solution. The range of problems to which knowledge engineering methods can be applied is circumscribed by practicality. The most useful forms of problem representation, for architects and other visually oriented people, are graphical in character.

Drawings

Techniques of graphical representation have been put to use from earliest times as a help in solving some types of empirical problem. The most pressing problem which faced palaeolithic mankind some 15,000 years ago was the hunt. Food, other than that which could be gathered, had to be

secured by hunting. Finding prey, therefore, and killing and retrieving it were the matters that were uppermost in the minds of Magdalenian men. The logic of sympathetic magic induced them to paint on rock walls vivid graphic representations of hunting parties and their prey. Those that were painted on walls deep in limestone caves such as those of Lascaux, Trois Freres and Altamira in southwestern France and northern Spain have survived virtually intact to the present day.

Writing of European prehistoric art, Nancy Sandars has observed that;

> "The special relationship between hunter and prey, that is nor only physical, gives to Palaeolithic animal art its peculiar power; the impression that the animals are not neutral. Without this relationship it is doubtful if there could have been any such art at all." (Sandars, 1968)

The notion, 15 millennia ago, was that a graphical representation of what was hoped for would help to realise those hopes. In modern times the progress chart pinned upon the wall of the foreman's hut sometimes seems to be connected to the state of the work on the building site only by a similarly slender thread of magical association.

Maps

Topographical maps do not have so great an antiquity as cave paintings, but they have been made by peoples in every corner of the earth for a very long time. The earliest examples to survive were drawn on clay tablets at about 2300BC in Babylonia, but map making no doubt began long before the third millennium.

A map is connected with physical reality not by the links in a chain of sympathetic magic, but rather it stands in relation to the world as an analogy. The first syllable of this word is the Greek preposition 'ana' meaning in this

context 'anew', signifying that an analogy provides a fresh and different account of something. The graphical symbols which make up the map correspond to features on the surface of the earth in the manner of an analogical representation of those events. The map is thus a fresh account of the actual geographical facts. While maps, like the wall paintings of our prehistoric ancestors, will often be beautiful objects to look at their primary purpose is not decorative, but rather they are intended to serve as problem solving devices.

Far away from the district that is represented, the user of a map can discover accurate information about areas, distances, depths and heights on the land. He can identify the presence of settlements and discover the uses to which the land is put. Furthermore, he can solve problems of travel. Because of the analogous character of a map, a route on a map will provide an itinerary which, when followed on the ground, will take the traveller to a destination which he may never have seen in actuality. A search conducted through the symbols on the map thus leads to the solution of a practical problem. These two notions, of symbolic representation and of search, appear in all branches of artificial intelligence. They are particularly prominent in the design and implementation of expert systems. The operation of using a road or rail map to conduct a journey is very similar to the process of searching the knowledge that is represented in an expert system in order to arrive at the solution to a problem.

Graphs

A well-known puzzle is based upon what was until 1945 the layout of the medieval centre of the city of Konigsberg in Prussia. The positions of seven bridges over the river Pregel that connect two islands to the mainland and to one another is shown in Figure 9.1. The problem that is set by the puzzle is, in a single continuous journey, to cross each bridge only once and to arrive back at the starting

point. Although it is not obvious from a topographical map, there is in fact no solution to this puzzle.

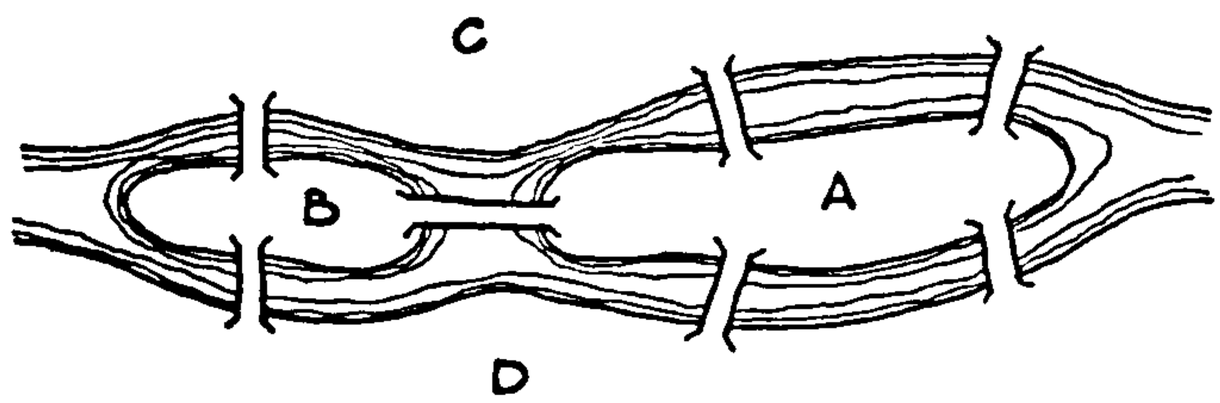

Figure 9.1 The Bridges of Konigsberg.

Figure 9.1 is a conventional topographical map of the analogical type. However, it is not well suited to represent the puzzle of the bridges of Konigsberg because of the fact that it is drawn to scale. The length of a bridge, or its exact rotational position on the periphery of an island is of no significance in arriving at a solution to the puzzle. In fact, the redundant information contained in Figure 9.1 is a disadvantage in this context because its presence makes the problem itself more difficult to think about clearly. However, by dispensing with the representation of scale it is possible to raise the diagram to a higher level of abstraction and so to reduce the depiction of the problem to it essential features. This is done by introducing the convention of the graph.

Those of us who are not mathematicians are accustomed to think of a graph as a Cartesian diagram. Two or three axes diverge at right angles from a common origin. The axes are scaled and a value in the field of the graph is determined by its position when projected onto the axes. A function can be represented on a graph by means of a set of such points. But a geometrical diagram of this sort is a particular type of the more general notion of a graph.

It is convenient to define a graph, in the general sense, as a diagram made up of only two types of component, points and lines. The set of points associated with a graph is

known as the vertices of the graph. A curve that is not self-intersecting is said to be a simple curve. The relationship that exists between the vertices of a graph is indicated by a set of simple curves, known as edges. The term node is often used for vertex in American texts.

If, for example, two related points are v_1 and v_2 , then the relation between them would be indicated by an edge which is designated by e = (v_1 ,v_2) or by the converse e = (v_2 ,v_1). Since the function of the edge is solely to indicate graphically that a relation exists between the two vertices, the length, straightness or layout of the curve is of no significance. This characteristic of an edge allows the topographical map in Figure 9.1 to be redrawn as a graph in this generalised sense of the term. Figure 9.2, made up only of vertices and edges, is the graph of the map appearing in Figure 9.1. The scale feature of the map has been dispensed with in the graph.

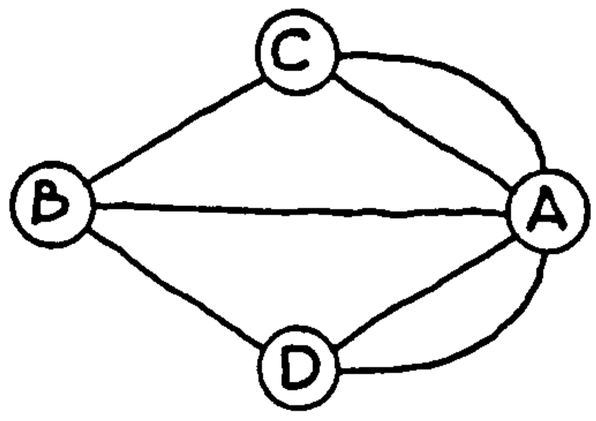

Figure 9.2 Graph of the Bridges of Konigsberg.

If there is a solution to the bridges of Konigsberg puzzle, then it must be possible to find a connected succession of edges through Figure 9.2 which traverses each edge only once and which returns to its starting vertex. A sequence of edges which traverses all the edges of a graph once only, and which ends where it began, is referred to as closed and, in distinction to a chain which is open ended, it is termed a circuit. It is possible simply from inspection to see that Figure 9.2 contains no such circuit. But a larger and more complicated graph would defeat the unsystematic observer. Is there, then, a general proof that the

bridges of Konigsberg puzzle has no solution? It turns out that a conclusive proof can be obtained by paying attention to the vertices of the graph. What is termed the degree of a vertex is the number of edges in contact with it. In Figure 9.2 vertex A is degree 5, while B, C and D are degree 3.

However, all the vertices which lie on a circuit must possess an outgoing edge for every incoming edge, which is to say that its degree must be multiples of 2 and therefore must be even. If the path is a chain rather than a circuit then two and only two vertices, at the open ends of the chain, may be odd of degree 1. In 1736 the Swiss mathematician Leonhard Euler generalised this rule to include paths which traverse a vertex more than once. He observed that each time a vertex is traversed by a circuit its degree is raised by 2, which is to say that its degree must always be even. Graphs in which all vertices are of even degree are known as Euler graphs. An Euler graph is therefore the only type of graph in which a circuit may be found. But since the vertices of the graph of the bridges of Konigsberg are all of odd degree it follows that no circuit is possible and therefore that the puzzle has no solution. This conclusion would hold true no matter how complicated the graph.

I have devoted some space to the puzzle of the bridges of Konigsberg because it shows that graphs can serve as surprisingly simple problem solving diagrams. Furthermore, a problem when represented as a graph can often be completely solved by calculation. A computer can, for example, easily be programmed to count the degree of all the vertices of even a very large graph. If all are even then a circuit can be found, and a chain through the graph exists if only two of its vertices are of odd degree. In this way the ability of a large number of edges to combine into a circuit or a chain may be computed quickly and economically. Furthermore, the Kirchoff-Bernoulli laws, relating flows and potential differences in a graph, are readily computable.

Such calculations are an important part of SOPHIE intelligent tutoring system described in Chapter 8.

For these reasons artificial intelligence programs for games playing, decision making and computer-aided learning often make use of graph theory. The application of graph theory to the type of graph known as a production system is of particular interest in the design and operation of expert systems.

Directed Graphs and Trees

An edge in the graph shown in Figure 9.2 indicates no more than that there is a relationship between two vertices. However, it is possible to use the convention of the graph to convey more than the bare fact of relationship. If the edge is given the attribute of direction, to indicate a flow of piped fluid or of electric current for example, it can be represented graphically by an arrow. A directed edge is known as an arc, and a graph made up of arcs is a directed graph. An arc designated by a = (v_1 ,v_2) is quite distinct from the arc a = (v_2 ,v_1). An entire directed graph can be represented by the expression G(V,A) where V and A are the sets of the vertices and arcs respectively.

Directed graphs can also represent non-material entities. An arc can, for instance, depict an energy flow or the flow of information from one vertex to another. When the graph is drawn in such a way that the vertices stand for decisions and the arcs for information flow, then what is represented is the structure of a logical process. When defined in this way graphs are useful devices for solving problems in logic.

Finally, one must take note of a particular type of graph known as a tree. The edges of the graph in Figure 9.2 divide the surface of the paper into 5 regions when the area surrounding the diagram is included in the count. A graph is a tree when, like the two diagrams in Figure 9.3,

it has only a single region. It follows that every pair of vertices in a tree is connected by exactly one edge, and that every sequence of arcs in a tree is a path.

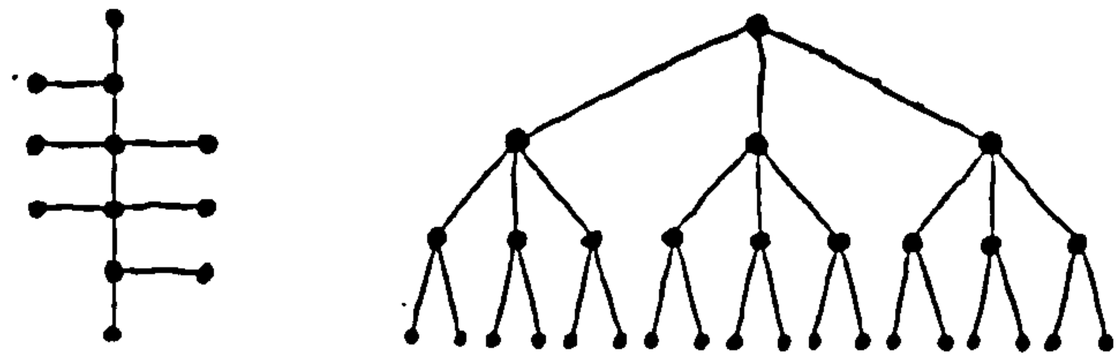

Figure 9.3 Trees.

A tree can, like a graph, be directed, and the problem of searching for a path in a directed tree has been and is an important topic in artificial intelligence.

Traversing a Simple Graph

A graph such as that shown in Figure 9.4 is known as a simple directed graph. The adjective simple denotes the fact that it has no parallel arcs and contains no feedback loops. Logic properly so called, which is deductive logic, is adequate for the purpose of traversing a simple graph but it will not serve to traverse a graph with one or more feedback loops. As I shall argue later, the process of design is characterised by the frequent and unavoidable occurrence of feedback loops. It is necessary, therefore, to consider the differences which exist between the formal properties of looped and simple graphs.

Let Figure 9.4 be a generalised representation of a process of reasoning about a problem.

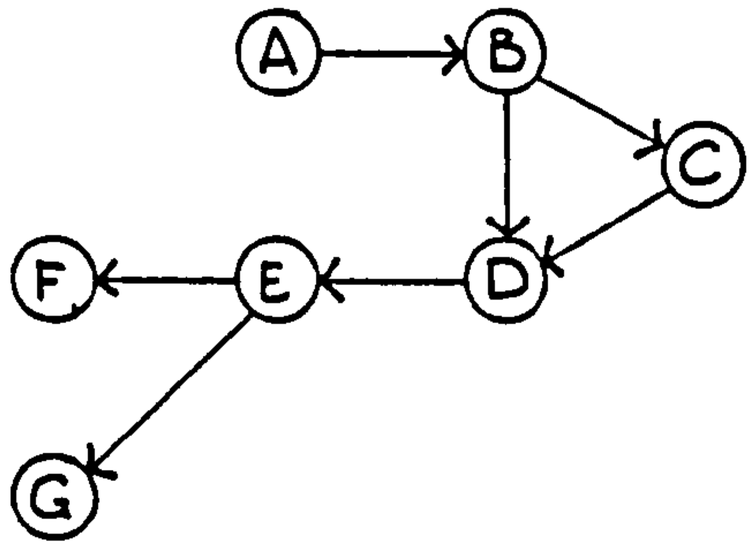

Figure 9.4
A Simple Directed Graph

The graph tells the reader that there are two possible solutions to the problem, denoted by vertices F and G, and that these solutions may be arrived at by two different routes of four or of five steps. At each vertex a decision as to whether to proceed or not must be taken, while at vertices B and E additional decisions are required as to which of the two succeeding arcs is to be followed. For our purposes it is sufficient to observe that each of these decisions can be taken independently of any subsequent decision. The graph contains no arc by which a decision at B, for example, would be affected by decisions taken later at vertices C to G. All deductive logical systems possess this feature, according to which a decision need never be reversed in the light of later decisions.

In deductive logic the flow of inference proceeds steadily forward from the premises until a conclusion is arrived at. Each inference is made by reference to a rule, and the resulting "well formed formula", or wff, ensures that the result is valid. The rules of logic are readily adapted to meet the requirements of the computer. For example, the rule of modus ponens in propositional logic emerges as the IF..THEN conditional statement to be found in some form in all computer languages. Quantifiers in the predicate calculus are conveniently represented on a computer as variables. The rule of double negation is accommodated arithmetically, and comparable representations are available for

all the rules of formal logic. It is therefore always possible to derive an algorithm to describe a wff, and wffs are consequently always computable. It follows that any process of reasoning that can be expressed in the form of a simple directed graph can be represented as a program and solved by computer. The converse also holds true. Undergraduates studying logic at Oxford University now make use of computer programs for the purpose of learning the rules of propositional and predicate calculus (Darby, 1988). However, the situation is less straightforward when the reasoning process under consideration includes feedback loops.

Traversing a Looped Graph

The graph shown in Figure 9.5 is identical to that in Figure 9.4 except that the direction of the arrow between vertices B and D has been reversed. The arc is DB rather than BD. The graph is no longer simple, for the three arcs joining vertices B, C and D form a feedback loop. The consequences of this configuration become evident when attention is given to vertex B.

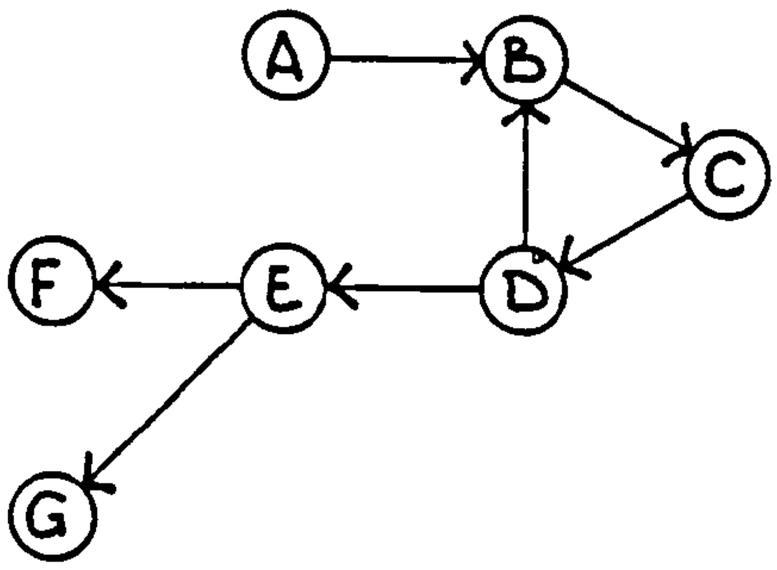

Figure 9.5
A Directed Graph Containing a Feedback Loop

Here, the decision to proceed along arc BC follows from the traversing of arc AB, but it is also dependent upon arc DB. That is to say, the decision at B is conditional upon a decision at D which has not yet been taken. This feature of Figure 9.5 violates the formation rules of deductive logic, by which no decision can be invalidated by a subsequent

decision. Were such a thing to occur, the argument would degenerate into mere circularity. There are therefore no logical rules by which the inferences which are depicted in Figure 9.5 can be guided. It follows that no algorithms are available, and that reasoning processes that are represented by directed graphs containing feedback loops are not computable.

Non-Monotonicity

One of the characteristics of inductive reasoning is that the conclusion reached may differ as new evidence is brought into the argument. The example of the revision of the notion that 'all swans are white' with the discovery of the black Australian swan was quoted in Chapter 1. This feature of induction has come to be known in the context of artificial intelligence as non-monotonicity.

> "'Non-monotonic' logical systems are logics in which the introduction of new axioms can invalidate old theorems. Such logics are very important in modelling the beliefs of active processes which, acting in the presence of incomplete information, must make and subsequently reverse assumptions in the light of new knowledge." (McDermott & Doyle, 1980)

The distinction between monotonic and non-monotonic reasoning brought forward by McDermott and Doyle has proved to be useful because it summarises a basic structural feature of all systems of inference. Every process of reasoning must be either monotonic or non-monotonic. It is evident, in the present context, that a simple directed graph is monotonic, while a directed graph containing a feedback loop is non-monotonic. I shall employ these two terms in the following discussion of looped graphs and computability. However, I shall not, for the reasons put forward in Chapter 1, employ the term logic or logical with reference to non-monotonic inference.

Non-monotonic reasoning processes do possess meaning, the absence of graph theorems notwithstanding. It is perfectly

reasonable for an architect to decide, for example, to clad a building under design in stone on condition that suitable stone turns out to be available in sufficient quantity. The decision is made provisionally, and in recognition of the fact that it may have to be revised later. In fact, a great many of the decisions that one makes in general life are meaningful in this provisional sense. Much of our thinking remains reasonable even when it lies outside the bounds of formal logic. Is it possible to bring computers to bear upon problems of this type nevertheless? It may be, perhaps, that an heuristic algorithm will meet the case and enable non-monotonic reasoning to be computed. The term 'heuristic' is used here in its artificial intelligence sense of "a technique that improves the efficiency of a search process, possibly by sacrificing claims of completeness" (Rich, 1983) rather than indicating something that stimulates investigation.

In an attempt to answer this question I return to a consideration of vertex B in Figure 9.5. It is possible to devise an algorithm which will enable the system to proceed from B to C provided that a later decision produces a certain predefined result. For example, the system might be set to traverse BC only if a variable at D exceeds a certain value. If the condition is not met the system loops back through arc DB and the process is repeated, perhaps with a different value for the variable. However, it is important to notice that the graph itself contains nothing which describes the character or the operation of this hypothetical variable. That is to say, the condition without which a non-monotonic system is paralysed is independent of the structure of the system. The condition that we have supposed to exist at vertex D in Figure 9.5 has been imposed upon the system rather than constituting a part of the graph itself. The graph, as a description of the process of inference, is incomplete.

The significance of this apparently inconsequential fact can be seen when the notion of incompleteness is expanded. Figure 9.6 is a variation upon the structure of Figure 9.5.

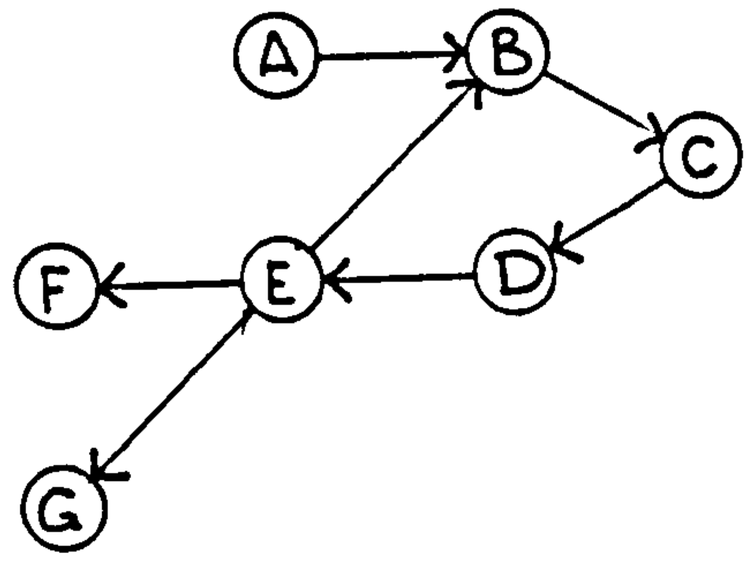

Figure 9.6
A Directed Graph Containing a Feedback Loop

Here the feedback loop is formed by the arc EB rather than, as before, by the arc DB. This graph might, for example, represent a situation in which the decision to proceed along BC was dependent upon whether EF or EG was subsequently selected at E. An alternative way of making the decision at B thus calls for the imposition of a fresh condition upon the system.

As in the case of the graph in Figure 9.5, the imposed condition implies knowledge of the system's context. The hypothetical variable at vertex D in Figure 9.5 might perhaps be a temperature or time interval, or a boolean state. The selection of arc EF or EG in Figure 9.6 could be made according to some measure of precedence, location or dimension. Contextual criteria such as these are all functions of the context in which the system of inference represented by the graph is designed to operate.

It emerges, then, that every distinct feedback loop in a graph calls for the imposition of at least one separate condition, and that every condition depends upon a knowledge of the system's context. Clearly, the parallel rise in the number of contextual conditions with the increase in the number of loops is an example of the infinite regress of context. But, as Hubert Dreyfus has pointed out, no

algorithm can be conceived of which can embrace an infinite quantity of knowledge. Despite the efforts of many ingenious logicians it remains true to say that non-monotonic reasoning is not computable, even by means of heuristics. It follows from these considerations that no algorithm can ever be devised which is capable of traversing a looped graph in an autonomous fashion. The difficulty has been put concisely by John McCarthy.

> "in order to fully represent the conditions for the successful performance of an action, an impractical and implausible number of qualifications would have to be included in the sentences describing them. For example, the successful use of a boat to cross a river requires, if the boat is a rowboat, that the oars and the rollocks be present and unbroken, and that they fit each other. Many other qualifications can be added, making the rules for using a rowboat almost impossible to apply, and yet anyone will be able to think of additional requirements not yet stated." (McCarthy, 1980)

A human being can make an exit from the regress of context, and so act successfully, because he possesses a point of view from which to judge the significance of the conditions and qualifications with which he is presented. His attempt to row across the river would be guided by what is succinctly described as common sense. Can a computer, which has no common sense, be programmed in such a way that it can, for its part, evade the infinite regress of context? Much ingenuity has been displayed in searching for an answer to this question (Winograd, 1980).

It is possible to decide that the rowing trip may go ahead unless a reason is known as to why it should not. This type of procedure, of belief in the absence of contradiction, is the basic notion of what are known as default logics. One might make a list of perhaps ten or a dozen reasons why the rowing trip should not be made, and then say that in the absence of any of these qualifications the trip may go ahead. These reasons can be stated as metarules (Reiter,

1980) or as a set of predicates. McCarthy (1980) proposed his circumscription method of default logic as a way of representing predicates and variables in a manner that lends itself to programming in LISP. In either case, and as the title of McCarthy's method suggests, the problem of regress of context is addressed by drawing a bound around that part of the problem context that is assumed to be pertinent. All other contextual issues are then assumed to be irrelevant. The quality of the automatic reasoning process is therefore entirely dependent upon the manner in which the conditions are specified. For this reason, default reasoning can only be applied to domains in which all the relevant conditions can be identified and described in advance. If, in the rowboat example, only qualifications about the size of the boat and the condition of the oars are included in the metarules or the set of predicates, then the system will not predict failure even if the hull has a leak below the waterline. Heuristic non-monotonic methods of reasoning, such as circumscription, are limited in scope because of the complexity of the context of any serious problem. For the same reason, a graph containing a feedback loop is a useful form of representation only when the problem is small and simple.

Architecture and Non-Monotonicity

The first of the 11 properties furnished by Horst Rittel in his description of design problems is that

> "Wicked questions have no definitive formulation. Any time a formulation is made, additional questions can be asked and more information requested." (Rittel, 1972)

The effect of this observation, whose truth no architect can deny, is that every decision that is taken in the process of designing a building is provisional, and subject to revision in the light of later events. For example, the orientation of a building may have to be changed when the number of spaces needing north light becomes known, the

building may need to be broken down into several units if it becomes to bulky, and in a few cases a good building is impossible to design on the proposed site and another location must be found. Reasoning processes of this kind, in which any decision may call into question any previous decision, are non-monotonic in character. Design is the most non-monotonic of all reasoning processes, because every decision can call for a reconsideration of not just some but every previous decision.

The conventions of graph theory break down when a completely non-monotonic process such as design is to be represented. If Figure 9.4, for instance, is redrawn to include the feedback loops characteristic of the process of design then the diagram no longer has any meaning.

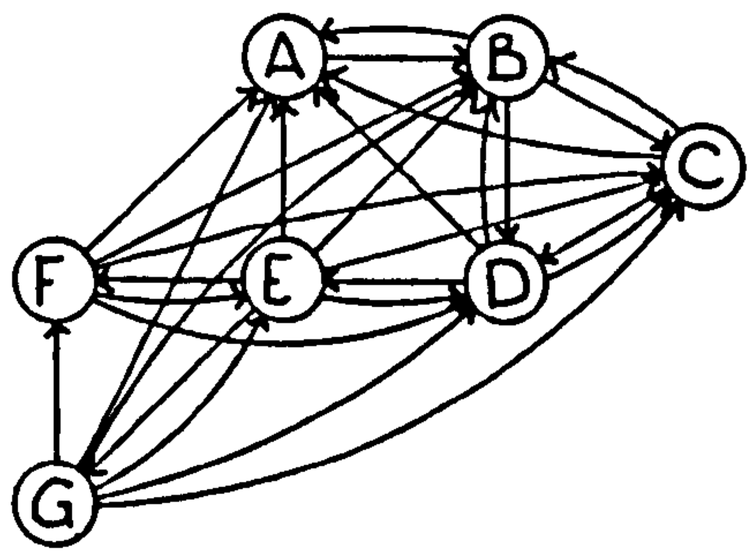

Figure 9.7
The Design Process as a Graph

The descriptive capacity of graph theory evaporates in a diagram such as Figure 9.7, and the truth preservation properties of deductive logic are destroyed by the presence of the feedback loops.

In the past attempts have been made, under the heading of design studies, to evade this difficulty. The best known effort to apply graph theory to design was that made in the late 1960's by Maver. The notion upon which his work was based was that a design proceeds from analysis of the problem to the synthesising of a solution. The solution is then appraised and the result of the appraisal is fed back

to a fresh attempt at synthesising a solution. The analysis-synthesis-appraisal model of design is illustrated in Figure 9.8.

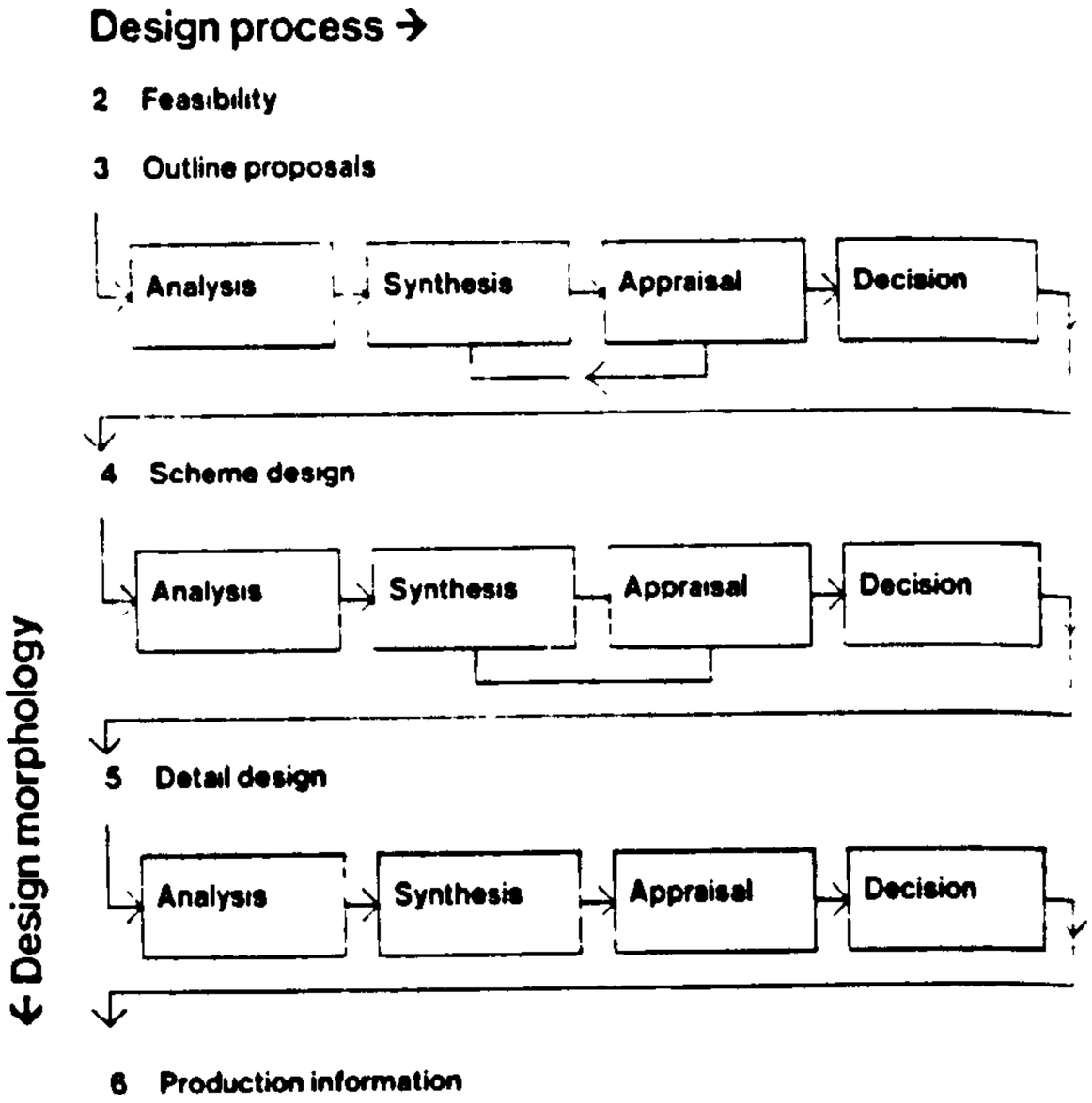

Figure 9.8
Framework for Design Management
(from Maver, 1970)

The need for feedback is recognised in this scheme. However, feedback is strictly circumscribed by the interpretation that has been given to design. There is no scope for the scheme design stage to influence the outline proposals, nor for detail design to feed back to either preceding stage. In fact, backtracking of this kind is discouraged.

> "The characteristic of a morphology is that the stages are sequential and not iterative; return from a later stage to an earlier stage is recognised as failure in the management of the design activity." (Maver, 1970)

In reality, this embargo is necessary if the breakdown in meaning which is illustrated in Figure 9.7 is to be avoided. But Figure 9.7 describes the necessary and inescapable process of decision making in design, and the fact that it

cannot be represented in terms of graph theory is responsible for the lack of interest on the part of architects in deductive models of design.

Conclusion

A simple directed graph is logically monotonic, and it can therefore be traversed by means of a suitably programmed computer. Problems of traversing a network in CPM or PERT analysis, which are monotonic in character, are routinely solved by computer. However, the pattern of inference of any graph which contains a feedback loop is non-monotonic. Because non-monotonic reasoning entails an infinite regress of context it is impossible in principle to solve this type of problem fully by means of a computer. Heuristic non-monotonic algorithms, which try to evade the regress, are limited by the scope of contextual considerations to small and simple problems.

These conclusions are very important when the problem under consideration is architecture. This follows from the fact that every inference in architectural design is non-monotonic and a graph which correctly represents the design process is replete with feedback loops. Computer programs applicable to architecture and design must, consequently, be able to function non-monotonically. It follows, also, that architectural design itself is inherently impossible to compute.

Chapter 10. PRODUCTION SYSTEMS

The methods of graph theory are capable of deciding if some path through a directed graph exists. There is, however, no general theorem which is capable of providing an answer to the more difficult question of which succession of arcs constitutes such a path. Operations like finding the critical path through a network representing a project, for example, can only be performed by numerical means. Dijkstra's algorithm, in which the path is assembled iteratively (Dijkstra, 1959), is often used to solve this problem in commercial network analysis programs.

The type of diagram to which conventional graph theory is applied has to be a static structure. Euler's rule would be powerless to find a circuit over the bridges of Konigsberg, were there one, if the graph in Figure 9.2 were to change while an answer to the puzzle was being sought. Similarly, if the sequence of operations on a building site which is being monitored by critical path analysis should be forced to change, then the revised situation must be represented by a fresh network if the new sequence of events on the critical path is to be rediscovered. The obligation to redraw the network frequently, so as to cast the operation into a new fixed form, is the main drawback to monitoring projects by network analysis. For this and similar reasons graph theory is a useful tool only when the problem under consideration has, at least for the time being, a fixed structure. This restriction excludes most of the problems of practical life, for events in the world are always in a state of flux. Proverbially, times change and we with them, and real-world events elude the static representation of a graph.

The Work of Emile Post

In the early 1940's, however, when computers were in their infancy the American mathematician Emile Post invented a novel type of dynamic directed graph through which it is

always possible to find a path (Post, 1943). Nearly all the expert systems that have been written during the last 25 years are based upon Post's formalism. Some space must therefore be devoted to his proposal.

Post begins his paper by observing that although the methods of symbolic logic can embrace an infinite set of variables, these methods apply equally well to a system that is defined as possessing only a finite sequence of symbols. Accordingly, Post's enunciates a formal system consisting of a pair of symbols, a primitive quantity 'a' and an operational variable 'P'. Quantities are described as primitive when they cannot be analysed within the system - in a particular situation they may be conceptually complicated while remaining primitive from the point of view of the system. A further symbol 'g' represents a finite sequence of a's and may include the null sequence. The three symbols are purely formal and may be held to represent any quantity or operation whatsoever.

Then the relationship of the quantity/operator pairs can be written:

$$g_{11} P_{i'_1} g_{12} P_{i'_2} \cdots g_{1m_1} P_{i'_{m_1}} g_{1(m_1+1)} \qquad (1)$$

$$g_{21} P_{i''_1} g_{22} P_{i''_2} \cdots g_{2m_2} P_{i''_{m_2}} g_{2(m_2+1)} \qquad (2)$$

$$\cdots\cdots\cdots\cdots\cdots\cdots$$

$$g_{k1} P_{i_1^{(k)}} g_{k2} P_{i_2^{(k)}} \cdots g_{km_k} P_{i_{m_k}^{(k)}} g_{k(m_k+1)} \qquad (3)$$

PRODUCES

$$g_1 P_{i_1} g_2 P_{i_2} \cdots g_m P_{i_m} g_{m+1} \qquad (4)$$

Figure 10.1
Canonical Form of a Production System

The sequences i' to i'_m and i" to $i"_m$ appearing in lines 1 and 2 are any finite sets of parameters. Line 3 of Figure 10.1 is therefore the description of the system for any sequence of parameters in any set of i. The symbol 'k' denotes the set of i. For any particular value the parame-

ter k can be cancelled out, to produce the expression in line 4. The formalism now consists of a small alphabet g, i, m and P, the single axiom that an operational variable may act upon a quantity, and the productions represented in line 4. Line 4 is described as the system in its canonical form because all other statements that emerge from the operation of the symbols derive from this first perfectly generalised statement. The canonical form states the 'general combinatorial decision problem' of the title of Post's paper.

Line 4 is referred to by Post as a production system because each term is produced from its predecessor and produces its successor entirely within the notational system. The word production, when used in this way, has no connection with practical activities such as making things or manufacturing products. Note that nothing is said about what the primitives or the operators consist of. Post's concern is with the formal relationships of the symbols and not with them as representations of entities outside the system.

Thus far the reader may be forgiven for thinking that Post's paper consists of little more than long-winded truisms. But the second stage in his argument makes the novel stipulation that the premises of a production must contain all the operational variables that appear in the conclusion.

> "We then add the restriction that each operational variable in the conclusion of a production is present in at least one premise of that production..." (Post, 1943:198)

That is to say, P_{im} in line 4 of Figure 10.1 must appear somewhere back down the line. The effect of this requirement is to make a production a cumulatively selective process. Operators, and consequently quantities, may be eliminated from the production system, but if they are then

any subsequent identical operator will also be removed. The usefulness of this stipulation will become apparent in the discussion of Alan Newell's adoption of the production system formalism in the 1960's.

The main body of Post's paper is taken up with the reduction of the canonical form of a production rule to its normal form. Normal here means establishing, rather than conforming to, a type or standard. This is carried out in four stages. The first step is to reduce the arbitrarily long sequence of assertions in the canonical form in Figure 10.1 to a single assertion. Secondly, the operational variables in the system are reduced to the one. Thirdly, the number of directions in which the system can proceed is reduced to one. Lastly, production is reduced to the form shown in Figure 10.2, which is a production in normal form.

$$gP$$

PRODUCES which may be written $gP \rightarrow Pg'$

$$Pg'$$

Figure 10.2
Normal Form of a Production

What this representation is saying is that for any set of sequences of primitives the repeated application of an operational variable to a pair of those sequences will always produce a new sequence in the set. A chain of productions, therefore, will not degenerate however long the sequence. It is Post's achievement to prove that the most general set of symbols can be combined by means of one remarkably simple rule.

It has become accepted terminology in the literature of artificial intelligence to represent a production as;

condition → action

This is equivalent to gP → Pg', even though Post himself never made use of these more recent terms.

Post's paper is written in a very abstract style and it runs to 20 closely argued pages. No commentaries upon it have been published, and for the lack of a detailed gloss I am unable to give as complete an account as I would wish of his reduction of the canonical to the normal form. Chapter 12 of Minsky (1967) is the best, but still unsatisfactory, explanatory text.

Production Systems and Artificial Intelligence

The interest of Post's formalism for artificial intelligence is that it offers a dependable mechanism by which a fresh tree can be created in response to a new structure of the data. Production systems evade the fixed character of fixed graphs. This feature was noticed by the American psychologist and computer scientist Allen Newell in 1966, when it was taken up by him as a promising way to structure a computer program which could, he hoped, simulate human thought processes.

> "Production systems are still a perfectly general scheme for information processing; they simply divide up the computation somewhat differently than a standard sequential programming language. The generality of production systems does not imply theoretical neutrality. They make it easy to express certain forms of organisation, hard to express others. Thus, they mould psychological theory to some extent. The issue will not be explored further in this paper, but its existence should be noted." (Newell, 1966a)

Throughout his paper Post discussed production systems as if they were one-dimensional phenomena. The lines of Figure

10.1 give a strong picture of linearity. However, the production rule formalism survives intact if the condition is the action of more than one preceding rule. It was Newell's achievement to realise the significance of this feature of the 1943 paper, which is only implied by Post's text, and so to place himself in a position to interpret production systems as a network.

During a discussion of the Firing Squad Synchronisation Problem, Newell suggests that the problem space, delineated by the square outline of Figure 10.3, can be thought of as traversable by a set of interconnected production rules.

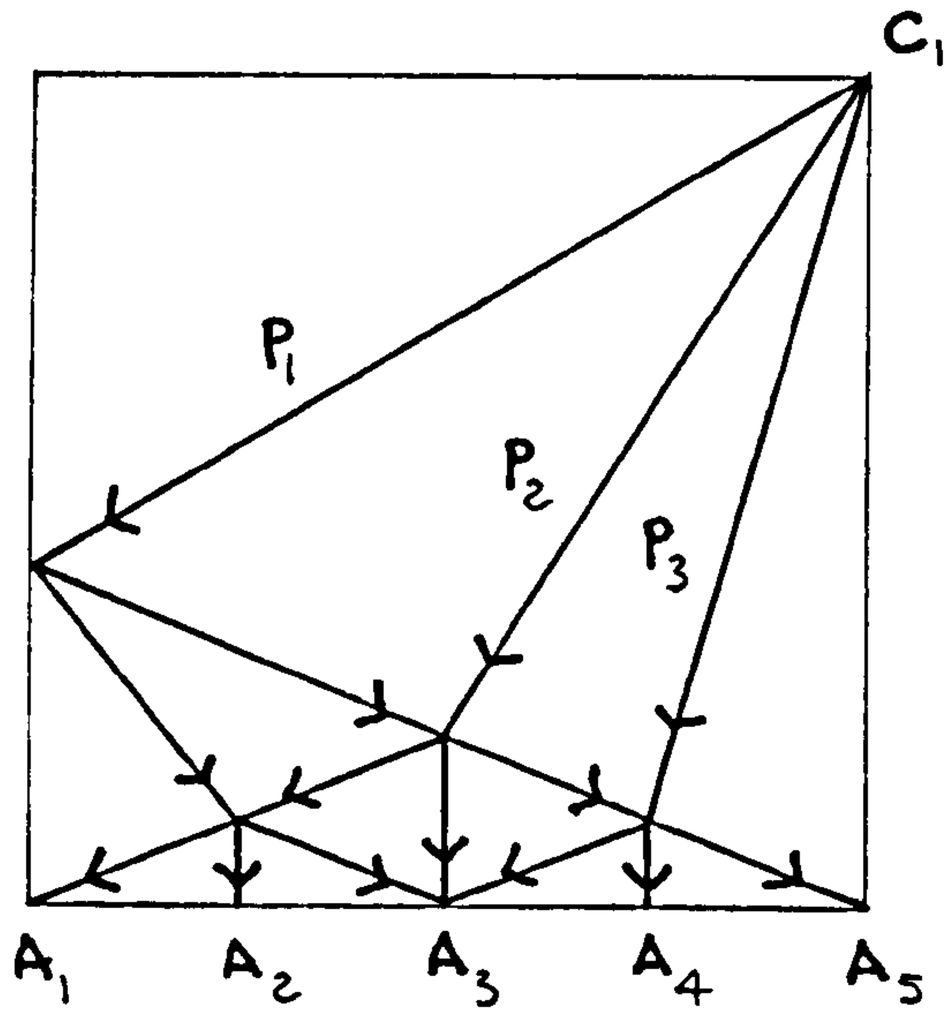

Figure 10.3
A Production System
(redrawn from Newell, 1966b)

In Figure 10.3 all the actions can occur if P_1 or P_2 are satisfied by C_1, while only A_3, A_4 and A_5 will result from the firing of P_3. Thus a production system can describe many possible processes by means of a single representation.

When a production system is interpreted according to Newell's insight it can be represented by means of a two-

dimensional network such as that shown in Figure 10.4(a). Vertices 1 to 7 are the conditions of 10 Postian rules, while vertices 19, 20 and 21 are the actions of five rules in this hypothetical system. The remaining vertices function as both conditions and actions according to which rule is under consideration.

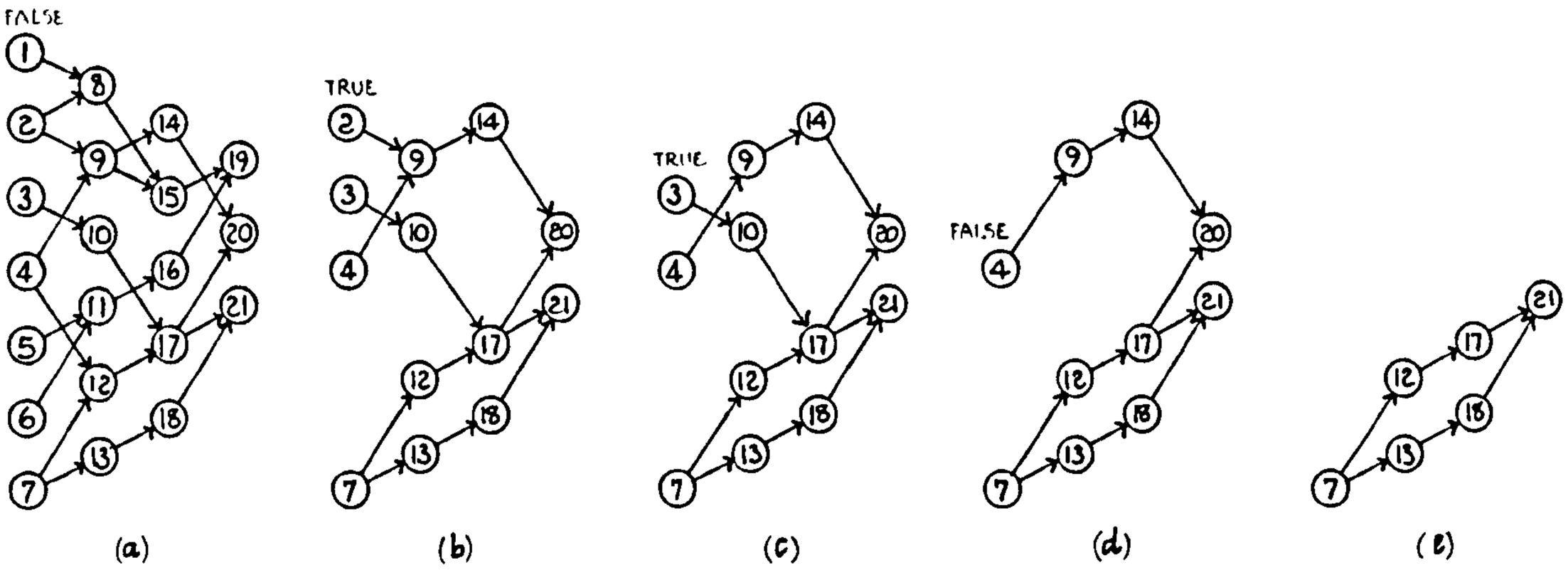

Figure 10.4
A Hypothetical Production System

The five stages (a) to (e) demonstrate the dynamic nature of a production system. If condition 1 is found to be false, then only the part of the graph in (b) remains active. The rules 1→8, 2→8, 8→15 and 15→19 are removed because vertex 1 is false, and rules 5→11, 6→11, 11→16 and 16→19 are also rendered inactive by the fact that action 19 has been proved false. If condition 2 in (b) is found to be true only rule 2→9 need be removed, and only rules 3→10 and 10→17 are lost if vertex 3 is true. If vertex 4 is false then the graph is clearly reduced to (e), in which action 21 is realised or not according to the state of condition 7.

Clearly the graph would modify into another pattern when the conditions are satisfied differently. It is the ability of a graph of production rules to adapt itself to information received about its environment that makes it a useful artificial intelligence tool. Production systems are particularly well adapted to representing domains which con-

tain many independent states. Problems of diagnosis, human and mechanical, frequently fall under this heading. There is no procedure capable of inferring the identity of a patient's illness deductively from a list of symptoms, for example, nor can the cause of a mechanical breakdown be calculated reliably from the observed malfunctions. The symptoms seem to be conceptually independent, but the source of the trouble can often be found by means of a search through the rules of a properly descriptive production system.

The DENDRAL Project

At the same time that Newell at Carnegie was adapting Post's idea to his own area of interest the geneticist Joshua Lederberg, the chemist Carl Djerassi and the computer scientist Edward Feigenbaum were at work at Stanford University on the other side of the North American continent. The topic with which their collaboration was concerned was organic chemistry rather than psychology.

The elements of which a chemical compound is made up can be found in a number of ways, one of which is the technique of mass spectrometry. If a compound is broken up into its components, by heat or bombardment with electrons, the resulting ions can be focussed into a beam and accelerated through a magnetic field. The path taken by the ions through the field will vary according to their mass, and the mass of all the components of the original compound can thus be measured. Furthermore, the ions can be counted and the spectrometer produces a result in the form of a histogram.

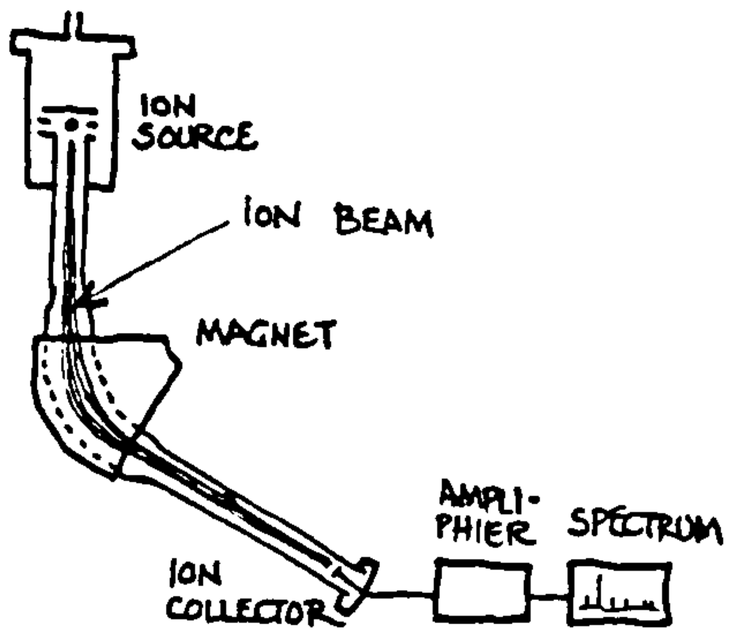

Figure 10.5
The Principle of Mass Spectrometry
(adapted from Gove, 1987)

It is no simple matter to interpret the output of a mass spectrometer. This is because any one spectrum of ions could represent the fragments of may different compounds. Which compound is the parent of the spectrum depends upon the regular but highly complex rules governing the ways which ions can combine with one another. The problem is at its most severe in organic chemistry because of the large size of many carbon compounds. A human chemist who is able to read a mass spectrograph, particularly of an organic compound, must have a large stock of knowledge about atomic and molecular structures at his disposal. Long training and much experience is needed to acquire this knowledge, and skilled mass spectrograph analysts are in short supply. The project undertaken by Lederberg, Djerassi and Feigenbaum at Stamford was therefore to write a computer program that could carry out organic mass spectrometry analysis at a human level of skill.

The outcome of their work was DENDRAL, an acronym for DENDRitic ALgorithm, in recognition of the tree-like nature of the program's search procedure. The first paper to be published on what later became known as Heuristic DENDRAL was Lederberg (1964). Heuristic DENDRAL works in three stages. Firstly, the operator provides a list of the likely compounds and a second list of those that are to be exclud-

ed as impossible. This shrinks the search space 'manually'. The second stage uses the program at the core of Heuristic DENDRAL, which searches a tree of structure building sub-programs and forms a list of all the molecular structures that are possible given the output of the spectrometer. Each sub-program is based upon what is known about a particular type of chemical compound, and the success or failure of the system is critically dependent upon the selection of the correct sub-program. Lastly, Heuristic DENDRAL outputs the mass spectra of all the solution candidates, and ranks them in order of goodness of fit with the experimental result.

Subsequent versions of DENDRAL, such as META-DENDRAL which incorporated an improved second stage known as COGEN, performed as well as a skilled organic chemist and deserve their reputations as the first useful expert systems. DENDRAL in its many versions functioned by means of a tree-search algorithm, and without using the formalism of a production system. The designers of DENDRAL attributed their success two features of the program in particular.

The first characteristic is that the program depends upon and makes use of a great deal of knowledge.

> "Behind the discussion of the information transfer process is the unquestioned assumption that the performance of Heuristic DENDRAL system depends critically on the amount of knowledge it has about mass spectrometry. Thus it is necessary to be able to add more and more theory to the program in the easiest possible way." (Buchanan, Sutherland & Feigenbaum, 1969)

Secondly, the program worked because its information covered only a restricted area of expertise, known as its 'domain'.

> "The Heuristic DENDRAL project, from 1968 to the present, and including COGEN, has produced a number of results of significance to chem-

> ists. The work has shown that it is possible for a computer program to equal the performance of experts in some very specialised areas of science." (Bennett et al, 1982)

No part of Heuristic DENDRAL is general or abstract in character. In this respect DENDRAL stands in opposition to GPS, which aspired to a general scope of applicability.

Production Systems and Programming

These two notions, of programming with knowledge and of using production systems as dynamic directed graphs, were brought together for the first time by Donald Waterman when he was working as a graduate student under the direction of Feigenbaum at Stanford University. In 1968 he submitted a doctoral thesis which contained what would now be called a rule-based expert system for playing poker. Waterman's program was published in the first issue of the journal Artificial Intelligence in 1970. In that paper Waterman acknowledges the help of Edward Feigenbaum and Allen Newell.

Waterman begins his paper by identifying what he calls the basic problem. The term 'heuristic' in this context means a technique for solving a problem by approximation.

> "Most heuristic programs to date have the heuristics built in as an integral part of the program. Even on close inspection it is difficult to decide exactly what heuristics are being used, what their effects are, and how they are related to one another. Then entire program, in a sense, becomes the representation of the embodied heuristics... In the scheme proposed in this paper, heuristics in a program are represented as an ordered set of production rules." (Waterman, 1970)

Waterman's ordered set of production rules would now be referred to as the program's knowledge base. The notion, introduced in this paper, of structuring a program so as to

distinguish clearly between knowledge and operations upon that knowledge has become one of the cornerstones of declarative programming.

The action of setting up a separate knowledge base brings in its train a difficulty that is not encountered in programs of a procedural style. Information in a computational procedure is processed according to the structure of the algorithm and control is exercised upon it in a tacit way. However, a knowledge base made up of production rules calls for an explicit control strategy if it is to be used correctly. Waterman employs a system of scoring designed to disfavour unsuccessful rules.

> "To make a decision via production rules, a symbolic subvector representing the game situation is compared to all left parts of the action rules, going from top to bottom until a match is found. The action rule which defines the decision, that is, the one whose left part matches the symbolic subvector, is easily located. After the decision is evaluated, the credit or blame can be assigned to the action rule, and to those above it, which defined the decision. Here blame is assigned to action rules leading to poor decisions, while action rules leading to good or acceptable decisions are ignored." (Waterman, 1970)

The final two thirds of Waterman's paper is devoted to using this scoring system of control to render the program self-modifying, or able to 'learn', in such a way as to improve its performance as a poked player. His work here is a continuation of that of Claude Shannon on playing the game of drafts, known in the United States as checkers.

Waterman's program displays three of the four distinguishing characteristics of what are now known as expert systems. These are the employment of the formalism of the production system, the separation of the knowledge base from the procedural parts of the program and the devising of a method of control. His poker playing program cannot, however, communicate why it has decided upon a particular

play. It possesses no facilities for explanation. Nevertheless, I think that Waterman's achievement is significant and his program can be regarded as the ancestor of all expert systems. The term 'expert system' was first used by Edward Shortliffe and Lawrence Fagan in a Stanford University research report of 1972.

Conclusion

Production systems have in recent years come into widespread use in the guise of expert system programs. Their success as a method of representing the knowledge of a domain follows from their self-modifying capability. The representation changes to reflect the state of the program's environment, forming a kind of mirror of events in the domain. However, a production system is a logical formalism in the strict sense that its inferences are deductive.

The deductive character of a production system can be gathered from a reconsideration of Figure 10.4. The rule 11 → 16, for example, was removed from the network in state (b) as a result of condition 1 being found to be false. This means that the rule cannot be used when the system is in subsequent states even if state (c) or (d) should bring it into relevance. Truth maintenance precludes the use of feedback loops in production systems, just as it does in every other type of deductive inference procedure.

I have argued in Chapter 9 that architecture, because of the non-monotonic nature of the activity of design, cannot be adequately represented by deductive logic. The necessity of introducing a feedback loop from each vertex to every previous vertex will defeat any deductive system. This is why expert systems, in the form in which they have evolved, are not used by architects in their role as building designers. However, non-deductive types of expert systems based upon classification methods are a programming possibility and promise to evade the restrictions of deduction.

Chapter 11. CLASSIFIER SYSTEMS

It is evident when analysing rule-based expert systems that the function of the network of rules is to place a set of individual productions into a correct relationship with a particular solution. On this interpretation, solutions are classified according to their question set, while questions are classified by reference to the solutions which they verify. The notion of classification is the basis of an alternative type of expert system which, as I hope to show, promises to be useful in the field of architectural design.

Classification

The concept of an expert system as a classifier can be illustrated by means of an example. Figure 11.1 shows, in the form of a production system, the well-known animal identification scheme proposed by Winston and Horn in 1981 and described by Duda and Gaschnig in the same year. I have chosen this small domain, which contains only 20 questions and seven solutions, because it is just big enough to illustrate the working of an expert system and because it calls for no specialist knowledge on the part of the user. The system in this representation has only 15 production rules.

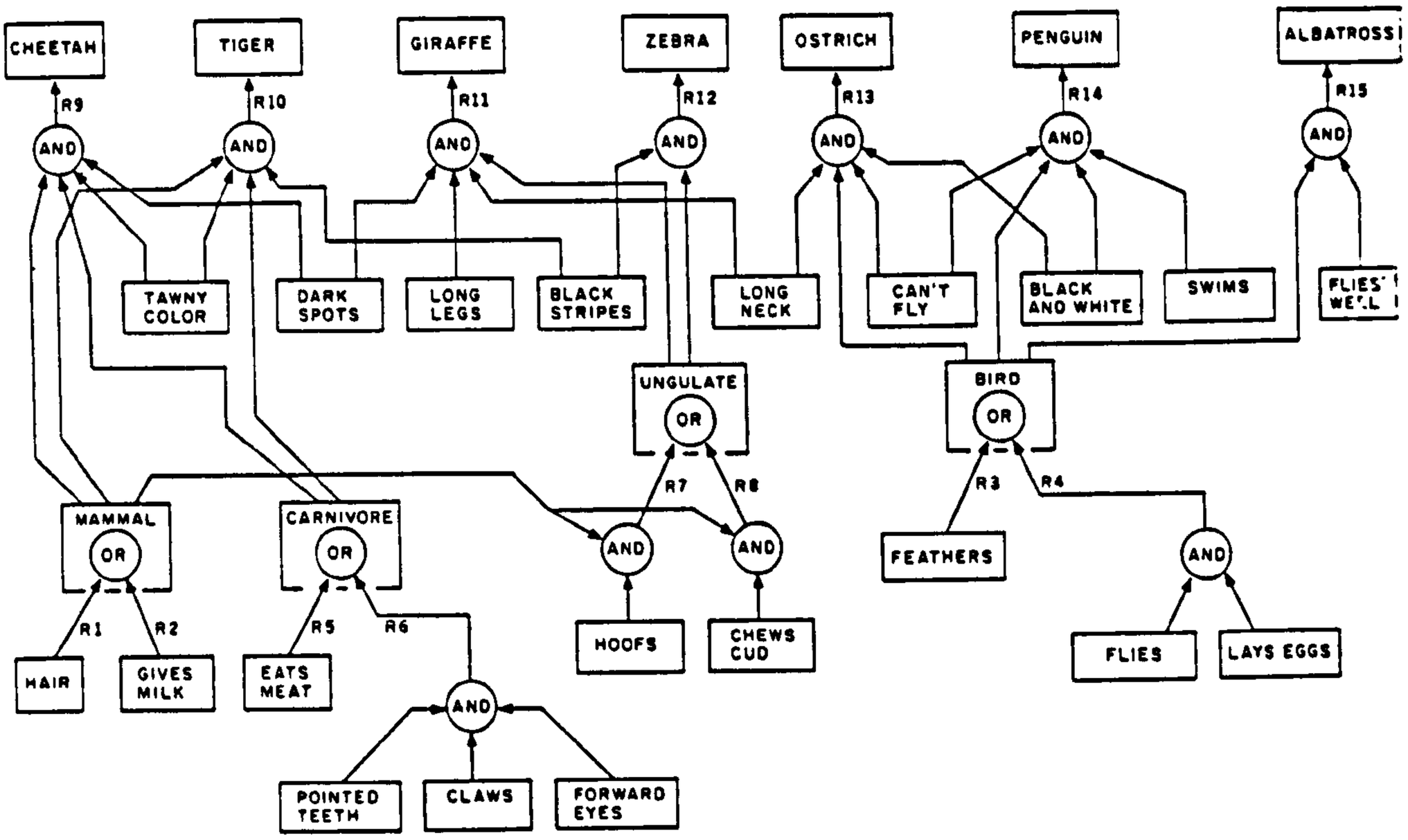

Figure 11.1
Animal Identification Scheme
(Duda & Gaschnig, 1981)

In this domain correct affirmative answers to the set of questions {lays eggs, flies, has feathers, flies well} results in the selection of the solution albatross as the identified animal. Similarly, the set {hair, gives milk, has hoofs, chews cud, has black stripes} identifies the animal as a zebra. There is a good deal of redundancy in this production system. An albatross cannot fly well, for instance, unless it can fly at all, while only mammals have hairy skins. However, it is a feature of a useful expert system that it can function properly when it is supplied with redundant information. Problems in the real world are very difficult to describe in a strictly logical fashion, and one of the main purposes in implementing a problem in the form of an expert system is to spare the user from unnecessary logical analysis.

The relationship between the 20 questions and the seven solutions can represented as accurately as a matrix as it cab in the form of a network. Figure 11.2 places the solu-

tions on the x-axis of a matrix while the y-axis contains the questions relating to the Duda and Gaschnig domain. Questions which must be answered 'yes' are marked by a black dot.

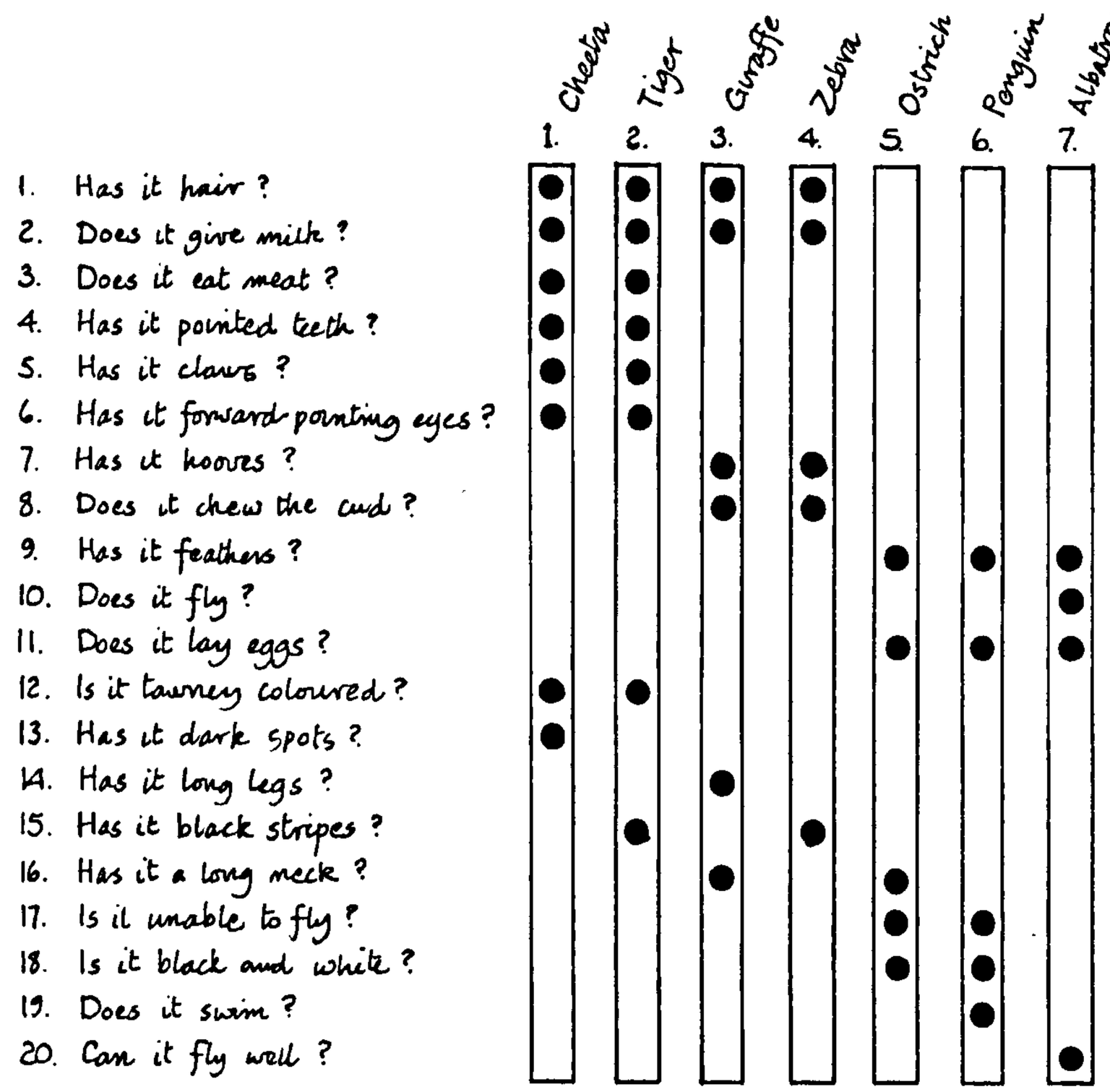

Figure 11.2
Duda & Gaschnig as a Matrix

Bit-Strings

In Pascal, as in most other programming languages, variables are stored as strings of binary bits. A character variable is stored as an eight-bit string, known as a byte, integers occupy the 32 bits of a four byte representation, while boolean variables are stored in the form of a single byte. A data type of particular interest in the present context is the array of integers. An array of integers is stored as a continuous sequence of four byte binary integers, and it therefore constitutes a bit string of arbitrary length. In Prospero Pascal the size of an array is

restricted to 64K bytes, which makes the largest possible array of integers the equivalent of a bit string containing 544,768 separate bits.

Bit-strings are a very compact way of representing information. By manipulating individual bits in the string, the presence or absence of 8 facts, or the truth or falsehood of 8 assertions, can be stored in memory within a single byte. Furthermore, bit-strings lend themselves to rapid processing, since an alteration to the state of a variable is only a matter of changing a single bit. This property of bit strings has for long been exploited in the architecture of data base programs. But the properties of bit strings can be used to represent data in a metaphorical manner as well as literally.

It is evident, from an inspection of Figure 11.2, that the seven solutions can each be thought of as a variable represented by a bit-string consisting of 20 binary bits. Furthermore, the bits are ordered in such a way that the questions are represented consistently. That is to say, the ninth bit represents the question "Has it feathers?" in all seven strings. Answers to questions can be recorded by setting the appropriate bit to 0 for false or 1 for true. It can be seen that the solutions in this toy domain, where few questions need to be asked, can be represented as integers. In a useful expert system, where some hundreds of questions may be needed, the 32 bits of an integer string is too short, and a longer string composed of an array of integers will be called for.

This notion of an expert system as a classification of knowledge represented by bit-strings was taken up by Peter Frey, of Northwestern University in Illinois in the early 80's. In 1986 he published a description of his invention in the form of a magazine article and an accompanying 5¼" floppy disk (Frey 1986a and 1986b). The article describes, in a rather terse fashion, the design of his classifier

shell while the disk contains the listing of an implementation of the system. The implementation takes the form of a method of identifying domestic house styles in New England, and it is therefore of direct interest to architects.

The logical operations of a rule based expert system written in a conventional programming language are performed by a network of IF-THEN statements, while a shell in Prolog will proceed by trying to match a stated predicate with all the predicates in the knowledge base before moving on to the next rule. Both these operations are prodigal of processor time. Conventional expert systems, particularly those written in Prolog, are in consequence notably slow in operation.

However, strings of bits can be compared very concisely and economically by means of a pattern matching process such as that illustrated diagrammatically in Figure 11.3.

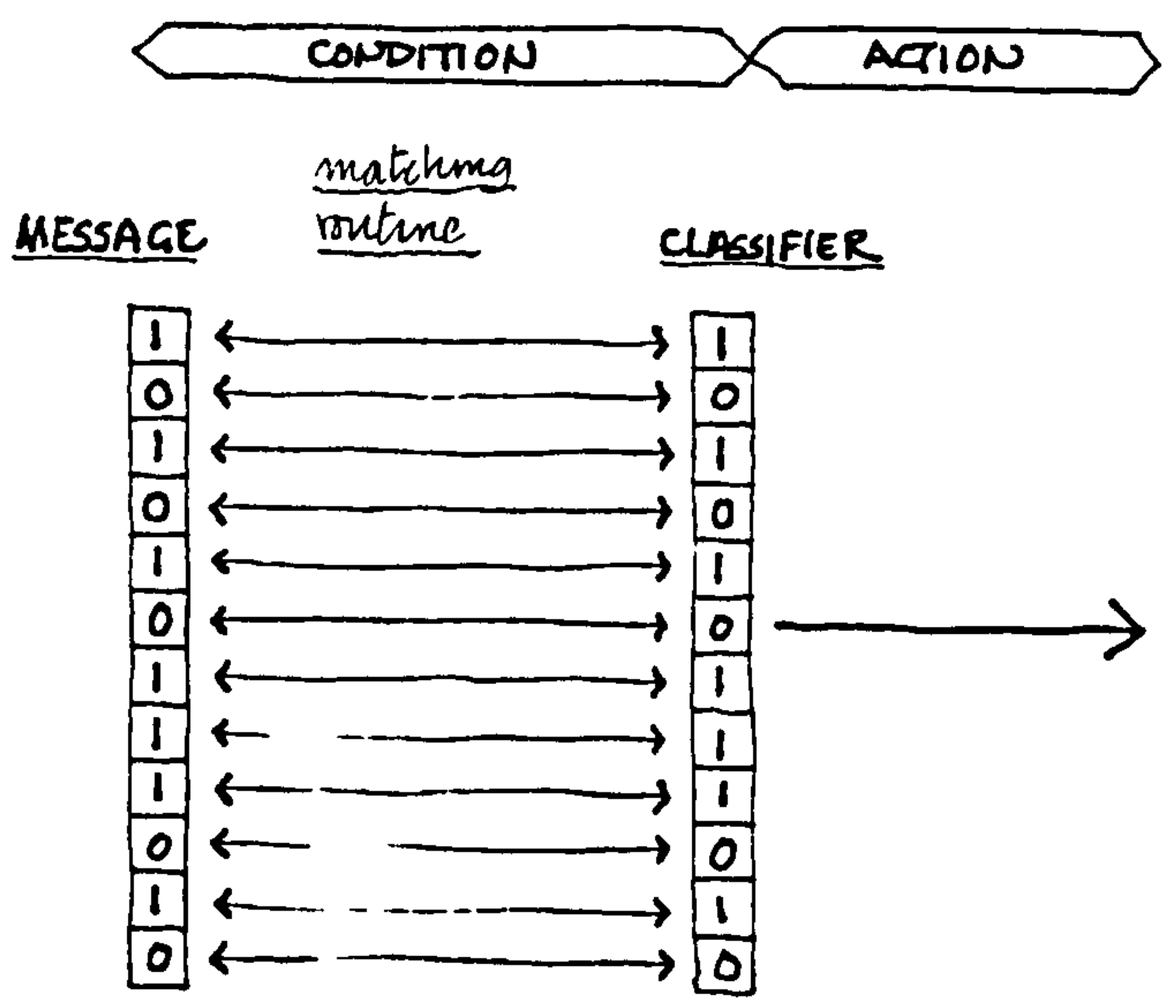

Figure 11.3
Classification by Pattern Matching.

In this diagram the term 'message' is adopted from Holland's work, referred to in Chapter 8, to denote a body of information about the current state of the environment of a system, while his word 'classifier' refers to a bit map

whose configuration represents the features of a situation. The two strings are shown in the diagram as corresponding to one another, although this may or may not be true of a particular case. I have followed Frey's example and have employed Holland's terms in the remainder of this text and in the design of my expert system shell Cortex.

Each horizontal arrow in Figure 11.3 indicates a matching of two binary bits, and achieves the same result as processing a rule in a production system. A pair of bit strings which were 33 bits long could carry out all the logic of the productions system shown in Figure 11.1 as 33 binary bit comparisons. This operation places a far smaller load on the processor than does a chain of conditional statements or the predicate matching operations of Prolog. It is one of the attractions of a bit-string based shell that logical operations can be carried out simply by comparing individual bits. The computational economy of the procedure opens the way to implementing large and very fast expert systems on table-top microcomputers.

The Frey Algorithm

In the expert system shell proposed by Frey the relevant facts about the topic of interest are recorded in a bit string known as the classifier. The pattern of this string is provided by the knowledge engineer, since its configuration is dependent upon an accurate and detailed knowledge of the domain. The classifier must, of course, be written beforehand as an essential step in any implementation of the shell. Answers provided to the system by the user are recorded in a second bit string known as the message. The system works by matching the pattern of these two strings. But the elements of both these strings are binary, while the facts to which they refer are many-sided. A false bit in the message may record either that the answer is negative, or that the question has not been answered at all. The classifier must be able to distinguish between the

features of the domain that are definitely known and those about which knowledge is imprecise or conjectural. In a system for identifying the style of a building, for instance, a mason's mark can securely date a building as Medieval, but plate window tracery may be attributable to the Gothic or to the Victorian eras. A useful expert system ought to be capable of making this kind of distinction.

That binary logic can serve to represent multi-dimensional entities was recognised by George Boole himself when he observed that;

> "We may in fact lay aside the logical interpretation of the symbols in the given equation; convert them into quantitative symbols, susceptible only to the values 0 and 1; perform upon them all the requisite processes of solution; and finally restore to them their logical interpretation." (Boole, 1854)

To make use of what in recent years has become more familiar terminology, Boole observed that any degree of logical complexity can be represented by a sufficiently large binary tree. Frey utilises this same principle by proposing to incorporate in his system additional bit-strings whose function is to specify the relative importance of bits in the message and classifiers. These additional strings he calls 'masks'.

Under the influence of the American behavioural psychologist Eleanor Rosch, Frey has adopted a three-fold model of the classifier for use in his expert system. Rosch (1977) observes that cognitive categories are conventionally thought of as discrete.

> "most studies carry the unexamined assumption that categories are arbitrary logical conjunctions of critical attributes which have definite boundaries and within which all instances possessing the critical attributes have a full and equal degree of membership." (Rosch, 1977)

Against this notion she argues that categories are characterised by the possession of a salient attribute around which other attributes are closely or loosely grouped. She concludes;

> "that color, and possibly form, categories appear round perceptually salient points in the domain and that such points form cognitive prototypes for the categories of those domains." (Rosch, 1977)

Rosch employs a metaphor drawn from computing when she likens the first notion of categorisation as 'digital' while describing the second and more adequate conception as 'analogue'. In coming to this conclusion she is, perhaps unknowingly, placing herself in agreement with Hubert Dreyfus's observations about the analogical character of human thinking.

While discussing the structure of his expert system Frey observes that "Eleanor Rosch has argued persuasively that natural categories do not have fixed boundaries" and he goes on to propose three classifier masks designed to model the Roschian categories.

> "To represent this aspect of categorising people, objects or events we employ three classifier masks specifying which attributes are absolutely essential to the category (type A), which ones are usually present (type B), and which ones are sometimes present (type C).This strategy permits flexibility in defining category prototypes (the classifiers) that seem necessary for real-world applications." (Frey, 1986a)

By way of these considerations Frey arrives at a design for an expert system which makes use of a message and a message mask, together with a classifier and three classifier masks. All five elements are represented in the computer as bit-strings. The manner in which these strings work together is shown in Figure 11.4.

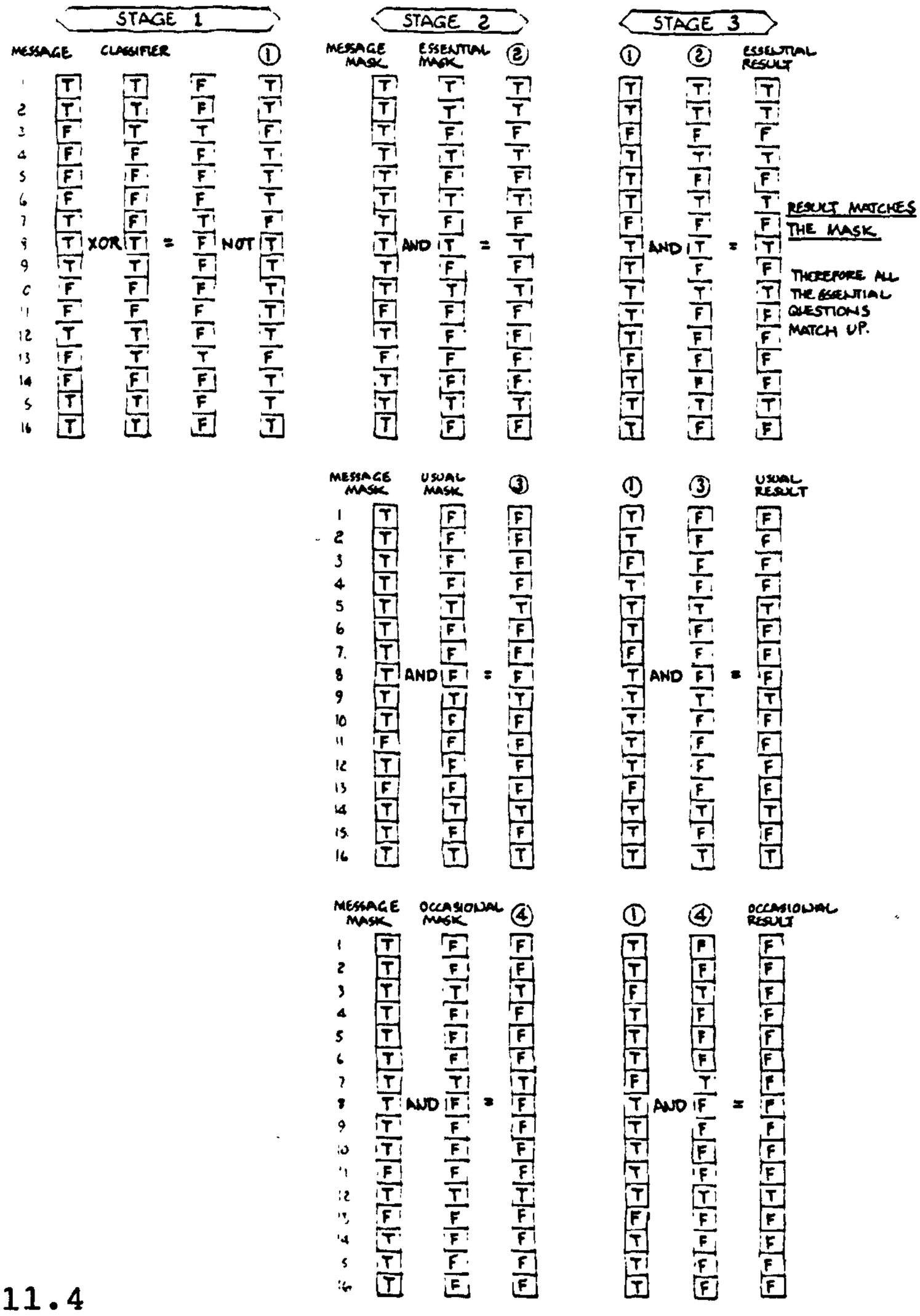

Figure 11.4
The Frey Algorithm

The message is a string of bits which records the answers given by the user of the system to the questions. A bit set to T in the message records that a question has been correctly answered, while an F means either that the answer given was wrong or that no answer was supplied. The answers that are needed if a particular solution is to be true are written into the classifier by the knowledge engineer.

In stage 1 of the algorithm the classifier and the message are compared by the XOR logical operator. This returns a T

when the operands differ, which occurs when the answer required by the classifier differs from the answer to the question. A NOT operation then reverses this result and produces intermediate result 1, in which a bit is set to true when the answer and the required answer are the same, and to false otherwise.

Because of the ambiguity of the message, a second stage is needed to distinguish a wrong answer from no answer at all. This is accomplished by stage 2. Here the message mask is ANDed with the essential mask of the classifier. The message mask records whether or not a question has been answered, while the essential mask identifies by means of a bit set to T those questions which must receive a correct answer. The AND operator returns T when both bits are true, and F otherwise. This result is recorded in intermediate result 2.

Finally, stage 3 ANDs the two intermediate results. The effect of this is to return T if the question is an essential one and it has been answered correctly. Otherwise the bit is set to F. A solution achieves a possible status when the bit string called essential result in Figure 11.4 is the same as the essential mask.

In a similar manner, when the message mask and the usual mask are ANDed and the result is ANDed with intermediate result 1, a usual result bit string is produced. This is set to T when a usual question has been correctly answered, and F otherwise. The correct answer to an occasional answer is identified by the same method, but this time involving the occasional mask in stage 2. In Figure 11.4 all four usual questions are shown as receiving correct answers, but the answers that appear in the message to two of the three occasional questions are found to be wrong. These bit matching procedures are carried out by the ADJ subroutine of House.Bas.

Frey says that his system is designed to reflect the ideas of Rosch on the nature of classification. The three-fold result produced by the procedures illustrated in Figure 11.4 must therefore be weighted in such a way as to reflect their status as type A, B or C attributes. This is done, in his House.Bas program, by attaching a score to each of the questions. As the file name extension indicates, the program is written in BASIC.

In the first place, the questions relevant to the domain of House.Bas are recorded in a long list of data items. The following is an excerpt from the questions data list.

```
REM   ***   COMPOSITION OF EXTERIOR WALL   ***
DATA 10, "wood exterior"
DATA 11, "stone exterior"
DATA 12, "brick exterior"
DATA 13, "stucco or adobe exterior"
DATA 14, "combination of wood and masonry or
stucco"
DATA 15, "unconventional exterior cladding"
```

The second part of the data section of the program is a list of classifiers. The first line of a classifier consists of an index number, a name string and a threshold value. For example;

```
DATA  27, "Richardsonian Romanesque", 30
DATA  3,11,999
DATA  71,-83,101,999
DATA  43,64,126,129,151,999
```

The three lines ending in 999 are the essential, usual and occasional classifier masks. It is essential that the massive masonry character of the style of H.H. Richardson should be reflected in a house built in his style, and consequently question 11, "stone exterior", is a question that must receive a correct answer in this case. If it does a contribution of 5 is made towards the threshold value of

30, while an incorrect answer scores -99. The effect of a wrong answer to an essential question is to exclude those classifiers to which it is relevant.

Questions which require negative answers are distinguished by a negative data entry. Question 83, for example, reads "symmetrically placed windows about a centered front door". The picturesque massing of a Richardsonian composition would preclude a symmetrical facade, and this question must therefore receive a negative answer.

When a usual question is correctly answered a score of 5 is added to a classifier's score, while an incorrect answer deducts 5. A question about a feature which is sometimes relevant also scores 5 if correctly answered, while an incorrect answer deducts only 1. In this way those questions which are identified by the knowledge engineer as essential to the identification of a classification are given a veto over the choice of a solution. Questions which relate to features usually present in the answer contribute to or detract from the score even-handedly, while those questions which are sometimes relevant may contribute to the score but can only slightly reduce it. In this fashion Frey implements the ideas of Rosch concerning human categorisation. The scoring operations of the masks are also carried out in the ADJ subroutine of House.Bas.

After each question is answered by the user of the system the program selects the three classifiers with the highest score. The questions that are relevant to these classifiers are then selected for presentation to the user. In due course the threshold value of one of these classifiers is reached, and it is then declared to be the correct solution. In this manner the program is so arranged as to concentrate upon the most likely solution. As Frey observes, "This strategy approximates the hypothesis-testing approach that is commonly observed in humans."

In Chapter 4 I have given an abbreviated account of the objections that Hubert Dreyfus has made to the computational model of human thinking. During the course of a discussion of the 'brute force' type of chess playing program Dreyfus contrasts the machine and the human ways of assessing a problem.

> "We need not appeal to introspection to discover what a player in fact does before he begins to count out; the protocol itself indicates it: the subject 'zeroed in' on the promising situation ('I notice that one of his pieces is not defended'). Only after the player has zeroed in on an area does he begin to count out, to test, what he can do from there." (Dreyfus, 1979)

This is closely analogous to the two main stages of the Frey algorithm. These are, firstly, a concentration upon a promising solution, and then the calculation of its probability. The close parallel gives one good hope that the algorithm will be useful in areas of application where, as Dreyfus points out, conventional computational methods have been largely unsuccessful.

Critique of the Frey Algorithm

The algorithm which has been described in outline in the previous section is an original contribution by Peter Frey to the literature of AI. It seems to me to be most ingenious, admirably original and to be very relevant to the practice of design.

In Chapter 10 I have argued that expert systems based upon formal logic cannot be applied to design problems successfully because design is not a deductive process. The Frey algorithm, which is based upon classification rather than logic, evades this difficulty and it therefore promises to be useful in the practice of design. Of particular importance is the ability of a classification system to accommodate feedback loops simply by structuring the classifier and its masks. The Frey algorithm, unlike rule-based expert systems, is well adapted to non-monotonic logic.

Input and output in the Frey algorithm is by means of character strings. It is therefore able to make use of the interpretive power of natural language at the point of interface with the user. (Finin, Joshi & Webber, 1986) However, its internal functioning is numerical rather than logical or linguistic and it is this feature of the algorithm that is, I believe, most open to criticism.

1

The scoring system just described is a purely numerical procedure. All three types of question contribute the same value of 5 to the selection of a solution, but essential, usual and sometimes questions subtract 99, 5 and 1 respectively from the score. There is no principle by means of which the values of these numbers can be substantiated. Why do all three question types have the same positive value? Why does a usual question have exactly five times the negating power of a sometimes question? Does not the Rosch thesis indicate that some usual questions are more closely related than others to a category? The only answer which can be returned to such objections is that the chosen values seem to make the system work. But a resort to unstructured empiricism at this point in the argument is lame. I conclude that, for the lack of a justifying principle, the scoring system of the Frey algorithm is unsound. A choice from amongst the classifiers ought to be made on logical rather than arbitrarily arithmetical grounds.

2

A more fundamental, if rather less sharply focussed, objection can be made to the notion of A, B and C questions. Frey sees the classifiers of his system as Roschian prototypes, and the three types of questions are intended to define them as such. His algorithm is, in fact, structured in such a way as to model reality according to Rosch's ideas. But it is, I think, a mistake to confuse a representation with a model of something. The relation of a model to reality is literal whereas a representation stands as a symbol in the place of some aspect of reality. For example,

3

the area of a large circle can be modelled by a small disk, but the formula πr^2 is a representation of it. Representation is more abstract than modelling, and there is no one-to-one relationship between the sign and what is signified in a representation.

If the purpose of an expert system is to mimic the performance of a human expert, then I think that it should be framed in such a way as to represent his knowledge rather than to model his methods. This means that an algorithm which obtains a seemingly expert answer to a problem quickly is better than a slower system which works in the same way as does the human expert. In the light of this distinction, I have designed the Cortex expert system shell to identify a solution quickly and accurately by representing it usefully, rather than creating a system to model the domain faithfully. The Cortex algorithm differs substantially from Frey's algorithm in ways that I shall explain in the next chapter.

When a particular classifier in the Frey algorithm has been selected as a possible solution, all the remaining questions in its masks must be asked and answered before another candidate classifier can be examined. This requirement follows from the necessity to arrive at an arithmetical score for the solution.

For example, if question 3 in the essential mask of classifier 27, Richardsonian Romanesque, is answered correctly with 'yes' then question 11 must be asked and answered. If this also receives a correct answer the first question in the usual mask, question 71 "rounded arches above windows?" must be asked. The procedure is repeated until the last question in the occasional mask, question 151 "a pinnacle on the roof?" is reached. If all questions receive a correct answer the classifier achieves its maximum possible score of 55. The effect of this way of sequencing the

presentation of questions is that the user of the system is required to answer a large number of questions including many which are irrelevant.

I think it is less burdensome to the user, as well as computationally more efficient, to ask him to supply answers only to those questions that differentiate one classifier from all other possible classifiers. This is the procedure that I have adopted in the design of the method of control in the Cortex shell.

Chapter 12. THE PLAN OF CORTEX

The design of Cortex is inspired by the work of Peter Frey. The notion of regarding an expert system as a classification system which can be represented by means of strings of bits is his invention, and it is the starting point for the development of Cortex. The most original programming routine in the Frey algorithm is the bit-matching procedure which has been described in Figure 11.4. I have made use of this routine in Cortex, where it appears in the procedure CalculateProbability. In addition, a modification of the routine is made use of in the procedure RemoveContradicted-Solutions.

However, Cortex differs from Frey's implementation in almost all other respects. This is for three reasons. Firstly, I believe that I have been able to design a better way of controlling the system, and that the method of calculating probability in Cortex is an improvement upon Frey's procedure. Secondly, Cortex is written in Pascal rather than Microsoft BASIC, and I have therefore been able to make use of much more advanced programming procedures than were available to Frey. A feature that has been of particular importance in the development of Cortex is the bit manipulation functions that are available in the Prospero Pascal compiler. The third difference follows from the fact that Cortex is written as a true shell whereas House.Bas is specific to its knowledge domain. File handling procedures are available in BASIC, but the knowledge base in House.Bas is encoded in the form of DATA statements. Files are of general applicability, but a DATA statement is specific and it must be embedded in a particular place in a BASIC program.

Control in Cortex

The scoring system of the Frey algorithm is intended to build up progressively to the identification of a solution as the correct one. Each answer supplied by the user makes

its contribution to the plausibility of a classifier, and in this way the environment of the system serves to provide evidence for the choice of one of the possible domain solutions as the best. Thus Frey's method is based, tacitly, upon an inductive procedure. But, as I have argued, there is a price to be paid for adopting this approach to the problem. In the Frey algorithm everything depends upon the allocation of scores, but no consistent explanation for the structure of the scores is available. This gives an arbitrary character to the results that can be obtained from the implementation of the algorithm in his House.Bas. I think that a better and more rigorous algorithm can be created by looking at the problem from a Popperian, and non-inductive, point of view.

Popper (1934) argues, that the scientific acceptability of a theory is dependent both upon its falsifiabilty and upon its resistance to empirical falsification. Hypotheses may be freely conjectured, but their acceptability as scientific theories is reserved for those hypotheses which can be but are not falsified. In a similar way, every solution contained in the implementation of an expert system is capable of being falsified by an answer supplied by the user to a question. If it is not so falsifiable then it has no role to play in the system.

An expert system which functions by rejecting falsified solutions will finish either with one or more unfalsified solutions, or with a confession that it can find no solution within the domain environment. This is the non-inductive control principle upon which the Cortex expert system shell is built. Since a solution in Cortex, like a Popperian theory, remains possible until it has been falsified no system of scoring is called for. The arithmetically arbitrary scoring mechanism of the Frey algorithm is not reproduced in Cortex.

The notion of a falsifiable solution points the way to a novel method of controlling an expert system. The problem of control is solved in Frey algorithm in the conventional way, which is by means of an ordered list. The next question to be sent to the screen is the next one in the list of questions relating to a favoured solution. The answer received from the user either causes the solution to be rejected, by giving it a score of -99 if the question is an essential one and the answer does not match or, if it is a correctly answered usual or occasional question, it contributes a score of 5 towards reaching its threshold value.

However, this procedure overlooks an important feature of the questions which are stored in the knowledge base. This is the fact that the greater the number of solutions in whose classifiers a particular question appears, the greater is the power of that question to falsify a solution. A question whose answer is called for by only a single solution string can contradict only that solution while a question appearing in 50 solution strings may, when answered, contradict them all. It is for this reason that Cortex solves the problem of which question to ask next by selecting that question which appears most frequently in the classifiers of those solutions which have not been falsified. The operation is performed by the procedure FindMostFrequentQuestion whose draft code is given in the next chapter. Controlling the system in this way has the effect of eliminating solutions as rapidly as possible and so concentrating the search upon the small number of possible solutions that remain. Conventional control methods, in which a fixed list must be worked through, zero-in upon the solution slowly by a process of sequential elimination rather than quickly by excluding early on all those solutions which cannot be correct.

But it is no more sufficient to select the next question on the basis of its frequency alone than it is to ask the next question that appears in a prepared list. In either case

the result will be that the user will have to find answers to many questions that are illogical or redundant in the light of the answers already given. In the domain illus-
1 trated in Figure 11.2, for example, there is no point in asking whether the animal has feathers if the system has already been told that it has hair. Question 9 is therefore redundant once question 1 has been answered affirmatively. The Cortex algorithm makes use of a question's power of discrimination to solve this aspect of the problem of control.

If, in Figure 11.2, the possible solutions have been re-
duced to numbers 5, 6 and 7 then the animal in question must be a bird. If so, there is no point in asking either question 9, has it feathers, nor question 11, does it lay
2 eggs, since the required answers to both questions are the same in all three solutions. That is the same thing as saying that these two questions have no power to discrimi-
nate between the remaining solutions. Those questions which are devoid of discriminatory power are identifiable solely by virtue of the formal feature that their required answers are the same in all the remaining possible solutions.

Cortex makes use of this fact, and excludes from the ques-
tions to be presented to the user any question for which all remaining required answers are the same. The algorithm according to which I have structured Cortex therefore works in two stages. First the list of solutions is pruned by
3 obtaining an answer to the most frequently occurring ques-
tion. This is in effect a search for features common to many of the possible solutions in the domain. Then those questions which cannot discriminate between the remaining solutions are excluded from the list of questions to be asked. This as a process of zeroing-in upon a shortened list of still-possible solutions. The cycle is then repeat-
ed. The code for both stages in the control of Cortex are in the procedure FindMostFrequentQuestion.

In this way the classification algorithm opens the way to a new method of control in expert systems. It is based neither upon ordered lists nor metaknowledge, but upon the falsifiabilty of solutions and questions. I claim two advantages for this algorithm. Firstly, it closely mimics the method of a human expert, who will begin by surveying the scope of the problem before concentrating his attention upon the most promising of the remaining solutions. Cortex therefore appears to the user to be acting in a natural way. Secondly, the algorithm is very flexible because its way of working is purely formal. The method requires no knowledge of the world, and the algorithm may therefore be implemented in any domain whatsoever. This flexibility follows from the abandonment of Frey's inductionism in favour of a procedure analogous to Popper's method of falsification.

Probability

Every useful expert system must have a way of assessing the degree of reliance that can be placed upon its solution to a problem. In conventional rule-based systems this is usually done by employing Baysian or Dempster-Shafian methods of calculating a probability. Frey was understandably reluctant to add arithmetical complications to his program if it could be avoided. In his algorithm the solution that first reaches its threshold value is assumed to be the correct one. Probability in House.Bas is implied rather than calculated.

> "If one classifier accumulates a score that exceeds a predetermined threshold, this house type is declared the winner and no further information is needed. The notion is that the weight of the evidence for this hypothesis is so strong that the decision is obvious." (Frey, 1986a)

However, it remains that the threshold value upon which the emergence of the winning candidate is based in the Frey algorithm is arrived at subjectively. For instance, the

threshold value of 30 which is assigned to Richardsonian Romanesque in the example quoted in the last chapter, could have been 25 or 40 without any loss of consistency or logic. As in the case of the problem of control, I have found that by abandoning Frey's scoring system an alternative and better method of assessing probability can be discovered.

In both the Frey algorithm and in Cortex, an essential question which fails to receive a correct answer has the effect of eliminating a solution from the list of possible candidates. Frey makes use of answers to usual and occasional questions to advance or retard the progress of the remaining candidates towards his system's winning post.

However, if the attempt to model reality along the lines of Rosch's categories is rejected in favour of the view of an expert system as a representation of reality, then the distinction between usual and occasional questions disappears. One attribute which is usually present and another which is sometimes present in a particular situation are alike in that neither need be present. Accordingly, Cortex contains only essential and usual questions in the knowledge base. Essential questions, as in the Frey algorithm, must receive matching answers, while usual questions may or may not.

If the answer to a usual question can corroborate the likelihood of a solution being correct, then it follows that the proportion of the usual questions relating to a particular solution that receive matching answers is a measure of the probability of the solution being correct. If, for example, a solution contains 50 usual questions in its question set and 35 of them receive matching answers, then the probability of the solution being correct is 70%. This is the definition of probability that is employed in Cortex. It is implemented in a straightforward piece of code in the procedure CalculateProbability.

The Coding of Cortex

The house style identification program House.Bas, with which Frey (1986b) demonstrates his algorithm, is written in Microsoft BASIC. The usual penalties that must be paid by those who program in BASIC, which are that the code is very difficult to read and the program runs slowly, are evident in House.Bas. In addition, two particular shortcomings of BASIC have forced Frey's program into an awkward shape. These are the absence of bit manipulation functions and the impossibility of compiling separate segments.

In a BASIC program all variables are common variables. Consequently, changes in the value of a variable can be made by any piece of code that invokes its name. In a complicated program it is very difficult to prevent such changes from occurring unintentionally, a phenomenon known as "programming by side-effects" (Cooper & Clancy, 1985). Bugs of this type can be very hard to find. In Cortex, data types which are used by more than one procedure are declared as common, but every variable is declared locally. Values are passed to other procedures when required as parameters. The structure of Cortex therefore makes it possible to identify the procedure in which any variable receives a value, and to trace the source of the value of any variable.

Cortex consists of 51 separately compiled segments which contain a total of some 1600 lines of Pascal code. With this structure a segment may be edited in the knowledge that the changes in the performance of the program are attributable to changes made to that segment and to no other. When combined with the local declaration of variables, a program in which all procedures and functions are contained in separately compiled segments is more robust, more easily read and is simpler to maintain than a typical BASIC program.

A program structure that combines the local declaration of variables with separate compilation of segments can hide information on a "need to know" (Sommerville, 1985) basis. In Cortex, for example, the list of possible solutions can only be pruned by RemoveContradictedSolutions, and the pruning can only take place as a result of the answer to the question passed to it as the most frequent question by the procedure FindResult. If variables were common and a side effect were to occur whereby the answer to a different question was selected, then the program would produce a wrong answer. These programming methods, which were not available to Frey when writing House.Bas, form the basis of the structure of Cortex.

Frey has good reason to bemoan a further difficulty with which he must contend.

> "Some mainframes have machine level instructions that count the number of bits that are set in a word. Microprocessors do not have this instruction, so the only way to do a speedy bit count is to examine the word in 8-bit sections and to use a table with 256 entries to look up the proper bit count." (Frey, 1986a)

This limitation forces Frey to resort to an extensive use of arrays. For example, in House.Bas the array MK is dimensioned to 16, the word length in Microsoft BASIC, and the elements are then filled with the 16 integer values found in the first four lines of DATA. Each value in turn sets the next bit in the word with the result that the elements of the array contain one set bit for all possible bit positions in the word. The elements of MK are then used to identify the bit corresponding to a particular classifier in line L2, or a question in the subroutine Query. In this way House.Bas places a heavy load on the processor, and the Frey algorithm cannot be expected to run fast, even when written in a compiled language, if the implementation calls for more than a few questions and solutions.

Although it is true, as Frey observes, that desktop computers do not have built-in facilities for counting bits it is also true that bit manipulation can be carried out by software as well as hardware means. BASIC, which is a kind of computer baby talk, lacks such facilities. However the Prospero Pascal compiler, with which Cortex has been written, contains four functions which enable an individual bit to be addressed and manipulated. The bits which represent a simple variable, or an array or a field of a record for example, can be tested, set, cleared or flipped using the Prospero functions. Cortex makes use of the Prospero functions to manipulate the message, classifier and mask strings directly, and in such a way as to make the contorted manoeuvres of House.Bas unnecessary. For example, the job of the array MK subroutine Query in House.Bas is performed by only two lines of code containing the testbit function in the procedure FindMostFrequentQuestion in Cortex. It may be noted that Turbo Pascal, its popularity notwithstanding, does not contain bit manipulation functions.

Cortex in Pseudocode

```
Display sign-on message
   domain specific information
CASE 1. user is knowledge engineer
        call knowledge base menu
     2. user wants to interrogate system
        call FindResult
1. KnowledgeBaseMenu
   CASE set up questions
            write, edit, delete, display or print question
            texts
      ditto question explanation texts
   set up solutions
      write, edit, delete, display or print solution
      texts
      ditto solution explanation texts
   set up classifiers
```

```
                display solution texts
                display question texts successively
                IF question is relevant
                   set bit
                IF classifier is incorrect
                   edit classifier
                   delete classifier
2. FindResult
   set up linked list of solutions in memory
   WHILE some essential questions remain unanswered
      find most frequently occurring question
         obtain answer
      make any solution contradicted by the answer inactive
      pass over question if all remaining answers are
      identical
   obtain answers to usual questions
   calculate probability
   display most likely solution
     IF probability is zero search is a failure
```

This program structure is represented in the following diagram.

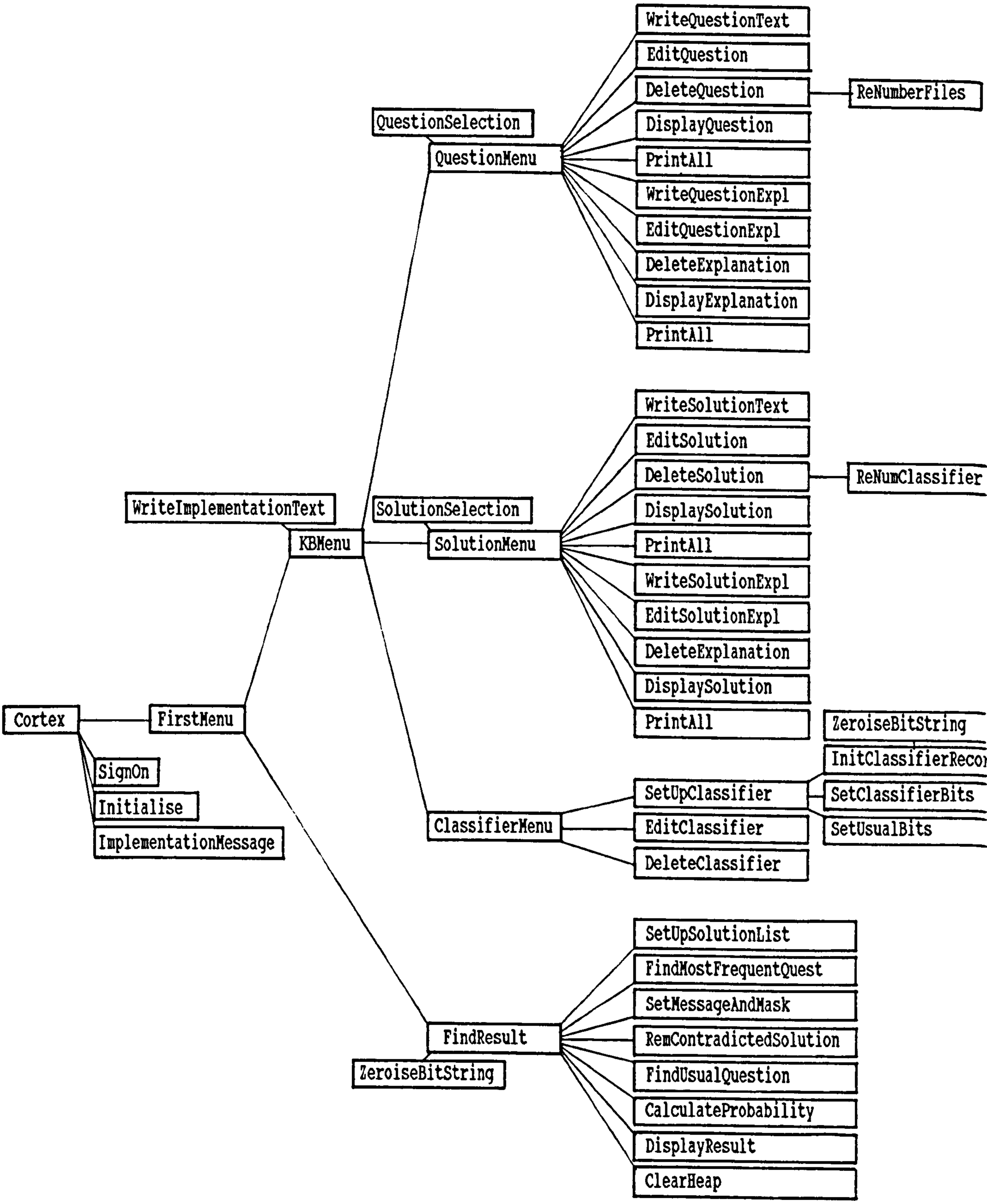

Figure 12.1
Flow of Control in Cortex

Chapter 13. THE CORTEX SHELL

The Segments of Cortex

The following segments, listed by their procedure names, will be required by the system.

1. Cortex - main program
   (i) initialise
   (ii) sign-on
   (iii) implementation
   (iv) FirstMenu
2. Sign-on - display on screen
   (i) welcome message
   (ii) implementation message
3. Initialise - some preliminaries
   (i) count questions
   (ii) count solutions
4. ImplementationMessage - message describing the domain
   (i) display information about the implementation
   (ii) display number of questions & solutions on file
5. FirstMenu - choose between
   (i) use Cortex, by the user or
   (ii) work on knowledge base, by knowledge engineer
6. KnowledgeBaseMenu - knowledge engineer's menu to
   (i) work on the questions
   (ii) work on the solutions
   (iii) write implementation message
   (iv) set up classifiers and masks
7. WriteImplementationText - message for the user
   (i) write the text
   (ii) store on disk
8. FormFileName - set up name in form needed to access disk files
   (i) obtain keynumber, title and DOS directory of file
   (ii) concatenate as a string
9. PushPen - manage the writing of text
   (i) call WRITER
   (ii) write text
   (iii) store text on file

10. DisplayTextFile - on screen
    (i) obtain file
    (ii) display text on screen
11. ZeroiseBitString - set all elements of a bit string to zero
    (i) pass sting title and field to procedure
    (ii) loop through all elements setting each to 0
12. QuestionMenu - choose work to be done on question files
13. QuestionSelection - manage procedure calls from QuestionMenu
14. WriteQuestionText - write and store the text of a question
15. WriteQuestionExplanation - write and store an explanation
    (i) check for question on file
    (ii) check for explanation on file
16. EditQuestion - alter the text of a question
17. EditExplanation - alter the text of an explanation
18. DeleteQuestionFile - remove a question text from file
    (i) delete question diskfile
    (ii) delete corresponding explanation diskfile
    (iii) recalculate CountOfQuestions
    (iv) renumber classifier
    (v) renumber essential masks
    (vi) renumber usual masks
19. ReNumberFiles - after a deletion
    (i) renumber subsequent files
    (ii) re-set bit strings
20. DeleteExplanation - delete the text the explanation of a question or solution
21. DisplayQuestion - display the text of a question on the screen
22. DisplayTextFile - obtain from disk and display on the screen
    (i) question text files
    (ii) solution text files
    (iii) explanation text files
23. DisplayExplanation - display the text of an explanation
24. PrintAllFiles - print questions, solutions or explanations
25. SolutionsMenu - choose work to be done on solution texts

26. SolutionSelection - manage procedure calls from SolutionMenu
27. WriteSolutionText - write and store the text of a solution
28. WriteSolutionExplanation - write and store an explanation
    (i) check for question on file
    (ii) check for explanation on file
29. EditSolution - alter the text of a solution
30. DeleteSolutionFile
    (i) delete solution diskfile
    (ii) delete solution explanation diskfile
    (iii) delete classifier and masks
31. ReNumberClassifier - renumber classifiers following a deletion
32. DisplaySolution - display the text of a solution on the screen
33. ClassifierMenu - choose operations on the classifier file
34. SetUpClassifier - set up the classifiers one by one
    (i) set fields of classifier file elements to 0
    (ii) display text of the solution
    (iii) display texts of all the questions
35. InitialiseClassifierBits
36. SetClassifierBits - operate on the bit strings
    (i) ask if the question is essential or usual, or is irrelevant
    (ii) ask if the answer must be T or F
    (iii) record answers on disk file with elements containing the fields
        (a) classifier
        (b) essential mask
        (c) usual mask
37. SetUsualBits - operate on the bit strings
    (i) set the bits for usual questions
    (ii) return to SetClassifierBits
38. EditClassifier - alter the bit settings of a classifier
39. DeleteClassifier - remove a classifier from the file
40. FindResult - main procedure for forming and using the message

41. SetUpSolutionsList - set up a linked list of solutions
    (i) fields to include essential and usual masks as arrays of integers, usual questions that have been answered, usual questions that have been answered correctly
42. FindMostFrequentQuestion - most frequently occurring question
    (i) search all the essential masks in the knowledge base
    (ii) count the occurrence of each question
    (iii) find the most frequently occurring question
43. MessageAndMasks - form the message
    (i) call MostFrequentQuestion
    (ii) display the most frequent question
    (iii) record the user's answer in the message
    (iv) record the user's answer in the message mask
    (v) repeat
44. RemoveContradictedSolution - search essential masks and;
    (i) compare each essential mask with the message bit
    (ii) delete from the list any solution whose essential mask is contradicted by a bit in the message string
    (iii) find the question which appears most often in the remaining essential masks by calling MostFrequent-Question
45. FindUsual Question - find any unanswered questions in the possible solutions
    (i) search the list of possible solutions
    (ii) if any usual questions are unanswered, return the number of the question
46. CalculateProbability - find the probability of the possible solutions
    (i) find out usual result
    (ii) count the number of matching answers for each usual mask
    (iii) calculate probability for each solution, using the count of usual questions
47. DisplayResult - screen display of;
    (i) most probable solution with probability or
    (ii) failure message
48. ClearHeap - remove the remaining solution list elements

The Segments Individually.

13.1 Cortex

As the diagram in Figure 12.1 shows, the main program of CORTEX serves only to start the system. It controls the presentation of the preliminary displays and the initialisation of the program.

13.1.1 Pseudocode.

```
display sign-on message
call implementation procedure
pass number of questions and solutions on file to first menu
provide for exiting the program
```

13.1.2 Draft Source Code.

```
PROGRAM  Cortex;
         { Main program. }
PROCEDURE SignOn (VAR CountOfQuestions, CountOfSolutions: integer); EXTERNAL;
PROCEDURE ImplementationMessage (CountOfQuestions, CountOfSolutions: integer; EXTERNAL;
PROCEDURE FirstMenu (CountOfQuestions, CountOfSolutions: integer); EXTERNAL;
VAR  CountOfQuestions, CountOfSolutions: integer;
BEGIN
SignOn (CountOfQuestions, CountOfSolutions);
ImplementationMessage (CountOfQuestions, CountOfSolution);
FirstMenu (CountOfQuestions, CountOfSolutions);
END.
```

13.2 SignOn

Display a screen welcoming the user to the system.

13.2.1 Pseudocode.

```
welcome the user to CORTEX
   use colour screen and draw a border
   carry out initialisation while welcome display is on screen
```

13.2.2 Draft Source Code.

```
SEGMENT  SignOn;
         { Sign on display. }
insert PASPC
insert PASDOS
PROCEDURE Blankln (number: integer); EXTERNAL;
```

```
PROCEDURE Initialise (VAR CountOfQuestions, CountOfSolutions: integer); EXTERNAL;
PROCEDURE SignOn (VAR CountOfQuestions, CountOfSolutions: integer);
VAR  Greeting: text
        Gate: char;
BEGIN
InitScreen;
Paper (7);
Ink (1);
TextFrame (true);
ScreenFile (Greeting);
CursorOff;
GoToXY (31,6);
writeln (Greeting,'******************');
GoToXY (31,7);
writeln (Greeting,'Welcome to Cortex');
GoToXY (31,8);
writeln (Greeting,'******************');
writeln;
Ink (9);
GoToXY (23,10);
writeln (Greeting,'The thinking man's expert system');
GoToXY (7,22)
writeln (Greeting,'Please wait while the solutions and questions on file are counted.');
Initialise (CountOfQuestions, CountOfSolutions);
GoToXY (7,22);
PutChattr (' ',7,9,66);
GoToXY (23,22);
writeln (Greeting,'Press any key to continue.');
Gate:= ConSilent;
CursorOn;
InitScreen
END;
BEGIN  END.
```

13.3 Initialise.

The number of questions and the number of solutions that are held on file are parameters that are used in several places in

Cortex. The number of files that exist when the program is entered is calculated before the first menu is sent to the screen.

13.3.1 Pseudocode.

```
count number of questions on file
    form question name
    WHILE fstat (question name) = true
        increment CountOfQuestions
count number of solutions on file
    form solution name
    WHILE fstat (solution name) = true
        increment CountOfSolutions
```

13.3.2 Draft Source Code.

```
SEGMENT  Initial;
        {Count questions and solutions on file. }
insert common types
insert PASPC
insert PASDOS
PROCEDURE  FormFileName (filenumber:integer; title: string; VAR FileName: string); EXTERNAL;
PROCEDURE  ZeroiseBitString (VAR Bits: bitstring); EXTERNAL;
PROCEDURE  Initialise (VAR CountOfQuestions, CountOfSolutions: integer);
VAR   FileName: string;
      QuestionFile, SolutionFile: string;
BEGIN
MkDir ('\shell\question');
ChDir ('\shell\question');                          { Calculate CountOfQuestions }
CountOfQuestions:= 0;
FindFile ('quest1',QuestionOnFile);                 { Is there a quest1? }
WHILE QuestionOnFile <> '' DO BEGIN                 { If so, count through the questions }
   CountOfQuestions:= CountOfQuestions + 1;
   FormFileName (CountOfQuestions + 1,'question',FileName);          { Is the next question on file? }
   FindFile (FileName, QuestionOnFile);  { Returns QuestionOnFile as empty when no question is found }
END;  { of WHILE }
MkDir ('\shell\solution');
ChDir ('\shell\solution');                          { Calculate CountOfSolutions }
CountOfSolutions:= 0;
FindFile ('solut1',SolutionOnFile);                 { Is there a solut1? }
WHILE SolutionOnFile <> '' DO BEGIN                 { If so, count through the solutions }
```

```
   CountOfSolutions:= CountOfSolutions + 1;
   FormFileName (CountOfSolutions,'solution',FileName);              { Is the next solution on file? }
   FindFile (FileName,SolutionOnFile);   { Returns SolutionOnFile as empty when no solution is found }
END;  { of WHILE }
ChDir ('\shell');
END;
BEGIN  END.
```

13.4 ImplementationMessage.

When Cortex has been implemented the user must be provided with information about the domain in which the implementation has been made. For example, information will be required about the type of domain and the scope of the implementation, and help may be needed as to the best way to formulate answers to the questions. This segment displays the required information by writing the implementation file to the screen.

13.4.1 PseudoCode.

```
display screen heading
write information to screen
```

13.4.2 Draft Source Code.

```
SEGMENT  Impment;
        { Display information about the domain. }
insert PASPC
PROCEDURE Blankln, PressKey; EXTERNAL;
PROCEDURE ImplementationMessage;
VAR   Disk: text;
      Line: string[100];
      Counter: integer;
BEGIN
ClrScr;
IF fstat ('Implment\message') = true THEN BEGIN
   writeln (' ':21,'Implementation Information');
   writeln (' ':21,'**************************');
   assign (Disk, '\shell\implemnt\message');
   reset (Disk);
   Counter:= 8;
   WHILE NOT eof(Disk) DO BEGIN
```

```
      readln (Disk, Line);                    { Copy a line from Disk to Line }
      GoToXY (10,Counter);                    { Move cursor to starting point of text }
      writeln (Line);                         { Send contents of Line to screen }
      Counter:= Counter + 1                   { Move to next screen line }
   END;  { of WHILE }
close (Disk, true);
writeln ('There are ',CountOfQuestion,' questions and ',CountOfSolutions,' solutions on file.');
END  { of IF }
ELSE
writeln (' ':7,'No implementation file has been written.');
END;
BEGIN  END.
```

13.5 First Menu.

Cortex, like any expert system, will be worked upon by a knowledge engineer or worked with by a user. This segment is the menu at which a decision is made as to the mode of operation of the system.

13.5.1 Pseudocode.

```
display list of choices
    use Cortex?
    work on knowledge base?
make selection with a CASE statement
```

13.5.2 Draft Source Code.

```
SEGMENT  FrstMenu;
        { Chooses between work on the knowledge base or use of the implemented system. }
insert PASPC
PROCEDURE Blankln (number: integer); EXTERNAL;
PROCEDURE MenuError (range: integer); EXTERNAL;
PROCEDURE KnowledgeBaseMenu (CountOfQuestions: integer); EXTERNAL;
PROCEDURE FindResult (CountOfQuestions, CountOfSolutions: integer); EXTERNAL;
PROCEDURE FirstMenu (CountOfQuestions, CountOfSolutions: integer);
VAR  Flag: boolean;
     Selector: integer;
BEGIN
Flag:= true;
WHILE Flag = true DO BEGIN
writeln ('Do you want to;');
```

```
writeln ('1. Use Cortex?');
writeln ('2. Work on the knowledge base?');
writeln ('3. Exit from Cortex?');
writeln ('Make your choice by typing a key number. Then press RETURN.');
read Selector;
CASE Selector OF
   1: FindResult (CountOfQuestions, CountOfSolutions);
   2: KnowledgeBaseMenu (CountOfQuestions, CountOfSolutions);
   3. Flag:= false;
   OTHERWISE BEGIN
      ClrScr;
      Blankln (8);
      MenuError (3)
   END;  { of OTHERWISE }
END;  { of CASE }
END;  { of WHILE }
END;
BEGIN  END.
```

13.6 KnowledgeBaseMenu.

The work of the knowledge engineer can be divided into three main tasks. These are the creation of the questions, the creation of the solutions and the writing of the classifier and its masks. A secondary task is the writing of the implementation message. This segment consists of the menu that chooses between these alternatives.

13.6.1 Pseudocode.

```
display list of choices
     work on questions
     work on solutions
     write a classifier
     write an implementation message
make selection with a CASE statement
```

13.6.2 Draft Source Code.

```
SEGMENT KBMenu;
       { Knowledge engineering main menu. }
insert PASPC
PROCEDURE Blankln (number: integer); EXTERNAL;
```

```
PROCEDURE MenuError (range: integer); EXTERNAL;
PROCEDURE QuestionMenu; EXTERNAL;
PROCEDURE SolutionMenu; EXTERNAL;
PROCEDURE ClassifierMenu; EXTERNAL;
PROCEDURE WriteImplementationText; EXTERNAL;
PROCEDURE KnowledgeBaseMenu (CountOfQuestions, CountOfSolutions: integer);
VAR  Flag: boolean;
     Selector: integer;
BEGIN
Flag:= true;
WHILE Flag = true DO BEGIN
   writeln ('Knowledge Base Menu.');
   writeln ('Do you want to;');
   writeln ('1. Write, edit, delete, display or print the text of a question?');
   writeln ('2. Write, edit, delete, display or print the text of a solution?');
   writeln ('3. Write, edit or delete a classifier?');
   writeln ('4. Write the text of the implementation screen?');
   writeln ('5. Return to the main Cortex menu?');
   writeln ('The Cortex shell can accept up to a maximum of ',MaxNumberOfQuestions:3,' questions.');
   writeln ('There is effectively no limit upon the number of solutions that can be accommodated.');
writeln ('Make your choice by typing a key number. Then press RETURN.');
read Selector;
CASE Selector OF
   1: QuestionMenu (CountOfQuestions);
   2: SolutionMenu (CountOfSolutions);
   3: ClassifierMenu (CountOfQuestions);
   4: WriteImplementationText;
   5: Flag:= false;
   OTHERWISE BEGIN
      ClrScr;
      BlankLn (8);
         MenuError (5)
      END;  { of OTHERWISE }
   END;  { of CASE }
END;  { of WHILE }
END;
BEGIN  END.
```

13.7 WriteImplementationText.

This segment writes and stores the text of the message that is sent to the screen by Implementation.

13.7.1 Pseudocode.

create text using PushPen
store text as a disk file

13.7.2 Draft Source Code.

```
SEGMENT WritImpl;
        { Create text of implementation message. }
PROCEDURE PushPen (directory, FileName: string); EXTERNAL;
PROCEDURE WriteImplementationText;
BEGIN
writeln ('Please write the text for the implementation message.');
PushPen (Implemnt','Message');
END;
BEGIN END.
```

13.8 FormFileName.

The disk files that are used by CORTEX are stored in sub-directories named 'question' and 'solution'. In each directory the file may be a question text, a solution text, or an explanation of either. There are, consequently, three titles under which a text file may need to be accessed. This segment forms these titles as strings.

21.8.1 Pseudocode.

for questions, solutions and explanations
   obtain keynumber and title as value parameters
   obtain FileName as variable parameter
   convert keynumber to a sting
   insert keynumber onto end of title

21.8.2 Draft Source Code.

```
SEGMENT  FormFile;
         { Set up name of file to be accessed on disk. }
PROCEDURE  FormFileName (Index: integer; title: string; VAR FileName: string);
VAR   Key: string;
BEGIN
str (Index,Key);                                  { Form the string Key from the integer Index }
IF title = 'question' THEN BEGIN
```

```
   FileName:= 'quest';                          { Write the stem of a questions filename }
   insert (Key,FileName,6);                     { Append the key number to complete the filename }
END  { of IF }
ELSE IF title = 'solution' THEN BEGIN
       FileName:= 'solut';                      { Write the stem of a solution filename }
       insert (Key,FileName,6);                 { Append the key number to complete the filename }
     END  { of ELSE IF }
     ELSE IF title = 'explanation' THEN BEGIN
            FileName:='explan';                 { Write the stem of an explanation filename }
            Insert (Key,FileName,7);            { Append the key number to complete the filename }
          END; { of ELSE IF }
END;
BEGIN  END.
```

13.9 PushPen.

The text editor WRITER accepts keyboard input and organises it into a textfile on disk. This segment calls WRITER, and then takes the formed text, gives it a name, and stores it in a specified sub-directory.

13.9.1 Pseudocode.

```
call WRITER
set up directory
form file name
read text from WRITER into new named file
```

13.9.2 Draft Source Code.

```
SEGMENT  PushPen;
        { Compose text and store it in specified sub-directory. }
PROCEDURE Writer; EXTERNAL;
PROCEDURE PushPen (directory, FileName: string);
VAR   Disk, DiskText; text;
      Line: string[100];
BEGIN
Writer;                                         { Writer stores text as DiskText on disk file TempFile }
assign (DiskText, 'TempFile');                  { TempFile is on \SHELL, not on \FORMAT }
reset (DiskText);                               { Open DiskText for input }
ChDir (directory);
assign (Disk, FileName);
rewrite (Disk);                                 { Open Disk for output }
```

```
   WHILE NOT eof(DiskText) DO BEGIN
     readln (DiskText, Line);                    { Read a line of DiskText into Line }
     writeln (Disk,Line)                         { Write Line to Disk }
   END;  { of WHILE }
close (Disk, true);
erase (DiskText)
END;
BEGIN  END.
```

13.10 DisplayTextFile.

Files stored on disk must be capable of being extracted and displayed. This segment accesses and displays such files.

13.10.1 Pseudocode.

```
connect disk file to temporary local file
change to disk file sub-directory
display heading
extract disk file
display at specified position on screen
return to SHELL sub-directory
```

13.10.2 Draft Source Code.

```
SEGMENT  DispFile;
        { Display text file on screen. }
insert PASPC
insert PASDOS
PROCEDURE DisplayTextFile (DiskFile, directory, heading: string; Index, displayline: integer);
VAR   TempFile: text;
      Line: string[100];
BEGIN
assign (Disk,FileName);                         { Connect disk file to temporary local file }
ChDir (directory);                              { Change to directory containing the file }
reset (TempFile);                               { Open Disk for input }
GoToXY (1,displayline);
writeln (heading,'no ',Index:3,'.');            { Display heading }
   WHILE NOT eof(TempFile) DO BEGIN
     readln (TempFile, Line);                   { Read a line of Disk into Line }
     GoToXY (24,displayline);                   { Position cursor }
     writeln (Line);                            { Display Line }
     displayline:= displayline + 1              { Move cursor down one line }
```

```
   END;  { of WHILE }
close (Disk,true);
ChDir ('\SHELL');                              { Return to SHELL sub-directory }
END;
BEGIN  END.
```

13.11 ZeroiseBitString.

The classification of questions and answers are recorded in CORTEX in the form of bitstrings, which take the form of arrays of integers. The program functions by setting every bit in a string to zero initially, and then setting individual bits to 1 as required by the classification scheme. This segment carries out the initialisation.

13.11.1 Pseudocode.

pass name of bitstring and relevant field as a VAR parameter
FOR loop through all the questions
set each element to 0

13.11.2 Draft code.

```
SEGMENT  ZeroBit;
        { Initialise bit strings. }
insert common types
PROCEDURE ZeroiseBitString (VAR Bits: bitstring);
VAR       Index: integer;
BEGIN
FOR Index:= 0 TO MaxNumberOfIntegers DO
   Bits[Index]:= 0;
END;
BEGIN  END.
```

13.12 QuestionMenu.

When the knowledge engineer chooses to work on the questions there are 10 operations that may need to be carried out. This procedure makes the selection between them.

13.12.1 Pseudocode.

display choice of operations
use a CASE statement to call the relevant procedure

13.12.2 Draft Source Code.

```
SEGMENT  Question;
        { Selects the operations to be performed on the questions file. }
insert PASPC
PROCEDURE QuestionSelection (Selector: integer; VAR CountOfQuestion: integer; VAR Flag: boolean); EXTER-
NAL;
PROCEDURE QuestionMenu (VAR CountOfQuestions: integer);
VAR  Flag: boolean;
     Selector: integer;
BEGIN
ClrScr;
Flag:= true;
WHILE Flag = true DO BEGIN
   writeln ('Questions Menu');
   writeln ('**************');
   writeln ('Do you want to;');
   writeln ('1.  Write the text of a question?');
   writeln ('2.  Write the explanation of a question?');
   writeln ('3.  Edit the text of a question?');
   writeln ('4.  Edit the explanation of a question?');
   writeln ('5.  Delete a question from the questions file?');
   writeln ('6.  Delete the explanation of a question from the file?');
   writeln ('7.  Display the text of a question?');
   writeln ('8.  Display the explanation of a question?');
   writeln ('9.  Print the text of a question?');
   writeln ('10. Print the explanation of a question?');
   writeln ('11. Return to the Knowledge Base Menu?');
   writeln ('Make your choice by entering a key number.');
   writeln ('Then press RETURN.');
   read (Selector);
   QuestionSelection (Selector, CountOfQuestions, Flag);
END;  { of WHILE }
END;
BEGIN  END.
```

13.13 QuestionSelection.

Carries out the calling of procedures from the question menu.

13.13.1 Pseudocode.

CASE

procedure calls

OTHERWISE

menuerror

13.13.2 Draft Source Code.

```
SEGMENT  QstSelec;

        { Manage the calling of procedures by the question and solution menus. }

insert PASPC

PROCEDURE WriteQuestionText (VAR CountOfQuestions: integer); EXTERNAL;

PROCEDURE WriteQuestionExplanation; EXTERNAL;

PROCEDURE EditQuestion; EXTERNAL;

PROCEDURE EditQuestionExplanation; EXTERNAL;

PROCEDURE DeleteQuestion (VAR CountOfQuestions: integer); EXTERNAL;

PROCEDURE DeleteExplanation (title: string); EXTERNAL;

PROCEDURE DisplayQuestion; EXTERNAL;

PROCEDURE DisplayExplanation (title: string); EXTERNAL;

PROCEDURE PrintAll; EXTERNAL;

PROCEDURE Blankln (number: integer); EXTERNAL;

PROCEDURE MenuError (range: integer); EXTERNAL;

PROCEDURE QuestionSelection (Selector: integer; VAR CountOfQuestions: integer; VAR Flag: boolean);

BEGIN

CASE Selector OF

   1:  WriteQuestionText (CountOfQuestions);

   2:  WriteQuestionExplanation;

   3:  EditQuestion;

   4:  EditQuestionExplanation;

   5:  DeleteQuestion (CountOfQuestions);

   6:  DeleteExplanation ('question);

   7:  DisplayQuestion;

   8:  DisplayExplanation ('question');

   9:  PrintAll;

   10: PrintAll;

   11: Flag:= false;

OTHERWISE

   MenuError (11);

END;  { of CASE }

END;

BEGIN  END.
```

13.14 WriteQuestionText.

The knowledge base consists of questions, solutions and of classifiers which relate the other two. This segment writes the text of a question and stores it on disk.

13.14.1 Pseudocode.

```
find the number of the last question on file
form file name with sequential number
write the text of the question using Writer
store file on disk using PushPen
increment CountOfQuestions
```

13.14.2 Draft Source Code.

```
SEGMENT  WritQust;
        { Write the text of a question and store it on disk. }
insert PASPC
insert PASDOS
PROCEDURE PushPen (directory, FileName: string); EXTERNAL;
PROCEDURE FormFileName (index: integer; title: string; VAR FileName: string); EXTERNAL;
PROCEDURE WriteQuestionText (VAR CountOfQuestions: integer);
VAR  FileName: string;
     QuestionOnFile: string;
     Counter: integer;
BEGIN
ClrScr;
ChDir ('\shell\question');                    { Questions are filed on Question sub-directory }
Counter:= 0;
REPEAT
   Counter:= Counter + 1;
   FormFileName (Counter,'question',FileName);
   FindFile (FileName,QuestionOnFile);
UNTIL QuestionOnFile = '';                    { Until QuestionOnFile returns empty }
writeln ('Enter the text of the question');
writeln ('Question no ',Counter:3);
PushPen ('question',FileName);                { Write question and store in Question sub-directory }
CountOfQuestions:= CountOfQuestions + 1;
ChDir ('\shell')
END;
BEGIN  END.
```

13.15 WriteQuestionExplanation.

This segment writes the explanation text for a question and stores it in the correct sub-directory. When the knowledge engineer chooses a question for which to write an explanation he may accidently input a number for which no question has been written, or for which an explanation has already been written. Both these occurrences must be provided for.

13.15.1 Pseudocode.

```
ask for the number of the question needing explanation text
  IF question not on file
     display warning
  IF explanation already written
     display warning
  ELSE write explanation text with PushPen
```

13.15.2 Draft Source Code.

```
SEGMENT WrtQexpl;
      { Write the text of the explanation of a question. }
insert PASPC
insert PASDOS
PROCEDURE  BlankIn, PressKey, PushPen, FormFileName; EXTERNAL;
FUNCTION  YesNo: boolean; EXTERNAL;
PROCEDURE WriteQuestionExplanation;
VAR   QuestionName, FileName: string;
      Index: integer;
      OK: boolean;
BEGIN
ClrScr;
ChDir ('question');
writeln ('Enter the number of the question whose explanation you want to write.');
writeln ('Then press RETURN.');
GoToXY (8,8);
read (Index);
FormFileName (Index,'question',QuestionName);
FormFileName (Index,'explanation',FileName);
OK:= true;
If fstat (QuestionName) = false THEN BEGIN
   writeln ('No question with this key number is on file.');
```

```
   PressKey;
   ClrScr
END;  { of IF }
IF (fstat(QuestionName) = true) AND (fstat(FileName) = true) THEN BEGIN
   writeln ('There is already an explanation for this question on file.');
   writeln ('Do you want to overwrite it? If so, press ''y'' or ''Y''.');
   GoToXY (8,13);
   OK:= YesNo;
   ClrScr
END;  { of IF }
If (fstat(QuestionName) = true) AND (OK = true) THEN BEGIN
   ClrScr;
   writeln ('Enter the text of the explanation.');
   writeln ('Explanation no ',Index:3);
   PushPen ('question',FileName);                { Write explanation and store in Question sub-directory
   ClrScr
END;  { of IF }
ChDir ('\shell');
END;
BEGIN  END.
```

13.16 EditQuestion.

A question textfile, like any other piece of text, will frequently need to be altered and improved. This procedure edits questions that exist on file.

13.16.1 Pseudocode.

```
prompt for the question to be edited
IF question not on file
   issue warning
display question on screen
edit using Writer
return corrected text to file
```

13.16.2 Draft Source Code.

This procedure has not yet been written.

13.17 EditExplanation.

This procedure edits an explanation of a question or a solution, and stores the amended text in the correct sub-

directory.

13.17.1 Pseudocode.

```
prompt for the question whose explanation is to be edited
IF question is not on file
   issue warning
IF explanation is not on file
   issue warning
ELSE edit text with Writer
```

13.17.2 Draft Source Code.

This procedure has not yet been written.

13.18 DeleteQuestionFile.

A question is represented, in an implemented version of Cortex, by a bit in the classifier string as well as a bit in the essential mask or the usual mask, and it may have an explanation text on file as well as the text of the question itself. All these must be removed when a question is deleted. The succeeding files and the three bit strings must be closed up by re-numbering all the subsequent questions.

13.18.1 Pseudocode.

```
prompt for the number of question to be deleted
    ERASE the question and the explanation files
decrement the number of the next question by 1
    REPEAT for all subsequent questions
decrement the number of the next question explanation by 1
    REPEAT for all subsequent explanations
for classifier, essential and usual mask in turn
    move bits for subsequent questions one place up the
list
    REPEAT until end of the string is reached
```

13.18.2 Draft Source Code.

```
SEGMENT  DelQuest;
        { Delete question file together with any explanation file, and reset classifier bit strings. }
insert common types
insert PASPC
insert PASDOS
PROCEDURE FormFileName (index: integer; title: string; VAR FileName: string); EXTERNAL;
```

```
PROCEDURE ReNumberFiles (Flag, CountOfQuestions: integer; title, FileName: string); EXTERNAL;
PROCEDURE Blankln (number: integer); EXTERNAL;
PROCEDURE DeleteQuestion (VAR CountOfQuestions: integer);
VAR   TextFile: text;
      TempFile: FILE OF classifiertype;
      FileName: string;
      Dummy: boolean;
      Flag, Index: integer;
BEGIN
   writeln ('What is the number of the question to be deleted?');
   read (Flag);
   FormFileName (Flag,'question',FileName);                    { Form question file name }
   ChDir ('question');
   IF fstat (FileName) = true THEN BEGIN
      assign (TextFile, FileName);                             { Connect variable to question disk file }
      erase (TextFile);                                        { Delete selected question disk file }
      ReNumberFiles (Flag,CountOfQuestions,'question');        { Close up succeeding files }
      FormFileName (Flag,'explanation',FileName);              { Form explanation file name }
      IF fstat (FileName) = true THEN BEGIN
         assign (TextFile, FileName);                          { Connect variable to explanation disk
   file }
         erase (TextFile);                                     { Delete explanation disk file }
         ReNumberFiles (Flag,CountOfQuestions,'explanation');  { Close up succeeding files }
      END;  { of IF }
   END;  { of IF }
   CountOfQuestions:= CountOfQuestions + 1;
   ChDir ('class');
   IF fstat ('Classif') = true THEN BEGIN
      assign (TempFile, 'Classif');                            { Connect variable to classifier disk file }
      reset (TempFile);
      WHILE NOT eof(TempFile) DO BEGIN
   FOR Index:= Flag TO CountOfQuestions DO BEGIN
      WHILE (Index+1) <= MaxNumberOfQuestions DO BEGIN
         IF testbit(TempFile^.essentialmask,Index) <> testbit(TempFile^.essentialmask,Index+1) THEN
            Dummy = flipbit(TempFile^.essentialmask,Index)          { Flip bit to value of next bit }
         IF testbit(TempFile^.usualmask,Index) <> testbit(Tempfile^.usualmask,Index+1) THEN
            Dummy = flipbit(TempFile^.usualmask,Index)              { Flip bit to value of next bit }
         IF testbit(TempFile^.classifier,Index) <> testbit(TempFile^.classifier,Index+1) THEN
```

```
                Dummy = flipbit(TempFile^.classifier,Index)                { Flip bit to value of next bit
            END;  { of WHILE }
        END;  { of FOR }
    get (TempFile);
    END;  { of WHILE }
END;  { of IF }
ClrScr;
END;
BEGIN END.
```

13.19 ReNumberFiles.

When a file is deleted, the succeeding files must be re-numbered so as to close up the series. This segment performs the necessary operations.

13.19.1 Pseudocode.

```
begin with the file immediately succeeding the deleted file
FOR this file TO end of list of disk files
    rename file to name of previous file
    go to next file
```

13.19.2 Draft Source Code.

```
SEGMENT ReNum;
        { Re-number a series of disk files. }
PROCEDURE FormFileName (index: integer; title: string; VAR FileName: string); EXTERNAL;
PROCEDURE  ReNumberFiles (Flag, CountOfQuestions: integer; title: string); EXTERNAL;
VAR   TextFile: text;
      FileName, ThisFileName, NextFileName: string;
      Index: integer;
BEGIN
FormFileName (Flag,title,FileName);                              { Form name of deleted file }
ThisFileName:= FileName;
FOR Index:= (Flag + 1) TO CountOfQuestions DO BEGIN
FormFileName (Flag,title,FileName);                              { Form name of next file }
NextFileName:= FileName;
IF fstat (NextFileName) = true THEN BEGIN
   assign (TextFile, NextFileName);                               { Connect variable with the next disk file }
   rename (TextFile, ThisFileName);                          { Rename disk file with the name of the previous file }
   close (TextFile, true)
END;  { of IF }
```

```
   ThisFileName:= NextFileName                                    { Update variable }
END;  { of FOR }
END;
BEGIN  END.
```

13.20 DeleteExplanation.

The text of an explanation of a question or a solution will often need to be deleted from the disk. However, a question may or may not have an explanation written for it. Explanations do not, like questions and solutions, form a continuous series and renumbering of files or closing up of bit strings is therefore unnecessary when an explanation is deleted.

13.20.1 Pseudocode.

```
Prompt for the number of the question or solution to be
deleted
form the name of the explanation
check that it is on file
  IF not
     issue warning
  ELSE erase file
```

13.20.2 Draft Source Code.

```
SEGMENT  DelExpl;
        { Delete the explanation of a question or solution. }
insert PASPC
insert PASDOS
PROCEDURE Blankln, PressKey; EXTERNAL;
PROCEDURE FormFileName (index: integer; title: string; VAR FileName: string); EXTERNAL;
PROCEDURE DeleteExplanation (title: string);
VAR   TextFile: text;
      FileName: string;
      Flag: integer;
BEGIN
ClrScr;
writeln ('What is the number of the ',title,' whose explanation you want to delete?');
read (Flag);
FormFileName (Flag,'explanation',FileName);
ChDir (title);
```

```
IF fstat (FileName) = true THEN BEGIN
   assign (TextFile,FileName);
   erase (TextFile);
END  { of IF }
ELSE BEGIN
   writeln ('No explanation for this ',title.' is on file.');
   PressKey;
END;  { of ELSE }
ChDir ('\shell');
ClrScr;
END;
BEGIN  END.
```

13.21 DisplayQuestion.

The knowledge engineer will need to be able to display the text of a question on the screen. This procedure obtains the display.

13.21.1 Pseudocode.

```
prompt for the question to be displayed
form the name of the question
check that it is on file
IF not
   issue warning
ELSE
   display question using DisplayTextFile
```

13.21.2 Draft Source Code.

```
SEGMENT  DispQust;
        { Display the text of a question on screen. }
insert PASPC
PROCEDURE BlankIn, PressKey, FormFileName, DisplayTextFile; EXTERNAL;
PROCEDURE DisplayQuestion;
VAR   FileName: string;
      Key: integer;
BEGIN
ClrScr;
writeln ('Enter the number of the question that you want to display.');
writeln ('Then press RETURN.');
read (Index);
```

```
FormFileName (Index,'question',FileName);
FileName:= concat ('\shell\question\',FileName);
IF fstat (FileName) = false THEN BEGIN
   writeln ('No question with this key number is on file.');
   PressKey;
   ClrScr;
END  { of IF }
ELSE BEGIN
   ClrScr;
   DisplayTextFile (FileName,'question','Question',Index, 7);
   PressKey (5);
   ClrScr
END;  { of ELSE }
END;
BEGIN  END.
```

13.22 DisplayTextFile.

Calling a file from disk and displaying it on the screen is a task which frequently recurs. This procedure carries out the operation for a specified file.

13.22.1 Pseudocode.

```
connect a temporary file variable to the disk file
open file
read the file a line at a time into a string
write the string to the screen
repeat until end of file
close file
```

13.22.2 Draft Source Code.

```
SEGMENT  DispFile;
        { Display disk file on screen. }
insert PASPC
insert PASDOS
PROCEDURE DisplayTextFile (DiskFile, directory, heading: string; Index, displayline: integer);
VAR  TempFile: text;
     Line: string;
BEGIN
assign (TempFile,DiskFile);
ChDir (directory);
```

```
reset (TempFile);
writeln (heading,' no ',Index:3,'.');
WHILE NOT eof(TempFile) DO BEGIN
   readln (TempFile, Line);
   writeln (Line);
   displayline:= displayline + 1;
END;  { of WHILE }
close (TempFile, true);
ChDir ('\shell');
END;
BEGIN  END.
```

13.23 DisplayExplanation.

This procedure carries out the display on the screen of either a question or a solution explanation disk file.

13.23.1 Pseudocode.

```
prompt for the explanation to be displayed
form the name of the explanation
check that it is on file
IF not
   issue warning
ELSE
   display the file using DisplayTextFile
```

Draft Source Code.

```
SEGMENT  DispExpl;
        { Display the text of the explanation of a question or a solution. }
insert PASPC
PROCEDURE PressKey, Blankln, FormFileName, DisplayTextFile; EXTERNAL;
PROCEDURE DisplayExplanation (title: string);
VAR   FileName: string;
      Index: integer;
BEGIN
ClrScr;
writeln ('Enter the number of the ',title,' whose explanation you want to display.');
writeln ('Then press RETURN.');
read (Index);
FormFileName (Index,'explanation',FileName);
FileName:= concat ('\shell\',title,'\',FileName);
```

```
IF fstat (FileName) = false THEN BEGIN
   writeln ('No explanation of this ',title.' is on file.');
   PressKey (7);
   ClrScr;
END  { of IF }
ELSE BEGIN
   ClrScr;
   DisplayTextFile (FileName,'question','Explanation',Index,7);
   PressKey;
  ClrScr
END;  { of ELSE }
END;
BEGIN  END.
```

13.24 Printall.

When a screen display is not sufficient a printout of a text file may be needed. This procedure obtains a file from disk and sends it to the printer.

13.24.1 Pseudocode.

prompt for the number of the file to printed
form the name of the file
using the Prospero Command procedure
   use the DOS command type>prn to send the file to the printer

13.24.1 Draft Source Code.

This procedure has not yet been written.

13.24 SolutionsMenu

When the knowledge engineer chooses to work on the solutions there are 10 operations that may need to be carried out. This procedure makes the choice between them.

13.24.1 Pseudocode.

display choice of operations

13.12.2 Draft Source Code.

```
SEGMENT     Solution;
             { Selects the operations to be performed on the solution files. }
PROCEDURE   Blankln (number: integer); EXTERNAL;
PROCEDURE   SolutionSelection (Selector: integer; VAR Flag: boolean); EXTERNAL;
```

```
PROCEDURE   SolutionMenu;
      VAR   Flag: boolean;
            Selector: integer;
BEGIN
ClrScr;
Flag:= true;
WHILE Flag = true DO BEGIN
   writeln;
   writeln (' ':18,'Solutions Text Menu.');
   writeln (' ':18,'********************');
   Blankln (2);
   writeln (' ':7,'Do you want to;');
   writeln;
   writeln (' ':10,'1.  Write the text of a solution?');
   writeln (' ':10,'2.  Write the explanation of a solution?');
   writeln (' ':10,'3.  Edit the text of a solution?');
   writeln (' ':10,'4.  Edit the explanation of a solution?');
   writeln (' ':10,'5.  Delete a solution from the solutions file?');
   writeln (' ':10,'6.  Delete the explanation of a solution from the file?');
   writeln (' ':10,'7.  Display the text of a solution?');
   writeln (' ':10,'8.  Display the explanation of a solution?');
   writeln (' ':10,'9.  Print the text of a solution?');
   writeln (' ':10,'10. Print the explanation of a solution?');
   writeln (' ':10,'11. Return to the knowledge base menu?');
   Blankln (2);
   writeln ('Make your choice by entering a key number.');
   writeln ('Then press RETURN.');
   read (Selector);
   SolutionSelection (Selector,Flag);
END;  { of WHILE }
END;
BEGIN  END.
```

13.26 SolutionSelection.

Carries out the calling of procedures from the solutions menu.

13.26.1 Pseudocode.

CASE

procedure calls

OTHERWISE

menuerror

13.26.2 Draft Source Code.

```
SEGMENT    SolSelec;
           { Manages the calling of procedures by the solutions menu. }
{$I \PROPAS\PASPC}
PROCEDURE  WriteSolutionText; EXTERNAL;
PROCEDURE  WriteSolutionExplanation; EXTERNAL;
PROCEDURE  EditSolution; EXTERNAL;
PROCEDURE  EditSolutionExplanation; EXTERNAL;
PROCEDURE  DeleteSolution; EXTERNAL;
PROCEDURE  DeleteExplanation (title: string); EXTERNAL;
PROCEDURE  DisplaySolution; EXTERNAL;
PROCEDURE  DisplayExplanation (title: string); EXTERNAL;
PROCEDURE  PrintAll (title: string); EXTERNAL;
PROCEDURE  Blankln (number: integer); EXTERNAL;
PROCEDURE  MenuError (range: integer); EXTERNAL;
PROCEDURE  SolutionSelection (Selector: integer; VAR Flag: boolean);
BEGIN
CASE Selector OF
   1:  WriteSolutionText;
   2:  WriteSolutionExplanation;
   3:  EditSolution;
   4:  EditSolutionExplanation;
   5:  DeleteSolution;
   6:  DeleteExplanation ('solution');
   7:  DisplaySolution;
   8:  DisplayExplanation ('solution');
   9:  PrintAll ('solution');
   10: PrintAll ('solution');
   11: Flag:= false;
   OTHERWISE BEGIN
      ClrScr;
      Blankln (8);
      MenuError (11)
   END;  { of OTHERWISE }
END;  { of CASE }
```

```
END;
BEGIN  END.
```

13.27 WriteSolutionText.

The knowledge base consists of questions, solutions and of classifiers which relate the other two. This segment writes the text of a solution and stores it on disk.

13.27.1 Pseudocode.

```
find the number of the last solution on file
form file name with sequential number
write the text of the solution using Writer
store file on disk using PushPen
increment CountOfSolutions
```

13.27.2 Draft Source Code

```
SEGMENT     WritSoln;
             { Write the text of a solution and store it on disk. }
PROCEDURE   BlankIn (number: integer); EXTERNAL;
PROCEDURE   FormFileName (index: integer; title: string; VAR FileName: string); EXTERNAL;
PROCEDURE   PushPen (title, FileName: string); EXTERNAL;
PROCEDURE   WriteSolutionText (VAR CountOfSolutions: integer);
      VAR   FileName: string[30];
             SolutionOnFile: string[30];
             Counter: integer;
BEGIN
ChDir ('\shell\solution');                          { Solutions are filed on solution sub-directory }
Counter:= 0;
REPEAT
   Counter:= Counter + 1;
   FormFileName (Counter,'solution',FileName);
   FindFile (FileName,SolutionOnFile);
UNTIL SolutionOnFile = '';                          { Until SolutionOnFile returns empty }
writeln (' ':7,'Enter the text of the solution.');
writeln ('Solution no ',Counter:3);
PushPen ('solution', FileName);                     { Write solution , and store in 'solution' sub-directory }
CountOfSolutions:= CountOfSolutions + 1;
ChDir ('\shell')
END;
BEGIN  END.
```

13.28 WriteSolutionExplanation.

This segment writes the explanation text for a solution and stores it in the correct sub-directory. When the knowledge engineer chooses a solution for which to write an explanation he may accidently input a number for which no solution has been written, or for which an explanation has already been written. Both these occurrences must be provided for.

13.28.1 Pseudocode.

```
ask for the number of the solution needing explanation text
  IF solution not on file
     display warning
  IF explanation already written
     display warning
  ELSE write explanation text with PushPen
```

13.28.2 Draft Source Code.

```
SEGMENT    WrtSexpl;
           { Write the text of the explanation of a solution. }
PROCEDURE  BlankIn (number:integer); EXTERNAL;
PROCEDURE  PressKey (margin:integer); EXTERNAL;
FUNCTION   YesNo: boolean; EXTERNAL;
PROCEDURE  FormFileName (index:integer;title:string;VAR FileName:string); EXTERNAL;
PROCEDURE  PushPen (title,FileName:string); EXTERNAL;
PROCEDURE  WriteSolutionExplanation;
     VAR   SolutionName, FileName: string;
           Index: integer;
           OK: boolean;
BEGIN
ClrScr;
ChDir ('solution');
writeln (' ':7,'Enter the number of the solution');
writeln (' ':7,'whose explanation you want to write.');
writeln;
writeln (' ':7,'Then press RETURN.');
GoToXY (8,8);
read (Index);
FormFileName (Index,'solution',SolutionName);
FormFileName (Index,'explanation',FileName);
```

```
OK:= true;
IF fstat (SolutionName) = false THEN BEGIN
   writeln (' ':7,'No solution with this key number is on file.');
   PressKey (7);
END;  { of IF }
IF (fstat(SolutionName) = true) AND (fstat(FileName) = true) THEN BEGIN
   writeln (' ':7,'There is already an explanation for this solution on file.');
   writeln (' ':7,'Do you want to overwrite it? If so, press ''y'' or ''Y''.');
   GoToXY (8,13);
   OK:= YesNo;
END;  { of IF }
IF (fstat(SolutionName) = true) AND (OK = true) THEN BEGIN
   Blankln (2);
   writeln (' ':7,'Enter the text of the explanation.');
   writeln ('Explanation no ',Index:3);
   PushPen ('solution',FileName);          { Write explanation and store in Solution sub-directory }
END;  { of IF }
ChDir ('\shell');
END;
BEGIN  END.
```

13.29 EditSolution.

A solution textfile, like any other piece of text, will frequently need to be altered and improved. This procedure edits solutions that exist on file.

13.16.1 Pseudocode.

```
prompt for the solution to be edited
IF solution not on file
   issue warning
display solution on screen
edit using Writer
return corrected text to file
```

13.16.2 Draft Source Code.

```
SEGMENT    EditSoln;
           { Edit the text of an existing solution. }
PROCEDURE  PressKey (margin:integer); EXTERNAL;
PROCEDURE  Blankln (number:integer); EXTERNAL;
PROCEDURE  EditSolution;
```

```
BEGIN
ClrScr;
Blankln (8);
writeln (' ':7,'The procedure EditSolution has not yet been written.');
Blankln (15);
PressKey (7);
ClrScr;
END;
BEGIN  END.
```

13.30 DeleteSolutionFile.

A solution is represented, in an implemented version of Cortex, by a classifier bit-string, and it may have an explanation text on file as well as the text of the solution itself. Both must be removed when a solution is deleted. The succeeding files and bit strings must be closed up by re-numbering all the subsequent solutions.

13.30.1 Pseudocode.

```
prompt for the number of solution to be deleted
    ERASE the solution and the explanation files
decrement the number of the next solution by 1
    REPEAT for all subsequent solutions
decrement the number of the next solution explanation by 1
    REPEAT for all subsequent explanations
```

13.30.2 Draft Source Code.

```
SEGMENT     DelSoln;
            { Delete solution file. }
PROCEDURE   PressKey (margin:integer); EXTERNAL;
PROCEDURE   Blankln (number:integer); EXTERNAL;
PROCEDURE  DeleteSolution;
BEGIN
writeln (' ':7,'The procedure DeleteSolution has not yet been written.');
PressKey (7);
END;
BEGIN  END.
```

13.31 ReNumberClassifier.

When a solution file is deleted the corresponding classifi-

er bit-string must be removed from the file 'Classif'.

13.31.1 Pseudocode.

find the next record using Seek
write this to the preceding record (the one to be deleted)
repeat until end of file

13.31.2 Draft Source Code.

This procedure has not yet been written.

13.32 DisplaySolution.

The knowledge engineer will need to be able to display the text of a solution on the screen. This procedure obtains the display.

13.32.2 Pseudocode.

prompt for the solution to be displayed
form the name of the solution
check that it is on file
IF not
   issue warning
ELSE
   display question using DisplayTextFile

13.32.2 Draft Source Code.

```
SEGMENT     DispSoln;
             { Display the text of a question on screen. }
PROCEDURE   PressKey (margin: integer); EXTERNAL;
PROCEDURE   BlankIn (number: integer); EXTERNAL;
PROCEDURE   FormFileName (filenumber:integer; title: string; VAR FileName: string); EXTERNAL;
PROCEDURE   DisplayTextFile (FileName, directory, heading: string; Index, displayline: integer); EXTERNAL;
PROCEDURE   DisplaySolution;
VAR         FileName: string[30];
             Index: integer;
BEGIN
writeln (' ':7,'Enter the number of the solution');
writeln (' ':7,'that you want to display.');
writeln (' ':7,'Then press RETURN.');
GoToXY (8,11);
read (Index);
FormFileName (Index,'solution',FileName);
FileName:= concat ('\shell\solution\',FileName);
```

```
IF fstat (FileName) = false THEN BEGIN
   BlankIn (2);
   writeln (' ':7,'No solution with this key number is on file.');
   PressKey (7);
END  { of IF }
ELSE BEGIN
   DisplayTextFile (FileName, 'solution', 'Solution', Index, 7);
   PressKey (5);
END;  { of ELSE }
END;
BEGIN  END.
```

13.33 ClassifierMenu.

When the knowledge engineer chooses to work on the questions there are three operations that may need to be carried out. This procedure makes the selection between them.

13.33.1 Pseudocode.

display the choice of operations
use a CASE statement to call the relevant procedure

13.33.2 Draft Source Code.

```
SEGMENT     Classify;
             { Select the operations to be performed on the classifier file. }
PROCEDURE   BlankIn (number: integer); EXTERNAL;
PROCEDURE   MenuError (range: integer); EXTERNAL;
PROCEDURE   SetUpClassifier (CountOfQuestions: integer); EXTERNAL;
PROCEDURE   EditClassifier; EXTERNAL;
PROCEDURE   DeleteClassifier; EXTERNAL;
PROCEDURE   ClassifierMenu (CountOfQuestions: integer);
      VAR   Flag: boolean;
             Selector: integer;
BEGIN
ClrScr;
Flag:= true;
WHILE Flag = true DO BEGIN
   writeln (' ':20,'Classifier Menu.');
   writeln (' ':20,'****************');
   writeln (' ':11,'Do you want to;');
   writeln (' ':14,'1. Set up a classifier?');
```

```
writeln (' ':14,'2. Edit a classifier?');
writeln (' ':14,'3. Delete a classifier?');
writeln (' ':14,'4. Return to the Knowledge Engineering Menu?');
writeln (' ':7,'Make your choice by typing a keynumber.');
writeln (' ':7,'Then press RETURN.');
read (Selector);
CASE Selector OF
   1: SetUpClassifier (CountOfQuestions);
   2: EditClassifier;
      3: DeleteClassifier;
      4: Flag:= false;
      OTHERWISE BEGIN
          ClrScr;
          Blankln (8);
          MenuError (4)
      END;  { of OTHERWISE }
   END;  { of CASE }
END;  { of WHILE }
END;
BEGIN  END.
```

13.34 SetUpClassifier.

When the texts of the questions and the solutions have been written and stored on file, the knowledge engineer must set them into relationship with each other. This is done by writing a classifier for every solution.

When the part of CORTEX which is concerned with writing classifiers is entered, the knowledge engineer calls up the solution for which he wants to write a classifier. The text of the solution is displayed at the top of the screen. On the bottom half of the screen is displayed the text of the first question in the questions sub-directory. The display asks for an answer to two queries.

First, is it essential that this question be answered correctly if the solution is to be true? If so the first bit in the essential mask is set to true, otherwise it is left set

to false. Should the question be recorded as not essential, a further query is displayed asking if it is usually necessary for the question to be answered correctly. If it is, the first bit in the usual mask is set to true. Otherwise, it is left set to false and the next question from the questions sub-directory is displayed on the screen.

When the knowledge engineer defines a question as either essential or usual an additional query is displayed asking whether the answer must be true or false. His answer is recorded in the first bit of the classifier, whereupon the next question in the questions sub-directory is displayed. The process is then repeated for the next question, and so on through all the questions in the knowledge base. When all the questions have been defined for the first solution the knowledge engineer can call another solution onto the screen, or he can exit to the CORTEX main menu.

The effect of working through all the solutions in the knowledge base in this manner is that every solution is provided with its own classifier, essential mask and usual mask. This information is stored as a disk file so that it is available to the procedures which are brought into play by user of the system. The skill and knowledge which goes into the way that the classifiers are set up is the principle factor in determining the intelligence with which the implemented system will operate.

13.34.1 Pseudocode.

```
add record type to common type file
declare TempFile of common type item3
declare Temp a record of common type item3
set Temp fields to zero
display "What is the number of the solution for which you
want to set up a classifier?"
read SolutionNumber: integer
display solution text
call SetClassifierBits to
    display the question texts in succession
```

```
    manipulate the bit strings
    store the record Temp on the disk file Classif
END  ( of FOR )
start over or exit
```

13.34.2 Draft source code.

```
SEGMENT StUpClas;
      ( Write the classifier, essential mask and usual mask for a solution. )
insert common types
insert PASPC
insert PASDOS
PROCEDURE Blankln (number: integer); EXTERNAL;
PROCEDURE FormFileName (filenumber: integer; title, directory: string; VAR FileName: string); EXTERNAL;
PROCEDURE DisplayTextFile (FileName, directory, heading: string; filenumber, firstline: integer); EXTERNAL
PROCEDURE InitialiseClassifierRecord (SolutionNumber:integer;VAR Temp:solutiontype); EXTERNAL;
PROCEDURE SetClassifierBits (CountOfQuestions:integer;VAR Temp:solutiontype;VAR Flag:boolean); EXTERNAL;
FUNCTION Ask (leftmargin: integer; question: string): boolean; EXTERNAL;
PROCEDURE SetUpClassifier (CountOfQuestions: integer);
VAR  TempFile: FILE OF solutiontype;
     Temp: classifiertype;
     FileName: string;
     SolutionNumber: integer;
     Answer, Flag: boolean;
BEGIN
Flag:= true;
WHILE Flag = true DO BEGIN
   writeln ("What is the number of the solution for which you want to write a classifier?");
   read (SolutionNumber);
   InitialiseClassifierRecord (SolutionNumber,Temp);
   ClrScr;
   writeln ("Solution no ",SolutionNumber:3);
   FormFileName (SolutionNumber,'solution','solution',FileName);
   FileName:= concat ('\SHELL\SOLUTION\',FileName);
   IF fstat (FileName) = false THEN
      writeln ('No solution with this key number is on file.');
   ELSE BEGIN
      DisplayTextFile (filename,'solution','Solution',SolutionNumber,2);    { Display solution text }
      SetClassifierBits (CountOfQuestions,Temp,Flag);
      assign (TempFile,'Classif');              { Connect file variable TempFile to disk file Classif }
```

```
      append (TempFile);                                    { Move pointer to end of file }
      write (TempFile,Temp);                                { Add Temp to end of TempFile }
      close (TempFile,true);
   END;  { of ELSE }
Answer:= Ask (7,'Do you want to write another classifier.');
IF Answer = true THEN BEGIN
   Flag:= true;                                                        { Exit from procedure }
   TextWindow (1,1,25,80)                                { Restore text window to whole screen }
ELSE BEGIN
   Flag:= false;
END;  { of WHILE }
END;
BEGIN  END.
```

13.35 InitialiseClassifierRecord

Before a classifier is written the fields of its record, other than the keynumber, must be set to zero.

13.35.1 Pseudocode.

Set keynumber field to solution number.

Set other fields to zero, using procedure ZeroiseBitString for bit strings

13.35.2 Draft Source Code.

```
SEGMENT InitClas;
       { Intitialise the fields of the record Temp. }
insert common types
PROCEDURE ZeroiseBitString (VAR Bits:bitstring); EXTERNAL;
PROCEDURE InitialiseClassifierRecord (SolutionNumber:integer;VAR Temp:solutiontype);
BEGIN
Temp.keynumber:= SolutionNumber;
ZeroiseBitString (Temp.essentialmask);
ZeroiseBitString (Temp.usualmask);
Temp.totalusual:= 0;
Temp.usualtrue:= 0;
Temp.probability:= 0;
ZeroiseBitString (Temp.classifier);
END;
BEGIN  END.
```

13.36 SetClassifierBits.

This procedure manipulates the classifier bit strings.

13.36.1 Pseudocode.

```
FOR index:= 1 TO CountOfQuestions DO BEGIN
    display text of first question
    display "Is it essential for this question to receive
                a correct answer?"
IF "yes" set essential mask bit to T
    setbit (Temp.essentialmask,index);
ELSE display "Does the solution usually require a correct
                answer to this question?"
IF "yes" set usual mask bit to T
    setbit (Temp.usualmask,index);
IF essential or usual mask set to T
    display "Does the solution require this question to be
                answered by 'yes' or by 'no'. Y/N?"
IF answer is 'yes' set the classifier bit to T
    setbit (Temp.classifier,index);
append Temp to disk file
```

13.36.2 Draft Source Code.

```
SEGMENT  ClasBits;
        { Set the bits of the classifier and its masks. }
insert common types
insert PASPC
insert PASDOS
PROCEDURE FormFileName (filenumber:integer;title,directory:string;VAR FileName:string); EXTERNAL;
PROCEDURE DisplayTextFile (FileName,directory,heading:string;filenumber,firstline:integer); EXTERNAL;
PROCEDURE SetUsualBits (Index:integer;VAR Temp:solutiontype); EXTERNAL;
FUNCTION Ask (leftmargin:integer;question:string): boolean; EXTERNAL;
FUNCTION YesNo: boolean; EXTERNAL;
PROCEDURE SetClassifierBits (CountOfQuestions:integer;VAR Temp:solutiontype;VAR Flag:boolean);
VAR  TempFile: FILE OF solutiontype;
     FileName: string;
     Index: integer;
     Answer, Dummy: boolean;
BEGIN
FOR Index:= 1 TO CountOfQuestions DO BEGIN                    { Loop through all the questions }
   TextWindow (1,10,80,25);
```

```
FormFileName (Index,'question',FileName);
FileName:= concat ('\shell\question\',FileName);
DisplayTextFile (FileName,'question','Question',Index,8);              { Display question text }
GoToXY (8,7);
Answer:= Ask (1,'Is it essential that this question receives a correct answer. Y/N?');
IF Answer = true THEN BEGIN
   Dummy:= setbit (Temp.essentialmask,Index);                          { Set essential mask to true }
   Answer:= Ask (7,'Is the answer that is always needed to this question ''yes'' or ''no''. Y/N?');
   IF Answer = true THEN
      Dummy:= setbit (Temp.classifier,Index);
   END  { of IF }
   ELSE
      SetUsualBits (Index,Temp);
END;  { of FOR }
END;
BEGIN  END.
```

13.37 SetUsualBits.

This procedure manipulates the bits of the classifier and its masks that record the status of usual questions.

13.37.1 Pseudocode.

```
when a question is defined as not essential
prompt for usual question
IF answer is 'yes'
     set usual mask to true
     increment totalusual field
     prompt for answer true or false
IF true
     set bit to true
```

13.37.2 Draft Source Code.

```
SEGMENT UsulBits;
       { Set the usual bits in a classifier. }
insert common types
insert PASPC
insert PASDOS
FUNCTION YesNo: boolean; EXTERNAL;
FUNCTION Ask (leftmargin:integer;question:string): boolean; EXTERNAL;
PROCEDURE SetUsualBits (Index: integer);
```

```
VAR  Temp: classifiertype;
     Answer, Dummy: boolean;
BEGIN
writeln ('Does the solution usually require');
writeln (' ':7,'a correct answer to this question. Y/N?');
Answer:= YesNo;
IF Answer = true THEN BEGIN
   Dummy:= setbit (Temp.usualmask,Index);                    { Set usual mask to true }
   Temp.totalusual:= Temp.totalusual + 1;
   ClrScr;
   writeln ('Is the answer that is usually needed');
   writeln (' ':7,'to this question ''yes'' or ''no''. Y/N?');
   Answer:= YesNo;
      IF Answer = true THEN
         Dummy:= setbit (Temp.classifier,Index);             { Set classifier to true }
END;  { of IF }
END;
BEGIN  END.
```

13.38 EditClassifier.

Adjustments and improvements to the way that Cortex functions are made by working on the questions and solutions, and also by editing the classifiers that establish the relations between them. This procedure edits a selected classifier and places the revised version on file in the disk file Classif.

13.38.1 Pseudocode.

```
prompt for the solution whose classifier is to be edited
if the solution has no classifier on file
    issue a warning
ELSE call set up classifier
    rewrite classifier
    store revised classifier on file
```

13.38.2 Draft Source Code.

```
SEGMENT    EditClas;
           { Edit a classifier. }
insert common types
PROCEDURE  Blankln (number:integer); EXTERNAL;
```

```
PROCEDURE   FormFileName (filenumber:integer;title:string;VAR FileName:string); EXTERNAL;
FUNCTION    Ask (leftmargin:integer;question:string): boolean; EXTERNAL;
PROCEDURE   EditClassifier;
      VAR   TempFile: FILE OF solutiontype;
             SolutionFile, QuestionFile: text;
             TempRecord: solutiontype;
             FileName: string[30];
             Line: string[100];
             SolutionNumber, QuestionNumber, Counter: integer;
                Answer, ClassifierSetting, Dummy: boolean;
BEGIN
writeln (' ':7,'Enter the number of the solution whose classifier you want to edit.');
read (SolutionNumber);
writeln (' ':7,'The text of the solution whose classifier you are editing is;');
FormFileName (SolutionNumber,'solution',FileName);
FileName:= concat ('\shell\solution\',FileName);
assign (SolutionFile,FileName);
reset (SolutionFile);
Counter:= 8;
WHILE NOT eof(SolutionFile) DO BEGIN
   readln (SolutionFile, Line);
   writeln (Line);
   Counter:= Counter + 1;
END;  { of WHILE }
writeln (' ':7,'Enter the number of the question that you want to change.');
read (QuestionNumber);
writeln (' ':7,'The question whose bit you are editing is;');
FormFileName (QuestionNumber,'question',FileName);
FileName:= concat ('\shell\question\',FileName);
assign (QuestionFile,FileName);
reset (QuestionFile);
Counter:= 16;
WHILE NOT eof(QuestionFile) DO BEGIN
   readln (QuestionFile, Line);
   writeln (Line);
   Counter:= Counter + 1
END;  { of WHILE }
assign (TempFile,'Classif');
```

```
update (TempFile);
seek (TempFile,(SolutionNumber - 1));
read (TempFile, TempRecord);
ClassifierSetting:= testbit(TempRecord.classifier,QuestionNumber);
writeln (' ':7,'The setting of this question bit is ',ClassifierSetting);
Answer:= Ask (7,'Do you want to change its setting. Y/N?');
IF Answer = true THEN BEGIN
   Dummy:= flipbit(TempRecord.classifier,QuestionNumber);
   seek (TempFile,(SolutionNumber - 1));
   write (TempFile,TempRecord);
   close (TempFile,true)
END;  { of IF }
END;
BEGIN  END.
```

13.39 DeleteClassifier.

If a solution is no longer relevant, but its text is to be retained on disk, the corresponding classifier will need to be deleted. This procedure removes a classifier from the disk file Classif. Classifiers are stored as elements of this file, and when an element is removed the succeeding elements must be closed up so as to maintain a continuous sequence.

13.39.1 Pseudocode.

```
prompt for the solution whose classifier is to be deleted
go to file element with this key number
IF no file element has this number
    issue warning
ELSE read the next file element into a temporary record
overwrite the element for deletion with the temporary
record
repeat for all succeeding file elements
```

13.39.2 Draft Source Code.

```
SEGMENT DelClass;
        { Delete a specified classifier and close up the succeeding file elements. }
insert PASPC
insert PASDOS
PROCEDURE DeleteClassifier (CountOfQuestions: integer);
```

```
VAR    TempFile: FILE OF classifiertype;
       Temp: classifiertype;
       Selector, Counter, Index: integer;
BEGIN
writeln ('What is the number of the solution whose classifier you want to delete?');
read (Selector);
assign (TempFile,'Classif');
reset (TempFile);
WHILE NOT eof(TempFile) DO BEGIN
  IF TempFile^.keynumber <> Selector THEN                    { Filepointer not at selected element }
    get (TempFile);                                              { Go to next element }
  IF eof (TempFile) = true THEN
    writeln ('No classifier has been written for this solution.')  { Selected classifier not found }
  ELSE BEGIN                                                 { File pointer is at selected element }
    Counter:= Selector;
    FOR Index:= Selector TO CountOfQuestions DO BEGIN
      seek (TempFile,(Counter+1));                           { Go to next file element }
      Temp.essentialmask:= TempFile^.essentialmask;          { Set Temp's fields to this element's
values }
      Temp.usualmask:= TempFile^.usualmask;
      Temp.classifier:= TempFile^.classifier;
      Temp.totalusual:= TempFile^.totalusual;
      Counter:= Counter - 1;
      seek (TempFile,Counter);                                   { Go back to selected file element }
      write (TempFile,Temp);                                 { Overwrite element with values of next element }
      seek (TempFile,(Counter+1));                               { Go to next file element }
    END;  { of FOR }
  END;  { of ELSE }
END;  { of WHILE }
close (TempFile, true);
END;
BEGIN  END.
```

13.40.1 FindResult.

When the user starts up CORTEX the system presents the welcome screen followed by the implementation screen. This tells him what is the domain in which CORTEX has been implemented, and it may contain advice on how to answer the

questions to the best effect.
The system, using SetUpSolutionList and MostFrequentQuestion, creates a linked list of all the solutions in the knowlege base. The essential masks are then searched and the number of the question that occurs in them most often is ascertained. Upon exiting from the implementation screen the user is presented with this first question on the screen and he is prompted for an answer.

The user provides an answer to the question which follows from his knowledge of, and point of view towards, the implementation domain. His answers are recorded in the message and message mask strings by the procedure MessageAndMask. But an answer to a question will, in logic and sense, make some of the other questions in the knowledge base redundant. For example, if in an animal identification system the user replies that the creature lays eggs, then it is otiose to go on to ask if the same creature gives milk. For the sake of completeness, one may observe that the last statement holds true unless one is wading in a Tasmanian swamp and the creature under inspection happens to be a duckbilled platypus. An expert system must be provided with a means of excluding from the list of questions those which have been rendered irrelevant by already answered questions.

As soon as an answer is given the system eliminates from the list of solutions all those solutions whose bits contradict the answer. The remaining solutions are again searched by MostFrequentQuestion and the question that appears in them most often is displayed to the user for answering. This process is repeated until a possible solution is obtained, or until all the questions in the knowledge base have been answered and no solution has been arrived at. In this way the number of solutions to be searched is progressively reduced as the user provides more answers to questions.

Every solution for which all the essential questions have been correctly answered is a possible solution. The best candidate is identified by searching their usual masks, and obtaining the answer to all the questions that occurs in those masks. When all the questions that appear in the usual masks of the possible solutions have been answered, the probabilities of each can be calculated and the most likely solution is then displayed.

By means of this process of exclusion, the number of relevant questions and possible solutions is quickly restricted. Such a method of ordering the presentation of questions has the double advantage of excluding irrelevant questions, and of narrowing the search onto the most promising solution candidates. This is in accord with the way in which a human expert might work. He will devote most attention to the candidate solutions which emerge as the most likely to be correct, while progressively excluding those which appear unpromising.

13.40.2 Pseudocode.

```
set Message and MessageMask arrays to zero
call SetUpSolutionList to create linked list of solutions
call MostFrequentQuestion to find most common question in
                                                the solutions
WHILE there is another essential question to be found
   call MessageAndMask to record the user's answer to a
question
   call RemoveSolution to remove any solution contradicted
                                                by the answer
   IF all elements of the solutions list are removed
      display failure message and exit progam
   call UsualQuestion to search the usual masks of the
                                                possible solutions
   get answers to the usual questions of the possible solu-
                                                tions
   call Probability to calculate the probability of each
                                                solution
   display the most probable result with its probability
```

finish by clearing the solutions list from memory

13.40.3 Draft source code.

```
SEGMENT  FindRslt;
        { Main procedure for writing the message and obtaining the result. }
insert common types
PROCEDURE ZeroiseBitString (VAR Bit:bitstring); EXTERNAL;
PROCEDURE SetUpSolutionList (CountOfSolutions:integer;VAR Head:solutionpointer); EXTERNAL;
PROCEDURE MessageAndMask (MostFrequentQuestion:integer;VAR Message,MessageMask:bitstring); EXTERNAL;
PROCEDURE RemoveContradictedSolution (Message,MessageMask:bitstring;CountOfQuestions:integer;
                                      VAR Head:solutionpointer); EXTERNAL;
PROCEDURE CalculateProbability (Message,MessageMask:bitstring;CountOfQuestions:integer
                                      Head:solutionpointer); EXTERNAL;
PROCEDURE DisplayResult (Head:solutionpointer); EXTERNAL;
PROCEDURE ClearHeap (Head:solutionpointer); EXTERNAL;
FUNCTION FindMostFrequentQuestion (MessageMask:bitstring;CountOfQuestions:integer;
                                      Head:solutionpointer): integer; EXTERNAL;
FUNCTION FindUsualQuestion (CountOfQuestions:integer;Head:solutionpointer): integer; EXTERNAL;
PROCEDURE  FindResult (CountOfQuestions,CountOfSolutions:integer);
VAR        Message, MessageMask: bitstring;
           Head: solutionpointer;
           UQ, MFQ: integer;
BEGIN
ZeroiseBitString (Message);                                    { Set Message and MessageMask to zero }
ZeroiseBitString (MessageMask);
SetUpSolutionList (CountOfSolutions,Head);                     { Create solutions list }
MFQ:= FindMostFrequentQuestion (MessageMask,CountOfQuestions,Head); { Find number of MFQ }
WHILE MFQ <> 0 DO BEGIN                                { Search essential masks and prune solutions list }
   MessageAndMask (MFQ,Message,MessageMask);                   { Set Message and MessageMask for MFQ }
   RemoveContradictedSolutions (Message,MessageMask,MFQ,Head); { Remove any contradicted solution }
   MFQ:= FindMostFrequentQuestion (MessageMask,CountOfQuestions,Head);  { Find no of MFQ remaining }
END;  { of WHILE }
UQ:= FindUsualQuestion (CountOfQuestions,Head);                { Find number of UQ }
WHILE UQ <> 0 DO BEGIN
   MessageAndMask (UQ,Message,MessageMask)                  { Set Message and MessageMask for UQ }
   UQ:= FindUsualQuestion (CountOfQuestions,Head);             { Find number of next UQ }
END;  { of WHILE }
CalculateProbability (Message,MessageMask,CountOfQuestions,Head);
DisplayResult (Head);
```

```
ClearHeap (Head)
END;
BEGIN  END.
```

13.41 SetUpSolutionList.

This segment creates a linked list of all the solutions in the database. A linked list keeps track of relevant solutions more efficiently than arrays because no-longer-relevant elements can be erased from the list readily. It is impossible to erase elements selectively from an array, and therefore the length of a search path through an array cannot be curtailed.

The values of the essential mask, usualmask and classifier fields are copied from the disk file of the classifier into the corresponding fields of the linked list. It is by consulting the value of the essential masks in the list that the possible solutions are identified, while the classifier and the usual mask enable the probability of the possible solutions to be calculated.

13.41.1 Pseudocode.

```
make space in memory for the last element in the solution
list
set its key to total number of solutions and its pointer to
NIL
working back through the total number of solutions
  make space for temporary variable
  store the loop index as its serial number
  link this element to current variable
  make the new element the current variable
finish by making the head pointer the current variable
link a variable to the disk file containing the classifier
  working through the total number of solutions
  copy classifier, masks and totalusual fields from file
                                  variable to current pointer
  set usualtrue and probable fields to zero
get the next element of the file variable
```

13.41.2 Draft Source Code.

```
SEGMENT  SetUpSol;
        { Set up a linked list of solutions and return the Head pointer. }
insert global types
PROCEDURE SetUpSolutionList (CountOfSolutions: integer; VAR Head: solutionpointer);
VAR       Classifier: FILE OF solutiontype;
          TempRecord: solutiontype;
          Temp, Current: solutionpointer;
          Index: integer;
BEGIN
  new (Current);                                    { Make space in heap for last element in the list }
  Current^.KeyNumber:= CountOfSolutions;            { Number of last element := total number of solutions }
  Current^.Next:= NIL;                                  { Set pointer field of last element to NIL}
  FOR Index:= (CountOfSolutions - 1) DOWNTO 1 DO BEGIN  { Work backwards from last element in the list }
    new (Temp);                                     { Make space in heap for a new element in the list }
    Temp^.keynumber:= Index;                        { Set keynumber to loop index }
    Temp^.next:= Current;                           { Set to point to current element in the list }
    Current:= Temp                                  { Make the current element the new element}
  END;  { of FOR }
  Head:= Current;  { Move head pointer to first solution, which is current pointer, on exiting from FOR loop
  assign (Classifier, 'Classif');  { Connect Classifier to disk file 'Classif', written with SetUpClassifier
  reset (Classifier);
  Temp:= Head;                                      { Position Head at beginning of solutions list }
  FOR Index:= 1 TO CountOfSolutions DO BEGIN
    read (Classifier,TempRecord);                   { By-pass record 0, which is blank }
    Temp^.essentialmask:= Classifier^.essentialmask;  { Set essentialmask to value in Classifier field }
    Temp^.usualmask:= Classifier^.usualmask;        { Set usualmask to value in Classifier field }
    Temp^.classifier:= Classifier^.classifier;      { Set classifier to value in Classifier field }
    Temp^.totalusual:= Classifier^.totalusual;      { Set totalusual to value in Classifier field }
    Temp^.usualtrue:= 0;
    Temp^.probability:=0;
    Temp:= Temp^.next;                              { Move pointer to next record }
  END;  { of FOR }
  END;
  BEGIN  END.
```

13.42 FindMostFrequentQuestion.

When the implemented shell is started up, the first question to be presented to the user must be that which occurs

most frequently in the essential masks. This segment finds thatquestion. Subsequent questions to be presented to the user are always the most frequent to occur in the remaining relevant solutions, and this segment finds these also. The linked list of solutions which was created in SetUpSolutionList contains fields for the essential and the usual mask applying to each solution. The solutions list serves as the vehicle for the search.

If no more unanswered questions can be found in the essential masks of the elements of the solution list, then those elements that remain in the list are all possible solutions. In this case, the function returns a value of 0 and control passes to the segment UsualQuestion to search the usual masks, rather than the essential masks, of the candidate solutions.

13.40.1 Pseudocode.

```
declare type in common types file
declare variable, TopQuestion, to record
    (i) the key number of the most frequent question so far
    (ii) the number of times it occurs in the essential
masks
declare another variable, QuestionCounter, to record
(i) the key number of the question about to be counted
(ii) the number of times it occurs in the essential masks
set both fields of TopQuestion to zero
for each question in turn,
(i) go to the first element in the solution list
(ii) set QuestionCounter.questionnumber to 1
(iii) use testbit to read the bit representing question 1
(iv) increment QuestionCounter.countofoccurrance if the bit
                                                    is set
to true
(v) IF QuestionCounter.countofoccurrances >
                                    TopQuestion.countofoccurrances
                set TopQuestion = QuestionCounter
IF there are no more essential questions to be asked
then all the questions needed to identify all the possible
```

solutions have been asked

in which case, the function returns the value of zero ELSE assign TopQuestion.questionnumber to MostFrequentQuestion, and return this value to the calling procedure FindResult

13.42.2 Draft Source Code.

```
SEGMENT MFrqQust;
insert common types
insert PASDOS
FUNCTION  FindMostFrequentQuestion (MessageMask:bitstring;CountOfQuestions:integer;
                                    Head:solutionpointer): integer;
VAR       Current: solutionpointer;
          TopQuestion, QuestionCounter: questiontype;
          Index: integer;
          AllFalseFlag, AllTrueFlag: boolean;
BEGIN
TopQuestion.questionnumber:= 0;
TopQuestion.countofoccurrance:= 0;
FOR Index:= 1 TO CountOfQuestions DO BEGIN
   QuestionCounter.questionnumber:= Index;
   QuestionCounter.countofoccurrance:= 0;
   Current:= Head;
   WHILE Current <> NIL DO BEGIN
      IF (testbit (Current^.essentialmask,Index) = true) AND (testbit (MessageMask,Index) = false) THEN
         QuestionCounter.countofoccurrance = (QuestionCounter.countofoccurrance + 1);
         IF testbit(Current^.classifier,Index) = true THEN
            AllTrueFlag:= true
         ELSE
            AllFalseFlag:= true;
      END;  { of IF }
   Current:= Current^.next
   END;  { of WHILE }
   IF AllFalseFlag AND AllTrueFlag = true THEN
      IF QuestionCounter.countofoccurrance > TopQuestion.countofoccurrance THEN
         TopQuestion:= QuestionCounter
```

```
END;  { of FOR }
FindMostFrequentQuestion:= TopQuestion.questionnumber   { No of remaining most freq occurring question }
END;
BEGIN  END.
```

13.43 MessageAndMask.

When a question is answered by the user the bit in the message mask corresponding to that question is set to true. Whether the answer is true or false is recorded in the message. The code in this segment sets up these two bit-strings.

The message and the message mask may be several hundred bits long in a realistic implementation of the shell. However, in Prospero Pascal an integer is represented by four eight-bit bytes, or 32 bits. A single CORTEX bit string will consequently represent more than one integer. In an array of integers, however, the 32 bit integers follow sequentially from one array element to the next. A long bit string therefore represents an array of integers, rather than a single integer. For this reason the message and the message array are declared as integer arrays.

13.43.1 Pseudocode.

```
declare message and messagemask as VAR parameters
display most frequently occurring question
display cue for explanation
IF asked, display explanation
ask for answer to the question
IF answer is yes
   set message and messagemask to true
ELSE
   set messagemask to true
   leave message false
```

13.43.2 Draft Source Code.

```
SEGMENT  MessMask;
        { Display the most frequently occurring question and obtain the user's answer to it. }
insert common types
insert PASPC
```

```
insert PASDOS
PROCEDURE Blankln (number: integer); EXTERNAL;
PROCEDURE PressKey (margin:integer); EXTERNAL;
PROCEDURE FormFileName (filenumber: integer; title: string; VAR FileName: string); EXTERNAL;
PROCEDURE DisplayTextFile (DiskFile, directory, heading: string; filenumber, firstline: integer); EXTERNA
FUNCTION Ask (margin:integer; question: string): boolean; EXTERNAL;
PROCEDURE SetMessageAndMask (LiveQuestion: integer; VAR Message, MessageMask: bitstring);
VAR  FileName: string[30];
     Ch: char;
     Answer, Dummy: boolean;
BEGIN
FormFileName (LiveQuestion,'question',FileName);
DisplayTextFile (FileName,'question','Question',LiveQuestion,2);   { Display the question text }
writeln ('Please answer the question ''Yes'' or ''No''.');
writeln ('Press ''W'' for ''Why'' if you want to see an explanation of this question.');
Gate:= ConSilent;
IF (Gate = 'W') OR (Gate = 'w') THEN BEGIN
   FormFileName (LiveQuestion,'explanation',FileName);
   FileName:= concat('\shell\question\',FileName);
   IF fstat (FileName) = false THEN BEGIN
      writeln ('No explanation for this question is on file.');
      PressKey (7);
   END  { of IF }
   ELSE BEGIN
   DisplayTextFile (FileName,'question','Explanation',LiveQuestion,8);  { Display the explanation text }
   END;  { of ELSE }
   REPEAT Gate:= ConSilent UNTIL
   (Gate = 'Y') OR (Gate = 'y') OR (Gate = 'N') OR (Gate = 'n');
END;  { of IF }
IF (Gate = 'Y') OR (Gate = 'y') THEN
  Dummy:= setbit (Message,LiveQuestion);          { Set Message to true if answer is 'yes' }
  Dummy:= setbit (MessageMask,LiveQuestion);      { Set MessageMask to true if answer is 'yes' or 'no' }
END;
BEGIN  END.
```

13.44 RemoveContradictedSolution.

A solution whose essential classifier contains a bit that does not match the corresponding message bit must be incor-

rect. This segment of CORTEX removes from the list of solutions any solution which, for this reason, cannot be correct. The effect of shortening the list is to accelerate the next search for the most frequently occurring question.

A question removes a solution from the list because one of the solution's essential questions has been found to be contradicted by the answer. It follows, as a corollary, that all the solutions which remain in the solutions list must be possible solutions if the answer to a question fails to remove any solution. This is an important milestone in the working of CORTEX bacause from this point onwards it is no longer necessary to search for the most frequently occurring question. It is sufficient, when all the remaining solutions are possibly correct, to find which is the most probably correct. For this reason, this procedure sets a switch to true and returns its value to Message as soon as no more elements can be removed from the solutions list.

13.44.1 Pseudocode.

```
go to the first element of the solutions list
compare the message bit with the corresponding essential
mask bit using the Frey algorithm
IF they differ THEN
   remove solutions list element
repeat for all solutions list elements
```

13.44.2 Draft Source Code.

```
SEGMENT RemSoln;
{ Remove from the list of solutions any whose essential mask is contradicted by an answer to a question }
insert global types
PROCEDURE RemoveContradictedSolutions (Message, MessageMask: bitstring; MFQ: integer;
                                       VAR Head: solutionpointer);
VAR   Current, Previous: solutionpointer;
      IntermediateResult1, IntermediatResult2, EssentialResult: boolean;
BEGIN
Current:= Head;                                  { Current set to first element of solutions list }
WHILE Current <> NIL DO BEGIN
   IntermediateResult1:= NOT (testbit(Message,MFQ) XOR testbit(Current^.classifier,MFQ));
```

```
IntermediateResult2:= testbit (MessageMask,MFQ);
IF IntermediateResult1 AND IntermediateResult2 = false THEN   { If the result is false,
  IF Current = Head THEN BEGIN                                  { To delete first solution record }
     Head:= Head^.next;                                         { Move head pointer to next element }
     dispose (Current);                                         { Delete first element }
     Current:= Head;                                 { Set current pointer to what is now the first element }
  END  { of IF }
  ELSE BEGIN                                                    { Delete if non-head element }
     Previous^.next:= Current^.next;                            { By-pass current element }
     dispose (Current);                                         { Delete by-passed element }
     Current:= Previous^.next;                                { Set current pointer equal to next element }
    END  { of ELSE }
   ELSE BEGIN                                          { Not deleting element because still possible solution }
     Previous:= Current;                                        { Move previous pointer to current element }
     Current:= Current^.next                                    { Move current pointer to next element }
   END;  { of ELSE }                                          { If Current^.next is NIL, WHILE loop is exited }
END;  { of WHILE }
END;
BEGIN  END.
```

13.45 FindUsualQuestion.

When no element can be removed from the solution list by RemoveContradictedSolutions then all the remaining elements are possible solutions. These possible solutions must be ranked in order of probability so that the most probable can be presented to the user as the solution to the problem. This is done by obtaining answers to all the questions that appear in their usual masks, and then choosing the solution with the largest proportion of correctly answered usual questions.

This function returns the number of any question that appears in the usual mask of any of the possible solutions. By checking the message mask for that question each time, those questions that have already received an answer because they are essential to some other solution are not asked again.

13.45.1 Pseudocode.

```
go to the first element of the solutions list
find the first question in the usual mask
IF the message mask for that bit = false
   return the number of the question to FindResult
      so that an answer can be obtained with MessageAndMask
ELSE go to next element in solutions list
go to the next element of the solutions list
```

13.45.2 Draft Source Code.

```
SEGMENT  UsulQust;
        { Return the number of next unanswered usual question for all the possible solutions }
insert global types
FUNCTION FindUsualQuestion (MessageMask:bitstring;CountOfQuestions:integer;Head:solutionpointer): integer;
VAR    Current: solutionpointer;
       Found: boolean;
       Index, QuestionNumber: integer;
BEGIN
Current:= Head;                                    { Go to beginning of solutions list }
Found:= false;
WHILE (Current <> NIL) AND (Found = false) DO BEGIN
   Index:= 1;
   WHILE (Index <= CountOfQuestions) AND (Found = false) DO
      IF (testbit(Current^.usualmask,Index) = true) AND (testbit(MessageMask,Index) = false) THEN BEGIN
         Found:= true;
         QuestionNumber:= Index;
      END  { of IF }
      ELSE
         Index:= Index + 1;
   Current:= Current^.next;
END;  { of WHILE }
IF Found = true THEN
   FindUsualQuestion:= QuestionNumber
ELSE
   FindUsualQuestion:= 0;
END;
BEGIN  END.
```

13.46 CalculateProbability

Answers will have been provided for all the questions that appear in the usual masks of the possible solutions by the time that this procedure is called. The number of usual questions for each solution will have been calculated when the classifiers and their masks were written. This procedure obtains the total of usual questions which have been answered correctly, and calculates a percentage probability of the solution being the correct one.

13.46.1 Pseudocode.

```
go to the beginning of the solutions list
find the first question in the usual mask
obtain a result for this question
IF the result is true
   increment a counter
go to the next usual question
go to the next solutions list element
```

13.46.2 Draft Source Code.

```
SEGMENT Probable;
        { Calculate the probability of the possible solutions being correct. }
insert global types
PROCEDURE CalculateProbability (Message, MessageMask: bitstring; CountOfQuestions: integer;
                                Head: solutionpointer);
VAR     Current: solutionpointer;
        IntermediateResult1, IntermediateResult2, UsualResult: boolean;
        Index: integer;
BEGIN
Current:= Head;                                          { Go to beginning of solutions list }
WHILE Current <> NIL DO BEGIN
   FOR Index:= 1 TO CountOfQuestions DO BEGIN
      IntermediateResult1:= NOT (testbit(Message,Index) XOR testbit(Current^.classifier,Index));
      IntermediateResult2:= testbit(MessageMask,Index) AND testbit(Current^.usualmask,Index);
      UsualResult:= IntermediateResult1 AND IntermediateResult2
      IF UsualResult = true THEN              { True when the answer to the usual question is correct }
         Current^.ususaltrue:= Current^.usualtrue + 1;     { Increment counter }
   END;  { of FOR }
Current^.probability:= (Current^.usualtrue DIV Current^.totalusual) * 100;
Current:= Current^.next                                  { Go to next solutions list element }
```

```
END;  { of WHILE }
END;
BEGIN  END.
```

13.47 DisplayResult.

The system produces the solution that, in the light of the answers supplied by the user to the questions, is most likely to be the correct one. This procedure extracts the text of the most probable solution from disk and displays it together with its probability expressed as a percentage.

13.47.1 Pseudocode.

```
go to the beginning of the solutions list
find the solution with the largest probability
   record its keynumber as a variable
   read its probability field
extract that solution from the solutions disk file
display the text of the solution and its probability
pause the program
```

13.47.2 Draft Source Code.

```
SEGMENT  DispRslt;
        { Display the most likely solution with its probability on screen. }
insert global types
insert PASPC
insert PASDOS
PROCEDURE PressKey (margin: integer); EXTERNAL;
PROCEDURE FormFileName (filenumber: integer; title: string; VAR FileName: string); EXTERNAL;
PROCEDURE DisplayTextFile (DiskFile,directory,heading:string;filenumber,firstline:integer); EXTERNAL;
FUNCTION YesNo (margin: integer); EXTERNAL;
PROCEDURE DisplayResult (Head: solutionpointer);
VAR    Disk: text;
       Current: solutionpointer;
       Line: string[100];
       Gate: char;
       FileName: string[30];
       FrontRunner, Counter: integer;
       Probability: real;
BEGIN
ClrScr;
```

```
IF Head^ = NIL THEN BEGIN
   writeln (' ':7,,I know of no solution that matches these answers.')
   PressKey;
END  { of IF }
ELSE BEGIN
   Current:= Head;                                    { Go to first solution list element }
   FrontRunner:= Current^.keynumber;
   Probability:= Current^.probability;
   WHILE Current <> NIL DO BEGIN
      IF Current^.probability > Probability THEN BEGIN     { Next element is the more probable }
         FrontRunner:= Current^.keynumber;                 { Update FrontRunner }
         Probability:= Current^.probability                { Update Probable }
      END;  { of IF }
   Current:= Current^.next
   END;  { of WHILE Current }
   FormFileName (FrontRunner, 'solution', FileName);
   assign (Disk, FileName);                          { Connect Disk with FrontRunner on disk file }
   ChDir ('solution');
   reset (Disk);
   writeln (' ':7,'The most likely solution is;');
   TextWindow (10,8,16,60);                          { Emphasise displayed solution text }
   TextFrame (true);
   Counter:= 10;
   WHILE NOT eof(Disk) DO BEGIN                      { Write solution text into screen box }
      readln (Disk, Line);
      GoToXY (12,Counter);
      writeln (Line)
      Counter:= Counter + 1
   END;  { of WHILE }
   close (Disk, true);
   ChDir ('\shell');
   writeln (' ':7,'The probability of this solution being correct is ',Probable:2,' percent.');
   writeln ('Press ''W'' for ''Why'' if you want to see an explanation of this solution.');
   writeln ('Press any other key to clear the screen and begin another session.');
   Gate:= ConSilent;
   IF (Gate = 'W') OR (Gate = 'w') THEN BEGIN
      FormFileName (FrontRunner,'explanation',FileName);
      FileName:= concat('\shell\solution\',FileName);
```

```
      IF fstat (FileName) = false THEN
         writeln ('No explanation for this solution is on file.')
      ELSE
         DisplayTextFile (FileName.'question','Explanation',FrontRunner,1);
   Gate:= Consilent;
   END;  { of IF }
END;  { of ELSE }
END;
BEGIN  END.
```

13.48 ClearHeap.

When the program has found the most probable solution, the heap will still contain all the remaining possible solutions. Upon leaving the program, in order to try another CORTEX analysis or to run another program, the solutions list should be removed from memory. This segment empties the heap.

13.48.1 Pseudocode.

```
go to head of linked list
WHILE NOT end of list
dispose of list element
exit to the segment FindResult
```

13.48.2 Draft Source Code.

```
SEGMENT  ClrHeap;
        { Remove the solutions linked list from memory. }
insert common types
PROCEDURE ClearHeap (VAR Head:solutionpointer);
VAR   Current, Succeeding: solutionpointer;
BEGIN
Current:= Head;
WHILE Current <> NIL DO BEGIN
   Succeeding:= Current^.next;
   dispose (Current);
   Current:= Succeeding;
END;  { of WHILE }
Head:= NIL
END;
BEGIN  END.
```

Chapter 14. IMPLEMENTATION OF CORTEX

An architectural design project will take on a different character according to the observer's point of view. To an historian it is a manifestation of its times, to a banker it is a money pump, to a building scientist it may be a thermal transfer network, a critic will see it in terms of stylistic trends, while to a conservationist it will probably be something fearful. An architect's own design can manifest itself to him in equally varied terms. It could be an attempt to explore a design idea in built form, perhaps merely a way of earning a living, or it may be the response to an inner compulsion to make visual sense of some corner of the built environment. The implementation that I propose to make of the Cortex shell starts from this last preoccupation.

A design will often begin with a diffused appreciation that something is amiss with someone's physical environment. A business cannot work effectively in its existing surrounding, a family is living less than well because its dwelling is ill-adjusted, or the public life of a community lacks a point of focus. An attempt to deal with the malaise usually begins as a dual process of gathering together information and starting to generate more information. Site and building surveys accumulate, regulations are appraised, client's needs and resources are assessed and drawings are made in an effort to summarise the steadily increasing mass of information. Other drawing, the early design sketches, begin the exploration of formal possibilities and themselves reveal the need to acquire more information. In this way the definition of the problem and the search for a solution proceed together and in a state of mutual support.

Architectural Precedent

The most pervasive type of design information, using that word in its general sense of 'that of which one is appraised', is knowledge of the work of other architects.

Every design is to some extent the result of the processing within the imagination of the architect of the buildings he has visited, seen illustrated, or heard described. Observation and assessment of the buildings of others goes on in the mind independently of a particular project, but it increases in intensity and specificity when a building design is in hand. Gathering information on examples of architecture and on architectural precedent is a part of every serious building design enterprise.

Most of this processing occurs subconsciously, and the results emerge in a way that is largely outside the control of the conscious mind. That is why Le Corbusier adopted the practice of letting a design task mature silently in his mind for some months before beginning to commit his ideas to paper. During that time the elements of the problem, including images of the buildings that he had studied over the years, would "float, simmer and ferment" in his imagination and lay the foundation for the emergence of the design of the building. This process is entirely non-monotonic, and can be regarded as the architect's response to the Rittellian 'wickedness' of design problems.

In the past an architect's knowledge of buildings that he has not himself visited was gained from books, magazines, and illustrations thrown onto auditorium or television screens by lecturers and film makers. The computer screen, however, has so far played little part in the visual enrichment of the architect's imagination. This is on account of the large data storage requirements of pictorial processing.

Because of the need to identify and specify each pixel, a single whole-screen colour image on a VGA monitor requires as much as 1mb of memory. Conventional magnetic storage media are rapidly overwhelmed by such large storage needs. But recently a new data storage medium has appeared in the form of optical disc technology. Because the data is read

my means of an extremely fine laser beam the tracks on an optical disc can be very close together, and the density of data storage is in consequence much higher than a floppy or a hard magnetic disc. A 12cm CD-ROM, for example, contains a track 20km long and can hold 525mb of data.

The high storage capacity of the optical disc makes it now possible for the architect to use the screen of a desk-top computer for pictorial, as opposed to graphical, purposes. Whole-screen full colour images, rather than simply vectorised line drawings, can be stored in large numbers on an optical disc and displayed on the screen of a modest size computer. The implementation of Cortex makes use of this technology for the purpose of describing the visual properties of the built environment.

Slide Libraries

Every school of architecture in Britain has a collection of 35mm slides of buildings, usually kept as part of its library stock. If the librarian or a member of staff is diligent the slide collection will be indexed and available for consultation or loan. The largest school slide library in Europe, that belonging to the Architectural Association, is the product of nearly a century of photography, collection and maintenance, The AA collection now consists of some 80,000 slides and it is an architectural visual resource of international importance. It contains photographs of nearly every significant twentieth-century building, together with views of most important European townscapes.

So large is the AA collection, however, that it is hard to exploit it fully. A whole morning spent with the index and slide viewer is not enough to ensure that all the photographs in the library which are relevant to a particular essay or lecture topic have been found. The inquirer is overwhelmed by the quantity of information which has been put together in the collection by so many photographers. In

fact, the collection is difficult to use at all unless the inquirer has a clear idea beforehand of what he is looking for.

No other school has so large a slide collection, and few have one that is as well-managed as that of the AA. However, most school slide libraries, despite their more modest size, are as difficult to use as is the AA collection. It is instructive to consider what are the problems which confront the user in his search for the right set of slides and what is the source of his difficulties.

The slides in the AA collection are indexed in three ways. Every slide can be found provided that the architect, the building type, or its date is known. A fourth 'subject' catalogue is kept on a card index. Each of the four indexes is structured hierarchically. The architect index is in the style of a telephone directory. An alphabetical order of surname is subdivided by first name. Dates are in simple chronological order, while the building type index is in alphabetical order subdivided according to country. Three presuppositions underlie the design of an indexing method of this type.

Firstly, the user is assumed to have a limited range of interest. No facility exists for finding all pictures showing brick domes, for instance, nor can examples of axially planned spaces be found. Only if author, building type, subject or date are known can the inquirer make progress, and no other information is useful. In the second place, the indexes are structured deductively. If the user begins a search with the name of a particular country then it is assumed that he can only be concerned with places within that country. This makes no provision for finding, perhaps, comparative examples of houses built on mountain slopes in both Spain and Peru or, alternatively, photographs of naturally ventilated buildings in several different desert climates. But an architect, when he is evolving

a design, does not think deductively. He will want to see what it looks like if two spaces interlock at the corners, for example, or how the circulation can work if a building is planned round courtyards. Furthermore, if no illustrations of a suitable courtyard circulation pattern can be found the area of interest may be shifted to buildings consisting of linked pavilions. In fact, the search pattern of a designer will evolve according to a non-monotonic principle of association rather than follow a logical deductive pathway.

The third presupposition is that the index system is passive. The initiative in the inquiry is provided by the user and the catalogue will yield, but not proffer, the information. Expert systems, which are by their nature interactive, are a type of artificial intelligence program which can overcome this limitation and provide the user of a slide library with intelligent access to its contents.

The Dublin Disc

The difficulty that users experience in using conventionally indexed collections of slides persuaded the slide librarian of the School of Architecture at University College, Dublin, to make an experiment using optical disc technology. About 10,000 slides from the school's collection of architectural photographs was copied onto a 30cm Philips Laservision optical disc during the autumn of 1985. The slides include images from all historical periods from the neolithic onward, and photographs of the important buildings of most European countries are recorded on the disc. (Hastings, 1986)

Only part of the capacity of a 30cm Laservision optical disc was taken up by images of buildings in the Dublin experiment. A single disc can contain as many as 54,000 images, which is more than two thirds the number of slides in the AA collection. One disc could hold more than five

collections the size of the Dublin slide library. So many images cannot be utilised effectively by means of conventional manual search methods.

A part of the Dublin experiment was to provide access to the contents of the disc through an IBM PC. The special software that was written to control the Laservision disc drive makes use of conventional database techniques. A search can be made on any of eight fields, and the usual help facilities are provided. However, the search method adopted at Dublin is only a small improvement on the manual index with which the user if faced at the AA. A computer keyboard is more convenient to use than a card index drawer, but the search method is the same.

The Questions

Since the act of formal creation occurs within the mind of the individual designer it follows that there are as many attitudes towards the design of buildings as there are architects. A computer system to access the images contained in a slide library, if it is to be useful to the architect in his role as a creator of form, needs to be responsive to this fact. It must make a selection of the material according to the point of view of the particular architect who is making the inquiry.

Appendix 1 of this thesis contains 47 quotations taken from the writings of architects, critics and historians which have appeared in print during the course of the last 130 years. They have been selected so as to express as wide a spectrum as possible of modern British and American attitudes towards architecture. Other approaches to the subject have characterised earlier periods of architecture, such as the Palladian or the Gothic revivals, but they are excluded from the list of questions because they are of antiquarian rather than operational interest to designers.

The area of architectural interest of the user is elucidated by asking him a selection of the quotations listed in Appendix 1. They are sent to the screen enclosed in quotation marks, but without title or attribution, and the user is asked if he agrees with the statement. His attitude towards his role as a designer of buildings is built up from the pattern of his answers.

An answer to a particular question will often render other questions irrelevant. For example, if the user concurs with the moral determinism of question 9 then it would make little sense to ask if he also agrees with the eclecticism of Charles Moore and Gerald Allen expressed in question 37. Conversely, the scope of the set of images to be displayed can be narrowed if the user agree with both the functionalism of question 17 and the industrialism of question 30. Cortex is written to respond to both of these circumstances.

It is possible for the user's answers to indicate an attitude that lies outside the understanding of the system. Should both questions 5 and 41 receive agreement, then the user will receive the answer that no suitable selection of slides can be made. An explanation to the effect that futurism and nostalgia are incompatible is available for sending to the screen in this case.

The Laservision Disc Reader

The equivalent of 12cm audio compact discs, known in the context of computer science as CD-ROM's, can be read by drives which are built into the chassis of a desk-top computer. Laservision discs, however, which are 30cm in diameter, require a separate disk drive. The drive is similar in size to the computer itself.

An analogue 30cm optical disc, such as that produced in Dublin, can be played on one of the six types of Philips Laservision disc drives. All six machines except the sim-

plest, known as the VP835, can be controlled by a desk-top computer. Larger models of Laservision disc drives include a CPU so that they can use application program read directly from the optical disc. However, the VP835B model which is used to implement Cortex contains no CPU and must be controlled by an external computer.

Output from the Laservision is in video format, and it therefore cannot be accepted by a computer monitor. Interface cards are available which are able to convert a video signal to digital so that a computer's own screen can display the image. The Cortex implementation, however, makes use of a separate video monitor to display output from the optical disk. Cortex output appears only on the computer screen. The installation is shown diagrammatically in Figure 14.1.

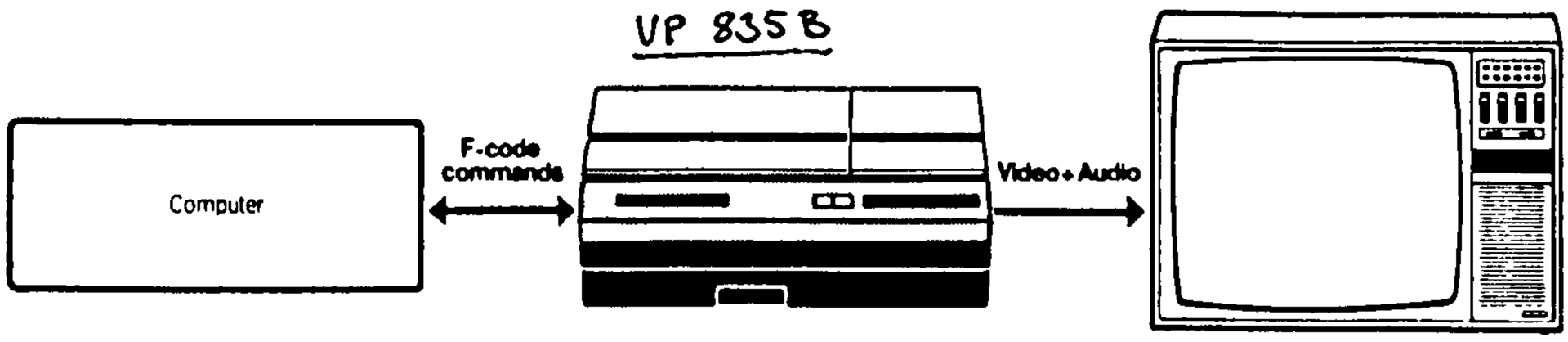

Figure 14.1
Cortex Laservision Installation

The VP835B is equipped with a built-in code, known as F-code, consisting of 73 commands for controlling the operation of the drive. The codes enable the computer to issue instructions to the disc drive to scan the disc forwards or backwards, to regulate the speed of scan, to select any frame, to group frames into chapters, to repeat a sequence of frames a specified number of time, to turn sound on and off, and to carry out a number of auxiliary operations such as beep, clearing the screen and ejecting the disc. For example, the command F1473R will bring frame 1473 to the

monitor screen and hold it there while Q18R will access the frames in chapter 18. In this implementation of Cortex the output of the program takes the form of F-code commands.

Cortex and the Dublin Disc

There is effectively no limit to the number of classifiers that can be written in a Cortex implementation. A classifier in Cortex is simply another record-type variable, upon which there is no limitation as to number. A separate classifier can therefore be written to represent the attitude to architecture that is revealed by any combination of answers to questions that a user may give. The first restriction upon search imposed by conventional indexing systems, that the user must be assumed to have a limited range of interests, is overcome in this implementation of Cortex. In a fully developed system as many attitudes as the knowledge engineer can think of can each be provided for.

Secondly, there is no necessity for one picture to be selected for display just because another has already been selected. There is no logical connection between the selection of one image and another, nor between the definitions of sets of images. The output of the system is determined by the associations of which the knowledge engineer is aware between the visual significance of the images that are recorded on the optical disc. An intimate and subtle knowledge of architecture and the process of design is needed to do this successfully, but no familiarity with logic or logic programming is called for. The knowledge engineer is free in Cortex to classify the images in a non-monotonic way that is appropriate to architectural design.

In the third place, the passive nature of traditional library systems is superseded in this implementation. A card index, or a computer search system such as GEAC will furnish the user with a reference provided that the correct information is supplied. The user must know ahead of time

what this information is, and the system has no ability to prompt him for the necessary input. A user cannot, for example, expect to succeed with a card index arrange according to architect if he inputs the name of a building, nor can the GEAC library system do more than confess failure under the same circumstance. But Cortex, like any developed expert system, will prompt the user to supply the information that is needed to complete a search successfully. The user must, in order to understand the significance of the images that are displayed on screen, bring to the system a knowledge of architecture and design but he can afford to be perfectly ignorant of the way that the computer search system functions.

An optical disc such as that produced by University College Dublin, because of its huge capacity, makes it possible for very large quantities of visual information to be stored and displayed. Cortex offers a way in which such large stores of pictures can be accessed in such a way as to be useful to the architect as designer and provider of building form. I think that a classificationist expert system and an optical disc of information can be brought together in a way that is useful to the architect when functioning in his central role as the designer of buildings and cities.

Chapter 15. CONCLUSION

In Chapter 1 of this text I attributed a predominant role to the leap of the imagination that is a necessary part of the creation of the design of a building. It is a point that has to be made because so many authors on design and computing assert otherwise, and attempt to show that design can sooner or later be represented by an algorithm.

> "Once we get used to the idea of applying rules to manipulate representations, it is easy to imagine designs and their descriptions generated in computations. The procedures followed to carry out these computations define and interpret languages of designs." (Stiny, 1985)

Similar notions have gained expression in Australia.

> "Meta-languages have been discussed as a way of formalising the complex mappings between meaning and artifact in design, such that designs can be generated that exhibit desired attributes. The rules of a grammar that operate on a vocabulary are considered as actions. We have considered how other grammars might operate on those actions. The assumption is made that designers are readily able to articulate such grammars and that they constitute a type of knowledge about design." (Coyne & Gero, 1986)

These two quotations are based upon the same assumption as that which underlay Wittgenstein's Tractatus, which is that meaning is a consequence of correct formal representation. They are similar to the claim that Wittgenstein makes for logic as the glue of semantics.

> "If we turn the constituent of a proposition into a variable, there is a class of propositions all of which are values of the resulting variable proposition. In general, this class too will be dependent on the meaning that our arbitrary conventions have given to parts of the original proposition. But if all the signs in it that have arbitrarily determined meanings are turned into variables, we shall still get a class of this kind. This one, however, is not

> dependent upon any convention, but solely on the nature of the proposition. It corresponds to a logical form - a logical prototype." (TLP 3.315)

I argue in Chapter 7 that the search for an effective procedure to represent the processes of thought must be abandoned for the same reason that Wittgenstein came to reject his notion, put forward in the Tractatus, that meaning is founded upon logic. The early assumptions of Herbert Simon, Allen Newell and Roger Schank about the algorithmic nature of thinking were too lightly entered into. Similarly, I think that any effort to underpin design with rules or meta-languages is vain. These ideas too, although many of the papers in which they appear date from the mid-1980's, belong to the infancy of artificial intelligence.

Graph Theory

The pictorial nature of graphs, using the term in its general sense of a diagram made up of points and lines, seems to place graph theory in close relationship to the graphic arts. There are indeed areas of overlap between the two. A map, for instance, may be regarded both as a terrestrial graph and as a work of graphic art. Furthermore, the words graph and graphic both derive from the Greek 'graphikos', meaning drawing or writing, which itself is ambiguous when translated into English. This intertwining of notions has lead commentators to sometimes confuse graph theory with graphical design. For example, an influential 27-page paper by Archer (1970) contains 59 graphs in support of an attempt to 'form the basis of a science of design.' It is necessary to disentangle the matted threads of this line of argument.

A graph can illuminate a logical argument. The puzzle of the bridges of Konigsberg proved to be amenable to graph theory analysis in Chapter 9 because the rules of the puzzle are fixed and the bridges across the Pregel do not

move. Consequently the solution to the puzzle can be deduced in a strictly logical manner from the premises. Graph theory is used to solve very large scale problems of this kind in electronic design, hydraulic systems and the construction of industrial plant. Success in these areas of activity make it superficially attractive to apply graph theory to conceptual activities such as design. Are not all graphs fundamentally the same? But argument by analogy is notoriously unreliable, because it is based upon a supposition that since the two terms of a comparison are alike in some respects they will be similar in others. But to do this begs the question, and inference by analogy is logically unsound. Much design theory rests upon this weak conceptual foundation.

The analogy breaks down when the logical character of graph theory is compared with the non-monotonic nature of design. The most conspicuous feature of the graph of the design process is the presence of a feedback loop leading from every vertex to every preceding vertex. Two things are achieved by this representation of feedback and repetition in design. Firstly, the true nature of the problem, which initially defied description, is elucidated. Second, a solution to that problem is revealed. That is the sense in which Dennis Lasdun observed that;

> "Handbooks will tell you that the job of an architect is to give the client what he wants. That is not your job or mine. Our job is to give the client, on time and on cost, not what he wants but what he never dreamed he wanted and, when he gets it, he recognises it as some thing he wanted all the time." (Lasdun, 1965)

That is to say, it is only when a design is completed that one is able to see what the problem was. This essential point, central to the enterprise of design, is generally ignored in design studies texts. It is often asserted, for example, that design is in some sense goal-directed or concerned with problem-solving.

> "The activity of designing is thus a goal-directed activity and normally a goal-directed problem-solving activity. The properties which are required to be exhibited by the proposed artifact are defined by the objectives of the problem. The details of the design are the designer's conclusions as to the means by which those properties may be provided." (Archer, 1970)

Graph theory could no doubt elucidate the activity of design if design were logical and goal-directed. However, design is in reality non-monotonic, iterative and exploratory. I think that the kind of misunderstanding of design which is exhibited by Bruce Archer in this quotation is the main reason why design studies have yet to exert any influence on the way that designers actually work.

A special type of self-adjusting graph derived from Emil Post's work on combinatorics has proved to be useful in artificial intelligence. Rule-based expert systems make use of the formalism of a production system to trace a path from a description of the problem environment to a solution. But production systems too are deductive, and they are therefore a poor representation of the process of design.

The Cortex Shell

In this thesis I have drawn a distinction between the necessarily logical functioning of computer hardware and the ability of a computer program to work in a non-monotonic fashion. It is possible to distinguish between the two aspects of computing because the workings of the machine itself cannot be altered by the user - a decision statement will always make a specified comparison, for example - whereas the output of the computer is susceptible to interpretation by a human mind. A logical result, such as an arithmetical solution, is no more or less valuable than a descriptive, diagnostic or exploratory program output. The type of output likely to be useful depends upon the needs

and viewpoint of the user of the system. Different minds will interpret the output symbols according to different criteria.

The principle upon which the expert system shell Cortex is based is that of classification. A method or system is required if a classification is to be made, but the method may or may not be logical in character. Hierarchical classifications are logical, in that items are reached by progression through sub-classes, sub-sub-classes and so on until the required fineness of definition is achieved. A small part of the Library of Congress catalogue reads;

```
Architectural design
     Design, Architectural
     Design
     Structural design
     Architectural drawing
     Architecture
          Composition, proportion, etc
          Details
     Communication in architectural design
     Crime prevention and architectural design
     Decoration and ornament, Architectural
     Data processing
          Architecture, computers in
```

Other classifications, such as the set of flowers growing in an herbaceous border, for example, or the constellations by which the fixed stars are identified, are based upon pictorial or associational rather than logically related ideas. The same could be said of the contents of the front page of a daily newspaper, the program of an orchestral concert, and not less important, the influences that shape the design of a building. Ideas and notions are associated in these cultural spheres in accordance with a real or possible human point of view rather than the abstract definitions of meaning which feature in logical systems such as library indexes.

In Cortex the classification of one item with respect to another is made when the classifiers are set up. The ele-

ments of a classifier are binary bits, and the information which can be conveyed by each bit is strictly Boolean. The fact, concept or assertion represented by a bit in a classifier is labelled true of false, but nothing is said about its relation with any other bit in its own or any other classifier. The classification which is achieved in any implementation of Cortex may therefore be deductive or not according to the circumstances of the case. In an implementation concerned perhaps with one of the technical aspects of architecture the domain could be represented in a strictly logical manner. However, in the implementation of the Dublin optical disc described in chapter 14 the non-monotonic properties of Cortex are exploited. This feature of the implementation corresponds to the non-monotonic nature of architectural design.

Further Research

In Chapter 8 I concluded the section on intelligent tutoring systems with a brief note about the line of research undertaken in recent years by the authors of SOPHIE. The qualitative physics that they have evolved is intended to provide a bridge between the external physical world and the inner mental world of meaning. But I believe that no formal system can bridge that gap. The proposal made by de Kleer that general terms such as equilibrium, oscillation or feedback can provided the bridge is vain, for qualitative physics only pushes the gulf between things and thought backwards by one step in the argument. When qualitative physics is in place, the question remains as to what do its general terms mean. This, like all such questions, can only be answered with reference to a point of view, or in terms of what Wittgenstein calls a language game. For the lack of an observing consciousness, the regress of meaning in qualitative physics becomes infinite.

I believe that a more promising line of research would be to implement the method of control that I have used in Cortex in the design of an ITS. In Cortex only a single

classification of solutions is necessary and it can be compared directly to the pattern of the user's answers. An ITS would necessitate an intermediate stage, in which selection is made from a set of mental models of the student and from operational models of the domain. These procedures would use the same selection algorithm as is employed in Cortex. A bit string recording the result of these selections would then be compared with a classifier representing the available advice in order to choose the next screen display. An ITS designed in this way would have the advantages that I claim for Cortex, of being very fast in operation and being applicable to any domain.

Blackboards were invented at Carnegie-Mellon University in the 1970's as part of an effort to program a computer to understand speech. The idea was that a central management procedure, or blackboard, would select from a number of software tools, or knowledge sources, according to their usefulness and applicability in solving part of a complex problem. Acting in unison under the direction of the blackboard, the knowledge sources would in combination find a complete solution to the whole problem. The blackboard of Hearsay-II, for example, could call on knowledge sources developed for;

> "extracting acoustic parameters, classifying acoustic segments into phonetic classes, recognising words, parsing phrases, and generating and evaluating predictions for undetected words of syllables." (Erman et al, 1980)

The problem of control in blackboard systems is similar to that which is encountered in the design of expert systems and intelligent tutoring systems.

> "The knowledge sources respond opportunistically to changes on the blackboard. There is a set of control modules that monitor the changes on the blackboard and decide what actions to take next. Various kinds of information are made globally available to the control modules. The information can be on the blackboard or kept

> separately. The control information is used by the control modules to determine the focus of attention." (Englemore, Morgan & Nii, 1988)

No one method of control has been found to be superior to all others in blackboard programs. However, all make use of information and thus are domain specific.

The most general control method for blackboard systems that has been devised so far is that proposed by Hayes-Roth (1985). The core of her program is a scheduling mechanism that maintains a rating for each knowledge source, and calculates a score according to the state of the system. When a score and a rating coincide the particular knowledge source is triggered. When the action of the knowledge source is finished the cycle is repeated.

This procedure is a more developed versions of the scoring system devised by Frey for his House.Bas program. An account of the Frey algorithm is given in Chapter 11. In both the Hayes-Roth and the Frey programs an action is taken as a result of a score accumulating to exceed a threshold value. But as in House.Bas, the scoring Hayes-Roth algorithm is arithmetical and, at root, arbitrary. In the example given in her paper, for example, one knowledge source is assigned a credibility of 1.0, and another of 0.8. No explanation of these figures is supplied, for the reason that they are the result of judgements about the usefulness of the particular tool in the domain under consideration.

In recent years the notion of a blackboard has been expanded to include any program whose purpose is to utilise several different sub-programs in a co-ordinated way so as to find the answer to large problems. I think that the Cortex algorithm, in which control is logical rather than arithmetic in character, could be developed to manage a blackboard system in a more efficient and consistent way than is possible with present methods.

The version of the Cortex shell that is presented in this thesis is developed only to the stage of a prototype. The prototype demonstrates that the algorithm is functional, that the capacity of the system is large and that it works very fast. Furthermore, it is an example of a department of artificial intelligence that can be of use to the architect. The system contains 512 questions, only 47 of which are linked to question files, and it can accept an effectively unlimited number of solutions. Processing takes place so quickly that no pause is visible to the eye.

However, the input and output routines are underdeveloped, and the screen displays are graphically poor. Cortex can readily accommodate windows and icons, and the system would be much more convenient to use with a mouse than it is with a keyboard. Cortex in the form of a developed product would be a convenient as well as a powerful system, and appropriate implementations of Cortex would be fast and responsive assistants to architects and other designers.

Appendix 1. TEXT OF THE QUESTIONS

In the optical disk implementation of Cortex the following questions are sent to the screen for answering by the user. The questions are presented to the user in quotation marks, but without title or attribution.

Question 1.

Truth
"In architecture there are two necessary ways of being true. It must be true according to the programme and true according to the methods of con struction. To be true according to the program is to fulfil exactly and simply the conditions imposed by need: to be true according to the methods of construction is to employ the materials according to their qualities and properties.... purely artistic questions of symmetry and apparent form are only secondary conditions in the presence of our dominant principles." (Viollet-le-Duc, 1863)

Question 2.

Arts & Crafts
"architectural beauty is the result of the harmonious and intelligent co-operation of the whole body of people engaged in producing the work of the workman." (Morris, 1893)

Question 3.

Nationalism
"To a casual observer, the interest we feel in the subject may appear to be the result of local prejudice or local choice, and our national style may seem to have no greater claim upon us than the style of a hundred other periods or countries. The fact, however, is the reverse - that the style is marked out from that of other countries in the most signal and remarkable manner... The character of a style of art does not depend upon the mere material from which it has been fabricated, but upon the sentiments and conditions under which it has been developed." (Mackintosh, 1893)

Question 4.

Ornamentation
"I have made the following discovery and I pass it on to the world: The evolution of culture is synonymous with the removal of ornament from utilitarian objects. I believed that with this discovery I was bringing joy to the world; it has not thanked me. People were sad and hung their heads. What depressed them was the realisation that they could produce no new ornaments." (Loos, 1908)

Question 5.

Futurism
"The tremendous antithesis between the modern and the ancient world is the outcome of all those things that exist now and did not exist then. Elements have entered into our life of whose very possibility the ancients did not even dream. Material possibilities and attitudes of mind have come into being that have had a thousand repercussions, first and foremost of which is the creation of a new ideal of beauty, still obscure and embryonic, but whose fascination is already being felt by the masses. We have lost the sense of the monumental, of the heavy, of the static; we have enriched our sensibility by a 'taste for the light, the practical, the ephemeral and the swift'. We feel that we are no longer the men of the cathedrals, the palaces, the assembly halls; but of big hotels, railway stations, immense roads, colossal ports, covered markets, brilliantly lit galleries, freeways, demolition and rebuilding schemes." (Sant'Elia, 1914)

Question 6.

Humanism
"by the same excellent - because unconscious - testimony of speech, arches 'spring', vistas 'stretch, domes 'swell', Greek temples are 'calm' and baroque facades 'restless'. The whole of architecture is, in fact, unconsciously invested by us with human movement and human moods. Here, then, is a principle complementary to the one just stated. We transcribe architecture into terms of ourselves." (Scott, 1914)

Question 7.

Standardisation
"Standardisation, to be understood as the result of beneficial concentration, will alone make possible the development of a universally valid, unfailing good taste." (Muthesius & Van de Velde, 1914)

Question 8.

Glass Architecture
"The surface of the earth would change greatly if brick architecture were everywhere displaced by glass architecture. It would be as though the Earth clad itself in jewellry of brilliants and enamel. The splendour is absolutely unimaginable. And we should have on the Earth more exquisite things than the gardens of the Arabian Nights. Then we should have a paradise on Earth and would not need to gaze longingly at the paradise in the sky." (Scheerbart, 1914)

Question 9.

Moral Determinism
"No one can doubt the educative value of visible beauty; therefore it would revolutionise our lives, if in all we produced and made we recognised the necessity of conveying some common moral sentiments." (Voysey, 1915)

Question 10.

Stripped Classicism
"if our structure remains simple, unadorned, without moulding, bare, we are then best able to arrange the decorative arts so that each object of art will retain its purest and clearest expression because it will be totally independent of the construction. Besides, who would not see that the use of such materials results in the obtaining of the horizontals and verticals that are proper to give to the construction that calm and equilibrium that will harmonise with the lines of nature?" (Garnier, 1917)

Question 11.

Expressionism
"Tell me what love is, what faith, and the iron will of hope - and I will tell you what it means to build: to bring the seventh day of creation one wave further in the tidal chain that lovingly toys with eternity. There is no greater Affirmer than the true builder. Everything about him is expansion, pressing outwards - the more rhythmical, harmonious and healthy the pulse of his soul, the more perfect and inimitable will be the superstructure he sets upon the world's countenance, like a victorious seal upon his existence." (Finsterlin, 1920)

Question 12.

Purism
"Architecture is the masterly, correct and magnificent display of masses brought together in light. Our eyes are made to see forms in light; light and shade reveal those forms; primary forms which light reveals to advantage; the image of these is distinct and tangible within us and without ambiguity. It is for that reason that these are beautiful forms, the most beautiful forms. Everybody is agreed as to that, the child, the savage and the metaphysician. It is of the very nature of the plastic arts." (le Corbusier, 1920)

Question 13.

Ville Radieuse
"Their outlines softened by distance, the skyscrapers raise immense geometrical facades all of glass, and in them is reflected the blue glory of the sky. An overwhelming sensation. Immense but radiant prisms. This is no dangerous futurism, a sort of literary dynamite flung violently at the spectator. It is a spectacle organised by an Architecture which has plastic resources for the modulation of forms seen in light." (le Corbusier, 1924)

Question 14.

Form Follows Function
"This meant in his courageous mind that he would put to the test a formula he had evolved, through long contemplation of living things, namely that form follows function, which would mean, in practice, that architecture might again become a living art, if this formula were but adhered to." (Sullivan, 1924)

Question 15.

Constructivism
"the machine naturally gives rise to a conception of entirely new and modern organisms possessing the distinctly expressed characteristics of movement - its tensions and intensity, as well as keenly expressed direction. Both of these characteristics give rise to concepts of new forms, whereby the tension and concentration inherent in this movement will unwittingly - irrespective of the author's own desires - become one of the fundamental moments of artistic conception." (Ginzberg, 1924)

Question 16.

The Bauhaus
"It is only through contact with newly evolving techniques, with the discovery of new materials and with new ways of putting things together, that the creative individual can learn to bring the design of objects into a living relationship with tradition and from that point to develop a new attitude towards design, which is:
a resolute affirmation of the living environment of machines and vehicles;
the organic design of things based on their own present-day laws, without gloss and wasteful frivolity;
the limitation to characteristic, primary forms and colours, readily accessible to everyone;
simplicity in multiplicity, economical use of space, material, time and money.
The creation of standard types for all practical commodities of everyday use is a social necessity."
(Gropius, 1926)

Question 17.

Functionalism
"Architecture as 'an emotional act of the artist' has no justification. Architecture as 'a continuation of the traditions of building' means being carried along by the history of architecture. This functional, biological interpretation of architecture as giving shape to the functions of life, logically leads to pure construction: this world of constructive forms knows no native country. It is the expression of an international attitude in architecture. Internationality is a privilege of the period. Pure construction is the basis and characteristic of the new world of forms." (Meyer, 1928)

Question 18.

Zeitgeist
"building is an elementary activity of man intimately linked with evolution and the development of human life. The destiny of architecture is to express the orientation of the age. Works of architecture can spring only from the present time." (CIAM, 1928)

Question 19.

Vitalism
"Consider well that a house is a machine in which to live but architecture begins where that concept of the house ends. All life is machinery in a rudimentary sense, and yet machinery is the life of nothing. Machinery is machinery only because of life. It is better for you to proceed from the generals to the particulars. So do not rationalise from machinery to life. Why not think from life to machines? The utensil, the weapon, the automaton - all are appliances. The song, the masterpiece, the edifice are a warm outpouring of the heart of man - human delight in life triumphant: we glimpse the infinite." (Wright, 1931)

Question 20.

Organic Architecture
"There remains an essential difference between the architect and the engineer. The work of the engineer has as its goal merely the performance of material work within the limits or in the domain of economic effects. That the result frequently contains other expressive values as well is a side-effect, a subsidiary phenomenon of his work. The architect, on the other hand, creates a Gestalt, a total form of work of spiritual vitality and fulfilment, an object that belongs to and serves an idea, a higher culture. This work begins where the engineer, the technologist leaves off: it begins when the work is given life. Life is not given to the work by fashioning the object, the building, according to a viewpoint alien to it, but by awakening, fostering, and cultivating the essential form enclosed within it." (Haring, 1932)

Question 21.

Priority of Space
"The effect of mass, of static solidity, hitherto the prime quality of architecture, has all but disappeared: in its place there is an effect of volume, or more accurately, of plane surfaces bounding a volume. The prime architectural symbol is no longer the dense brick, but the open box. Indeed, the great majority of buildings are in reality, as well as effect, mere planes surrounding a volume. With skeleton construction enveloped only by a protective screen, the architect can hardly avoid achieving this effect of surface, of volume, unless in deference to traditional design in terms of mass he goes out of his way to achieve the contrary effect." (Hitchcock & Johnson, 1932)

Question 22.

Broadacre City
"This tract of four miles square, by way of such liberal general allotment determined by acreage and type of ground, including apartment buildings and hotel facilities, provides for about 1400 families at, say, an average of five or more persons to the family. To reiterate: the basis of the whole is a general decentralization as an applied principle and architectural reintegration of all units into one fabric; free use of the ground held only by use and improvements; public utilities and government itself owned by the people of Broadacre City; privacy on one's own ground for all and a fair means of subsistence for all by way of their own work on their own ground or in their own laboratory or in common offices serving the life of the whole." (Wright, 1935)

Question 23.

Humanised Modernism
"The term 'rationalism' appears in connection with Modern architecture about as often as does 'functionalism'. Modern architecture has been rationalised mainly from the technical point of view, in the same way as the technical functions have been emphasised. Although the purely rational period of Modern architecture has created constructions where rationalised technique has been exaggerated and the human functions have not been emphasised enough, this is not a reason to fight rationalism in architecture. It is not the rationalisation itself which was wrong in the first and now past period of Modern architecture. The wrongness lies in the fact that the rationalisation has not gone deep enough. In-

stead of fighting rational mentality, the newest phase of Modern architecture tries to project rational methods from the technical field out to human and psychological fields." (Aalto, 1940)

Question 24.

Biological Analogy
"It has become imperative that in designing our physical environment we should consciously raise the fundamental question of survival, in the broadest sense of this term. Any design that impairs and imposes excessive strain on the natural human equipment should be eliminated, or modified in accordance with the requirements of our nervous and, more generally, our total physiological functioning. This principle is our only operational criterion in judging design or any detail of man-made environment, regardless of how difficult it may seem to apply the principle in specific cases." (Neutra, 1954)

Question 25.

Anarchism
"No inhibitions should be placed upon the individual's desire to build! Everyone ought to be able and compelled to build, so that he bears real responsibility for the four walls in which he lives. We must face the risk that a crazy structure of this kind may later collapse, and we should not and must not shrink from the loss of life which this new way of building will, or at least may, exact. A stop must finally be put to the situation in which people move into their living quarters like hens and rabbits into their coops." (Hundertwasser, 1958)

Question 26.

Monumentalisation of Technique
"I believe that in building you must deal with con struction directly. You must, therefore, understand construction. When you refine the structure and when it becomes an expression of our time, it will then and only then become architecture." (van der Rohe, 1960)

Question 27.

Order
"Design is form-making in order
Form emerges out of a system of construction
Growth is a construction
In order is creative force
In design is the means - where with what when with how much
The nature of space reflects what it wants to be
Is the auditorium a Stradivarius
or an ear
Is the auditorium a creative instrument
keyed to Bach or Bartok
played by the conductor
or is it a conventional hall
In the nature of space is the spirit and the will to exist in a certain way."
(Kahn, 1960)

Question 28.

Scientism
"Man has been blindly flying into his future on scientific instruments and formulas. The great news on the artist-scientist-intellectual frontier is that as the fog-and-black shadow of ignorance and misconception recedes, there looms a sublimely comprehensible conceptual patterning, which characterizes all mathematical principles heretofore only formulatively employed by the scientist, yet intuitively pursued by the artist as potentially modelable. Experimental science has validated the artist's intuitions but not his disciplines." (Fuller, 1960)

Question 29.

Place
"Man may readily identify himself with his own hearth, but not easily with the town within which it is placed. 'Belonging' is a basic human need - its associations are of the simplest order. From 'belonging' - identity - comes the enriching sense of neighbourliness. The short narrow street of the slum succeeds where spacious redevelopment frequently fails." (Newman, 1961)

Question 30.

Industrialism
"There can be no doubt that the first prerequisite for a good building has always been the best tools and the best methods, and it is in industrialisation that this condition is best fulfilled. For industri-

alisation brings within our reach a level of technical accuracy, quality and precision never before attained in the history of building. Industry, not the individual and nor craftsmanship, determines what can be achieved and thus establishes the boundaries of the possible." (Wachsmann, 1961)

Question 31.

Contextualism
"The main justification for honky-tonk elements in architectural order is their very existence. They are what we have. Architects can bemoan or try to ignore them or even try to abolish them, but they will not go away. Or they will not go away for a long time, because architects do not have the power to replace them (nor do they know what to replace them with), and because these commonplace elements accommodate existing needs for variety and communication. The old cliches involving both banality and mess will still be the context of our new architecture, and our new architecture will significantly will be the context for them." (Venturi, 1966)

Question 32.

Archigram
"Architecture will become infinite and transient. At last the dividing line between the things which we carry round in the palm of the hand and the whole city will merge together as parts of the hierarchy of designed, phased, chosen objects; to suit the condition and requirements of the time they will be able to be changed for something better." (Cook, 1967)

Question 33.

Visionary Architecture
"As architecture grows into a phenomenon of human ecology, the cities will become organisms reflecting in their structural complexity the complexity of the life they contain. Upgrading from aggregation to organisation signals the end of present-day architecture and the concept of individual structures... Life itself will be the servant of a rational protohuman world. If aesthetic man measures the weight of his burden, disassociates himself from the whimsical and the fashionable, he will conceive the cradles of future cultures and be responsible for the advent of the ultrarational world." (Soleri, 1971)

Question 34.

Conservation
"The architectural heritage is a capital of irreplaceable spiritual, cultural, social and economic value...This capital has been built up over the centuries: the destruction of any part of it leaves us poorer, since nothing new that we create, however fine, will make good the loss." (Committee of Ministers, 1975)

Question 35.

Urbanism
"The debate.... is that of urban morphology as against the zoning of planners. The restoration of precise forms of urban space as against the wasteland which is created by zoning. The design of urban spaces, both traffic and pedestrian, linear and focal is, on the one hand, a method which is general enough to allow flexibility and change and, on the other, precise enough to create both spatial and built continuity within the city." (Krier, 1975)

Question 36.

Typology
"The design process is a way of bringing the elements of a typology - the idea of a formal structure - into the precise state that characterises the single work."
(Moneo, 1976)

Question 37.

Eclecticism
"All of us every day face the onslaught of experiences which require varied, complex, and agile responses. This is to say that we inhabit a pluralist world, and that we ourselves are many-faceted creatures. Thus no single orthodoxy - including the single-minded return to copying buildings from the past - will do, no single set of forms and images to shape the environment we build for ourselves. The meaning of buildings like those around Rockefeller Plaza and the new ones along the Avenue of the Americas is that architecture can have many potent likenesses. The choice is altogether ours, and the task is to learn to cast our nets backwards in time - and outwards - to find what feels right for a given design problem, and what among the many options seems really worthwhile." (Moore & Allen, 1976)

Question 38.

Small is Beautiful
"Town planning needs to shift emphasis towards encouraging and managing the kinds of small scale change likely to be most relevant, the detailed planning of small sites and of such local environmental schemes. In other words, there needs to be a new and more sensitive style of working geared to the management of small scale urban change." (Shankland et al, 1977)

Question 39.

Zen of Architecture
"Architects sometimes say that in order to design a building, you must have 'an image' to start with, so as to give coherence and order to the whole. But you can never create a natural thing in this state of mind. If you have an idea - and try to add the patterns to it, the idea controls, distorts, makes artificial, the work which the patterns are trying to do in your mind. Instead you must start with nothing in your mind." (Alexander, 1979)

Question 40.

Semiotics
"Every human society communicates architectonically. The component units of an architectonic code or system consist of contrastively-opposed formations in media addressed to visual perception. Distinctions or disjunctions in material formation are intended to cue culture-specific difference in meaning in a manner precisely analogous to other semiotic systems such as verbal language or bodily gesturing. In the broadest sense, communication consists of the transmission of information regarding the perception of similarities and differences. The system of the built environment, like any semiotic code, is a complexly-ordered device for the cuing of such perceptions." (Preziosi, 1979)

Question 41.

Nostalgia
"For nostalgia is a strange and enigmatic longing for that which escapes reasoning but survives profoundly and forcefully in the feelings of the citizens. They themselves have never lost their taste for classical architecture and, despite all the efforts of the media to convince them of the contrary, they have never longed for the Bauhaus boxes

or other experiments of the building industry." (Krier, 1981)

Question 42.

Ritual
"Today if I were to talk about architecture, I would say that it is a ritual rather than a creative process. I say this fully understanding the bitterness and the comfort of the ritual." (Rossi, 1981)

Question 43.

Post Modernism
"architecture is doubly-coded, one half Modern and one half something else (usually traditional building) in its attempt to communicate with the public and a concerned minority, usually other architects... treating the city as an historical artifact, in stressing metaphor, complexity, symbolism, irony and a host of rhetorical means." (Jencks, 1983)

Question 44.

Classicism
"There are practically no ugly Georgian houses. I live in a Georgian house myself and I notice that most architects choose to live in them too. The estate agent will attach the word 'Georgian' or 'Queen Anne' to a house he is trying to sell because he knows it is very near everyone's dream. And why is this? It is because these houses have the right balance of window to wall, they give permanent protection from wind and weather, they are comfortable and beautiful, and they seem perfectly suited to man. And whether one is elderly and poor in a terraced almshouse, or noble and great in a mansion, the needs and aspirations of humanity are expressed in classical terms through bricks and stone." (Terry, 1983)

Question 45.

The Skyscraper
"the skyscraper - in terms of size, structure, and function, scale and symbolism, and, above all, human and urban impact - remains the single most challenging design problem of our time... The twentieth century architect's most telling and lasting response to his age is the topless tower of trade." (Huxtable, 1984)

Question 46.

Behavioural Determinism

"we can deduce that the various forms of social breakdown tend to occur in a set order as design features worsen to the successive degrees of depravity needed to undermine each social taboo in turn. The effect is a general one... and it is not a question of different designs being responsible for different kinds of behavioural lapse. Design variables appear to exert the same kind of demoralising influence, and the values within each variable affect the degree of demoralisation." (Coleman, 1985)

Question 47.

Commercialism

"Any client, whether his building is a museum or a hotel, surely wants to employ a designer who will provide high quality and efficient design fused into a building which is completed to budget and to programme. In short, he wants a friendly commercial service provided quickly and effectively, not patronising arrogance, selectively dispensed." (Wheatley, 1990)

Appendix 2. LISTING OF CORTEX

```
{
{ Common1.Typ}
{ Common types for use with CORTEX. }


CONST                       MaxNumberOfQuestions = 512;
                            MaxNumberOfIntegers = 16;      { Prospero integers are 32 bit (16x32 = 512) }

TYPE          bitstring = ARRAY [1..MaxNumberOfIntegers] OF integer;   { type of Message and
                                                                         MessageMask }

  classifiertype = RECORD
                    keynumber: integer;                  { type of Classifier in SetUpSolutionList }
                    essentialmask: bitstring;
                    usualmask: bitstring;
                    classifier: bitstring;
                    totalusual: integer
                    END;

solutionpointer = ^solutiontype;                         { type of Classif disk file }

    solutiontype = RECORD                                { type of elements of solution list }
                    keynumber: integer;
                    essentialmask: bitstring;
                    usualmask: bitstring;
                    totalusual: real;
                    usualtrue: real;
                    probability: real;
                    classifier: bitstring;
                    next: solutionpointer
                    END;

questiontype = RECORD                                    { type of MostFrequentQuestion }
                    questionnumber: integer;
                    countofoccurrance: integer
                    END;
```

```
{
PROGRAM     Cortex;
             { Main program. }


{$I Common1.Typ}
PROCEDURE   SignOn (VAR CountOfQuestions, CountOfSolutions: integer); EXTERNAL;
PROCEDURE   ImplementationMessage (CountOfQuestions, CountOfSolutions: integer); EXTERNAL;
PROCEDURE   FirstMenu (CountOfQuestions, CountOfSolutions: integer); EXTERNAL;

VAR         CountOfQuestions: integer;
             CountOfSolutions: integer;

BEGIN

SignOn (CountOfQuestions, CountOfSolutions);
ImplementationMessage (CountOfQuestions, CountOfSolutions);
FirstMenu (CountOfQuestions, CountOfSolutions);

END.
```

```
{
SEGMENT     SignOn;

{$I \PROPAS\PASPC}
{$I \PROPAS\PASDOS}
PROCEDURE   Blankln (number: integer); EXTERNAL;
PROCEDURE   PressKey (margin: integer); EXTERNAL;
PROCEDURE   Initialise (VAR CountOfQuestions, CountOfSolutions: integer); EXTERNAL;

PROCEDURE   SignOn (VAR CountOfQuestions, CountOfSolutions: integer);
      VAR   Greeting: text;
             Gate: char;

BEGIN

InitScreen;
Paper (7);
Ink (1);
ClrScr;
TextFrame (true);
ScreenFile (Greeting);
CursorOff;
GoToXY (31,6);
writeln (Greeting,'*****************');
GoToXY (31,7);
writeln (Greeting,'Welcome to CORTEX');
GoToXY (31,8);
writeln (Greeting,'*****************');
writeln;
Ink (9);
GoToXY (23,10);
writeln (Greeting,'The thinking man''s expert system.');
GoToXY (7,22);
writeln (Greeting,'Please wait while the solutions and questions on file are counted.');
Initialise (CountOfQuestions, CountOfSolutions);
GoToXY (7,22);
PutChattr (' ',7,9,66);
GoToXY (23,22);
writeln (Greeting,'Press any key to continue.');
Gate:= ConSilent;
CursorOn;
InitScreen

END;

BEGIN  END.
```

```
{
SEGMENT     Initial;
             { Count questions and solutions on file. }

{$I Common1.Typ}
{$I \PROPAS\PASPC}
{$I \PROPAS\PASDOS}
PROCEDURE   FormFileName (filenumber: integer; title: string; VAR FileName: string); EXTERNAL;
PROCEDURE   ZeroiseBitString (VAR Bits: bitstring); EXTERNAL;

PROCEDURE   Initialise (VAR CountOfQuestions, CountOfSolutions: integer);
VAR         FileName: string;
             QuestionOnFile, SolutionOnFile: string;

BEGIN

MkDir ('\shell\question');
ChDir('\shell\question');                                    { Calculate CountOfQuestions }
CountOfQuestions:= 0;
FindFile ('quest1',QuestionOnFile);                          { Is there a quest1? }

WHILE QuestionOnFile <> '' DO BEGIN                        { If so, count through the questions }
     CountOfQuestions:= CountOfQuestions + 1;
     FormFileName ((CountOfQuestions + 1),'question',FileName); { Is the next question on file? }
     FindFile (FileName, QuestionOnFile);   { Returns QuestionOnFile empty if no question found }
END;  { of WHILE }

MkDir ('\shell\solution');
ChDir ('\shell\solution');                                   { Calculate CountOfSolutions }
CountOfSolutions:= 0;
FindFile ('solut1',SolutionOnFile);                          { Is there a solut1? }

WHILE SolutionOnFile <> '' DO BEGIN                      { If so, count through the solutions }
   CountOfSolutions:= CountOfSolutions + 1;
   FormFileName ((CountOfSolutions + 1),'solution',FileName);  { Is the next solution on file? }
   FindFile (FileName, SolutionOnFile);  { Returns SolutionOnFile as empty if no solution found }
END;  { of WHILE }

ChDir ('\shell');

END;

BEGIN  END.
```

```
{
SEGMENT     Impment;
             { Display implementation information. }

{$I \PROPAS\PASPC}
PROCEDURE   Blankln (number: integer); EXTERNAL;
PROCEDURE   PressKey (margin: integer); EXTERNAL;

PROCEDURE   ImplementationMessage (CountOfQuestions, CountOfSolutions: integer);
      VAR   Disk: text;
             Line: string[100];
             Counter: integer;

BEGIN

ClrScr;
IF fstat ('\Shell\Implemnt\Message') = true THEN BEGIN
   Blankln (2);
   writeln (' ':21,'Implementation Information');
   writeln (' ':21,'*****************************');
   Blankln (3);
   assign (Disk, '\Shell\Implemnt\Message');
   reset (Disk);
   Counter:= 8;

   WHILE NOT eof(Disk) DO BEGIN
      readln (Disk, Line);                          { Copy a line from Disk to Line }
      GoToXY (10, Counter);                          { Move cursor to starting position of text }
      writeln (Line);                               { Send contents of Line to screen }
      Counter:= Counter + 1                         { Move to next screen line }
   END;  { of WHILE }

   close (Disk, true);
   Blankln (3);
   writeln (' ':10,'There are ',CountOfQuestions:3,' questions and ',CountOfSolutions:3,'
            solutions on file.');
END  { of IF }
ELSE BEGIN
   Blankln (8);
   writeln (' ':12,'No implementation file has been written.');
END;  { of ELSE }

Blankln (2);
PressKey (10);

END;

BEGIN  END.
```

```
{
SEGMENT     FrstMenu;
             { Chooses between work on the knowledge base or use of the implemented system. }

{$I \PROPAS\PASPC}
PROCEDURE   Blankln (number: integer); EXTERNAL;
PROCEDURE   MenuError (range: integer); EXTERNAL;
PROCEDURE   KnowledgeBaseMenu (VAR CountOfQuestions, CountOfSolutions: integer); EXTERNAL;
PROCEDURE   FindResult (CountOfQuestions, CountOfSolutions: integer); EXTERNAL;

PROCEDURE   FirstMenu (CountOfQuestions, CountOfSolutions: integer);
      VAR   Flag: boolean;
            Selector: integer;

BEGIN

Flag:= true;

WHILE Flag = true DO BEGIN
   ClrScr;
   Blankln (6);
   writeln (' ':10,'Do you want to;');
   writeln;
   writeln (' ':13,'1. Use CORTEX?');
   writeln (' ':13,'2. Work on the knowledge base?');
   writeln (' ':13,'3. Exit from Cortex?');
   writeln;
   writeln (' ':10,'Make your selection by typing a key number.');
   writeln (' ':10,'Then press RETURN.');
   blankln (4);
   read (Selector);

   CASE Selector OF
      1: FindResult (CountOfQuestions, CountOfSolutions);
      2: KnowledgeBaseMenu (CountOfQuestions, CountOfSolutions);
      3: Flag:= false;
      OTHERWISE BEGIN
          ClrScr;
          Blankln (8);
          MenuError (3)
      END;  { of OTHERWISE }
   END;  { of CASE }

END;  { of WHILE }

END;

BEGIN  END.



{
```

```
SEGMENT     KBMenu;
             { Knowledge engineering main menu. }

{$I Common1.Typ}
{$I \PROPAS\PASPC}
PROCEDURE   Blankln (number: integer); EXTERNAL;
PROCEDURE   MenuError (range: integer); EXTERNAL;
PROCEDURE   QuestionMenu (VAR CountOfQuestions: integer); EXTERNAL;
PROCEDURE   SolutionMenu (VAR CounOfSolutions: integer); EXTERNAL;
PROCEDURE   ClassifierMenu (CountOfQuestions: integer); EXTERNAL;
PROCEDURE   WriteImplementationText; EXTERNAL;

PROCEDURE   KnowledgeBaseMenu (VAR CountOfQuestions, CountOfSolutions: integer);
      VAR   Flag: boolean;
            Selector: integer;

BEGIN
Flag:= true;
WHILE Flag = true DO BEGIN
   ClrScr;
   Selector:= 0;
   Blankln (3);
   writeln (' ':18,'Knowledge Engineering Menu.');
   writeln (' ':18,'*****************************');
   blankln (2);
   writeln (' ':7,'Do you want to;');
   writeln;
   writeln (' ':10,'1. Write, edit, delete or print the text of a question?');
   writeln (' ':10,'2. Write, edit, delete or print the text of a solution?');
   writeln (' ':10,'3. Set up, edit or delete a classifier?');
   writeln (' ':10,'4. Write the text of the implementation screen?');
   writeln (' ':10,'5. Return to the main Cortex menu?');
   Blankln (2);
   writeln (' ':7,'Make your choice by typing a key number.');
   writeln (' ':7,'Then press RETURN.');
   Blankln (4);
   writeln (' ':7,'The Cortex shell can accept up to a maximum of ',MaxNumberOfQuestions:3,'
questions.');
   writeln (' ':7,'There is effectively no limit upon the number of solutions.');
   read (Selector);
   CASE Selector OF
      1: QuestionMenu (CountOfQuestions);
      2: SolutionMenu (CountOfSolutions);
      3: ClassifierMenu (CountOfQuestions);
      4: WriteImplementationText;
      5: Flag:= false;
      OTHERWISE BEGIN  ClrScr; Blankln (8); MenuError (5)  END;
   END;  { of CASE }
END;  { of WHILE }
END;

BEGIN  END.
```

```
{
SEGMENT     WritImpl;
             { Create text of implementation message. }

{$I \PROPAS\PASPC}
PROCEDURE   Blankln (number: integer); EXTERNAL;
PROCEDURE   Pushpen (directory, FileName: string); EXTERNAL;

PROCEDURE   WriteImplementationText;

BEGIN

ClrScr;
Blankln (2);
writeln (' ':12,'Please write the text for the implementation message.');
Pushpen ('Implemnt','Message')

END;

BEGIN  END.
```

```
{
SEGMENT     FormFile;
             { Set up the name of a file to be accessed on disk. }

PROCEDURE   FormFileName (index: integer; title: string; VAR FileName: string);
VAR         Key: string;

BEGIN
str (index, Key);                                  { Form the string Key from the integer index }
IF title = 'question' THEN BEGIN
   FileName:= 'quest';                             { Write the stem of a question filename }
   insert (Key,FileName,6);                    { Append the key number to complete the filename }
END  { of IF }
ELSE IF title = 'solution' THEN BEGIN
        FileName:= 'solut';                        { Write the stem of a solution filename }
        insert (Key,FileName,6);                   { Append the key number to complete the filename }
     END  { of ELSE IF }
     ELSE IF title = 'explanation' THEN BEGIN
             FileName:= 'explan';                  { Write the stem of an explanation filename}
             insert (Key, FileName,7)              { Append the key number to complete the filename }
          END;  { of ELSE IF}

END;
BEGIN  END.
```

```
{
SEGMENT     PushPen;
             { Calls Writer to compose text, then stores it onto disk in specified sub-directory. }

{$I \PROPAS\PASDOS}
PROCEDURE   Writer; EXTERNAL;

PROCEDURE   PushPen (directory, FileName: string);
VAR         Disk, DiskText: text;
             Line: string[100];

BEGIN

Writer;                                  { Writer stores text as DiskText on diskFile TempFile. }
assign (DiskText, 'TempFile');                    { TempFile is on \SHELL, not \FORMAT. }
reset (DiskText);                                 { Open DiskText for input. }
ChDir (directory);
assign (Disk, FileName);
rewrite (Disk);                                   { Open Disk for output. }

   WHILE NOT eof(DiskText) DO BEGIN
      readln (DiskText, Line);                    { Read TempFile to FileName line by line. }
      writeln (Disk, Line)
   END; { of WHILE }

close (Disk, true);
erase (DiskText)

END;

BEGIN  END.
```

```
{
SEGMENT     ZeroBit;
             { Initialise bit strings. }

{$I Common1.Typ}
PROCEDURE   ZeroiseBitString (VAR Bits: bitstring);
      VAR   Index: integer;

BEGIN

FOR Index:= 1 TO MaxNumberOfIntegers DO
   Bits[Index]:= 0;

END;

BEGIN  END.
```

```
{
SEGMENT     Question;
             { Selects the operations to be performed on the question files. }

{$I \PROPAS\PASPC}
PROCEDURE   BlankIn (number: integer); EXTERNAL;
PROCEDURE   QuestionSelection (Selector: integer; VAR CountOfQuestion: integer; VAR Flag:
boolean); EXTERNAL;

PROCEDURE   QuestionMenu (VAR CountOfQuestions: integer);
      VAR   Flag: boolean;
             Selector: integer;

BEGIN

ClrScr;
Flag:= true;

WHILE Flag = true DO BEGIN
   writeln;
   writeln (' ':18,'Questions Text Menu.');
   writeln (' ':18,'********************');
   BlankIn (2);
   writeln (' ':7,'Do you want to;');
   writeln;
   writeln (' ':10,'1.  Write the text of a question?');
   writeln (' ':10,'2.  Write the explanation of a question?');
   writeln (' ':10,'3.  Edit the text of a question?');
   writeln (' ':10,'4.  Edit the explanation of a question?');
   writeln (' ':10,'5.  Delete a question from the questions file?');
   writeln (' ':10,'6.  Delete the explanation of a question from the file?');
   writeln (' ':10,'7.  Display the text of a question?');
   writeln (' ':10,'8.  Display the explanation of a question?');
   writeln (' ':10,'9.  Print the text of a question?');
   writeln (' ':10,'10. Print the explanation of a question?');
   writeln (' ':10,'11. Return to the knowledge base menu?');
   BlankIn (2);
   writeln (' ':7,'Make your choice by entering a key number.');
   writeln (' ':7,'Then press RETURN.');
   read (Selector);
   QuestionSelection (Selector, CountOfQuestions, Flag);
END;  { of WHILE }

END;

BEGIN  END.
```

```
{
SEGMENT     QstSelec;
             { Manage the calling of procedures by the question menu. }

{$I \PROPAS\PASPC}
PROCEDURE   WriteQuestionText (VAR CountOfQuestions: integer); EXTERNAL;
PROCEDURE   WriteQuestionExplanation; EXTERNAL;
PROCEDURE   EditQuestion; EXTERNAL;
PROCEDURE   EditQuestionExplanation; EXTERNAL;
PROCEDURE   DeleteQuestion (VAR CountOfQuestions: integer); EXTERNAL;
PROCEDURE   DeleteExplanation (title: string); EXTERNAL;
PROCEDURE   DisplayQuestion; EXTERNAL;
PROCEDURE   DisplayExplanation (title: string); EXTERNAL;
PROCEDURE   PrintAll; EXTERNAL;
PROCEDURE   BlankIn (number: integer); EXTERNAL;
PROCEDURE   MenuError (range: integer); EXTERNAL;

PROCEDURE   QuestionSelection (Selector: integer; VAR CountOfQuestions: integer;
                               VAR Flag: boolean);

BEGIN

CASE Selector OF
   1:  WriteQuestionText (CountOfQuestions);
   2:  WriteQuestionExplanation;
   3:  EditQuestion;
   4:  EditQuestionExplanation;
   5:  DeleteQuestion (CountOfQuestions);
   6:  DeleteExplanation ('question');
   7:  DisplayQuestion;
   8:  DisplayExplanation ('question');
   9:  PrintAll;
   10: PrintAll;
   11: Flag:= false;
   OTHERWISE BEGIN
      ClrScr;
      BlankIn (8);
      MenuError (11)
   END;  { of OTHERWISE }
END;  { of CASE }

END;

BEGIN  END.
```

```
{
SEGMENT     WritQust;
             { Write the text of a question and store it on disk. }

{$I \PROPAS\PASPC}
{$I \PROPAS\PASDOS}
PROCEDURE   Blankln (number: integer); EXTERNAL;
PROCEDURE   FormFileName (index: integer; title: string; VAR FileName: string); EXTERNAL;
PROCEDURE   PushPen (title, FileName: string); EXTERNAL;

PROCEDURE   WriteQuestionText (VAR CountOfQuestions: integer);
      VAR   FileName: string[30];
            QuestionOnFile: string[30];
            Counter: integer;

BEGIN

ClrScr;
ChDir ('\shell\question');                    { Questions are filed on question sub-directory }
Counter:= 0;

REPEAT
   Counter:= Counter + 1;
   FormFileName (Counter,'question',FileName);
   FindFile (FileName,QuestionOnFile);
UNTIL QuestionOnFile = '';                    { Until QuestionOnFile returns empty }

Blankln (2);
writeln (' ':7,'Enter the text of the question.');
writeln;
writeln ('Question no ',Counter:3);
PushPen ('question', FileName);   { Write question , and store in 'question' sub-directory }
CountOfQuestions:= CountOfQuestions + 1;
ChDir ('\shell')

END;

BEGIN  END.
```

```
{
SEGMENT     WrtQexpl;
             { Write the text of the explanation of a question. }

{$I \PROPAS\PASPC}
{$I \PROPAS\PASDOS}
PROCEDURE   Blankln (number: integer); EXTERNAL;
PROCEDURE   PressKey (margin: integer); EXTERNAL;
FUNCTION    YesNo: boolean; EXTERNAL;
PROCEDURE   FormFileName (index:integer; title: string; VAR FileName: string); EXTERNAL;
PROCEDURE   PushPen (title, FileName: string); EXTERNAL;

PROCEDURE   WriteQuestionExplanation;
      VAR   QuestionName, FileName: string;
             Index: integer;
             OK: boolean;

BEGIN
ClrScr;
Blankln (3);
ChDir ('question');
writeln (' ':7,'Enter the number of the question');
writeln (' ':7,'whose explanation you want to write.');
writeln;
writeln (' ':7,'Then press RETURN.');
GoToXY (8,8);
read (Index);
FormFileName (Index,'question',QuestionName);
FormFileName (Index,'explanation',FileName);
OK:= true;
IF fstat (QuestionName) = false THEN BEGIN
   Blankln (2);
   writeln (' ':7,'No question with this key number is on file.');
   PressKey (7);
   ClrScr
END;  { of IF }
IF (fstat(QuestionName) = true) AND (fstat(FileName) = true) THEN BEGIN
   Blankln (2);
   writeln (' ':7,'There is already an explanation for this question on file.');
   writeln (' ':7,'Do you want to overwrite it? If so, press ''y'' or ''Y''.');
   GoToXY (8,13);
   OK:= YesNo;
   ClrScr
END;  { of IF }
IF (fstat(QuestionName) = true) AND (OK = true) THEN BEGIN
   ClrScr;
   Blankln (2);
   writeln (' ':7,'Enter the text of the explanation.');
   writeln;
   writeln ('Explanation no ',Index:3);
   PushPen ('question',FileName);   { Write explanation and store in Question sub-directory }
   ClrScr
END;  { of IF }
ChDir ('\shell');
END;

BEGIN  END.
```

```
{
SEGMENT     EditQust;
             { Edit the text of an existing question. }

{$I \PROPAS\PASPC}
PROCEDURE   PressKey (margin: integer); EXTERNAL;
PROCEDURE   Blankln (number: integer); EXTERNAL;

PROCEDURE   EditQuestion;

BEGIN

ClrScr;
Blankln (8);
writeln (' ':7,'The procedure EditQuestion has not yet been written.');
Blankln (15);
PressKey (7);
ClrScr

END;

BEGIN  END.
```

```
{
SEGMENT     EdtQExpl;
             { Edit the explanation of a question. }

{$I \PROPAS\PASPC}
PROCEDURE   PressKey (margin: integer); EXTERNAL;
PROCEDURE   BlankIn (number: integer); EXTERNAL;
PROCEDURE   EditQuestionExplanation;

BEGIN

ClrScr;
BlankIn (8);
writeln (' ':7,'The procedure EditQuestionExplanantion has not yet been written.');
BlankIn (15);
PressKey (7);
ClrScr

END;

BEGIN
END.
```

```
SEGMENT     DelQust;
             { Delete question file together with any explanation file, and reset classifier bit
strings. }

{$I Common1.Typ}
{$I \PROPAS\PASPC}
{$I \PROPAS\PASDOS}
PROCEDURE   FormFileName (index: integer; title: string; VAR FileName: string); EXTERNAL;
PROCEDURE   ReNumberFiles (Flag, CountOfQuestions: integer; title: string); EXTERNAL;
PROCEDURE   BlankIn (number: integer); EXTERNAL;

PROCEDURE   DeleteQuestion (VAR CountOfQuestions: integer);
VAR         TextFile: text;
            TempFile: FILE OF classifiertype;
            FileName: string;
            Dummy: boolean;
            Flag, Index: integer;

BEGIN
ClrScr;
BlankIn (5);
writeln (' ':7,'What is the number of the question to be deleted?');
GoToXY (8,8);
read (Flag);
FormFileName (Flag,'question',FileName);                          { Form question file name }
ChDir ('question');
IF fstat (FileName) = true THEN BEGIN
   assign (TextFile,FileName);                                { Connect variable to question disk file }
   erase (TextFile);                                          { Delete the selected question disk file }
   ReNumberFiles (Flag,CountOfQuestions,'question');          { Close up succeeding question files }
   FormFileName (Flag,'explanation',FileName);                { Form explanation file name }
   IF fstat (FileName) = true THEN BEGIN
      assign (TextFile,FileName);                             { Connect variable to explanation disk file }
      erase (TextFile);                                       { Delete corresponding explanation disk file }
      ReNumberFiles (Flag,CountOfQuestions,'explanation'); { Close up succeeding explan files }
   END;  { of IF }
END;  { of IF }
CountOfQuestions:= CountOfQuestions - 1;
ChDir ('\class');
IF fstat ('Classif') = true THEN BEGIN
   assign (TempFile,'Classif');                               { Connect variable to classifier disk file }
   reset (TempFile);                                          { Go to first file element }
   WHILE NOT eof(TempFile) DO BEGIN
      FOR Index:= Flag TO CountOfQuestions DO BEGIN
          WHILE (Index+1) <= MaxNumberOfQuestions DO BEGIN       { Check against over-run }
             IF testbit(TempFile^.essentialmask,Index) <> testbit(TempFile^.essentialmask,Index+1) THEN
                Dummy:= flipbit(TempFile^.essentialmask,Index);  { Flip bit to value of next bit }
             IF testbit(TempFile^.usualmask,Index) <> testbit(TempFile^.usualmask,Index+1) THEN
                Dummy:= flipbit(TempFile^.usualmask,Index);      { Flip bit to value of next bit }
             IF testbit(TempFile^.classifier,Index) <> testbit(TempFile^.classifier,Index+1) THEN
                Dummy:= flipbit(TempFile^.classifier,Index);     { Flip bit to value of next bit }
          END;  { of WHILE }
      END;  { of FOR }
   get (TempFile)                                                { Go to next file element }
   END;  { of WHILE }
END;  { of IF }
ClrScr;  END;  BEGIN  END.
```

```
{
SEGMENT     Renum;
             { Re-number a series of disk files. }

PROCEDURE   FormFileName (Index: integer; title: string; VAR FileName: string); EXTERNAL;

PROCEDURE   ReNumberFiles (Flag, CountOfQuestions: integer; title: string);
      VAR   TextFile: text;
            FileName, ThisFileName, NextFileName: string;
            Index: integer;

BEGIN

FormFileName (Flag,title,FileName);                    { Form name of deleted file }
ThisFileName:= FileName;                               { Set variable to value of deleted file }

FOR Index:= (Flag+1) TO CountOfQuestions DO BEGIN
   FormFileName (Index,title,FileName);                { Form name of next disk file }
   NextFileName:= FileName;             { Set variable NextFileName to name of next disk file }

   IF fstat (NextFileName) = true THEN BEGIN
      assign (TextFile,NextFileName);      { Connect file variable with the next disk file }
      rename (TextFile,ThisFileName);      { Rename disk file with the name of the previous file }
      close (TextFile, true);
   END;  { of IF }

   ThisFileName:= NextFileName             { Update ThisFileName to the name of the next disk file }
END;  { of FOR }

END;

BEGIN  END.
```

```
{
SEGMENT     DelExpl;
             { Delete the explanation of a question or solution. }

{$I \PROPAS\PASPC}
{$I \PROPAS\PASDOS}
PROCEDURE   Blankln (number: integer); EXTERNAL;
PROCEDURE   PressKey (margin: integer); EXTERNAL;
PROCEDURE   FormFileName (index: integer; title: string; VAR FileName: string); EXTERNAL;

PROCEDURE   DeleteExplanation (title: string);
      VAR   TextFile: text;
             FileName: string;
             Flag: integer;

BEGIN

ClrScr;
Blankln (5);
writeln (' ':7,'What is the number of the ',title);
writeln (' ':7,'whose explanation you want to delete?');
GoToXY (8,9);
read (Flag);
FormFileName (Flag,'explanation',FileName);
ChDir (title);
IF fstat (FileName) = true THEN BEGIN
   assign (TextFile,FileName);
   erase (TextFile);
END  { of IF }
ELSE BEGIN
   writeln (' ':7,'No explanation for this ',title,' is on file.');
   PressKey (7);
END;  { of ELSE }

ChDir ('\shell');
ClrScr;

END;

BEGIN  END.
```

```
{
SEGMENT     DispQust;
             { Display the text of a question on screen. }

{$I \PROPAS\PASPC}
PROCEDURE   PressKey (margin: integer); EXTERNAL;
PROCEDURE   Blankln (number: integer); EXTERNAL;
PROCEDURE   FormFileName (filenumber:integer; title: string; VAR FileName: string); EXTERNAL;
PROCEDURE   DisplayTextFile (FileName, directory, heading: string; Index, displayline: integer);
EXTERNAL;

PROCEDURE   DisplayQuestion;
VAR         FileName: string[30];
            Index: integer;

BEGIN

ClrScr;
Blankln (6);
writeln (' ':7,'Enter the number of the question');
writeln (' ':7,'that you want to display.');
writeln;
writeln (' ':7,'Then press RETURN.');
GoToXY (8,11);
read (Index);
FormFileName (Index,'question',FileName);
FileName:= concat ('\SHELL\QUESTION\',FileName);

IF fstat (FileName) = false THEN BEGIN
   Blankln (2);
   writeln (' ':7,'No question with this key number is on file.');
   PressKey (7);
   ClrScr;
END  { of IF }
ELSE BEGIN
   ClrScr;
   Blankln (6);
   DisplayTextFile (FileName, 'question', 'Question', Index, 7);
   Blankln (12);
   PressKey (5);
   ClrScr
END;  { of ELSE }

END;

BEGIN  END.
```

```
{
SEGMENT     DispFile;
             { Display disk file on screen }

{$I \PROPAS\PASPC}
{$I \PROPAS\PASDOS}
PROCEDURE   DisplayTextFile (DiskFile, directory, heading: string; Index, displayline: integer);
VAR         TempFile: text;
             Line: string[100];

BEGIN

assign (TempFile, DiskFile);
ChDir (directory);                              { Change to subdirectory containing the file. }
reset (TempFile);
GoToXY (1,displayline);
writeln (heading,' no ',Index:3,'.');

WHILE NOT eof(TempFile) DO BEGIN
   readln (TempFile, Line);                      { Read a line of text from TempFile to Line. }
   GoToXY (24,displayline);                      { Position cursor. }
   writeln (Line);                               { Output line of text to screen. }
   displayline:= displayline + 1                 { Move cursor position down one line. }
END;  { of WHILE }

close (TempFile, true);
ChDir ('\shell');                                { Return to SHELL directory. }

END;

BEGIN  END.
```

```
{
SEGMENT     DispExpl;
             { Display the text of the explanation of a question or a solution. }

{$I \PROPAS\PASPC}
PROCEDURE   PressKey (margin: integer); EXTERNAL;
PROCEDURE   BlankLn (number: integer); EXTERNAL;
PROCEDURE   FormFileName (index: integer; title: string; VAR FileName: string); EXTERNAL;
PROCEDURE   DisplayTextFile (filename, directory, heading: string; index, firstline: integer);
EXTERNAL;

PROCEDURE   DisplayExplanation (title: string);
VAR         FileName: string[30];
             Index: integer;

BEGIN

ClrScr;
BlankLn (6);
writeln (' ':7,'Enter the number of the ',title);
writeln (' ':7,'whose explanation you want to display.');
writeln;
writeln (' ':7,'Then press RETURN.');
GoToXY (8,11);
read (Index);
FormFileName (Index,'explanation',FileName);
FileName:= concat ('\shell\',title,'\',FileName);

IF fstat (FileName) = false THEN BEGIN
   BlankLn (2);
   writeln (' ':7,'No explanation for this ',title,' is on file.');
   PressKey (7);
   ClrScr;
END  { of IF }
   ELSE BEGIN
      ClrScr;
      BlankLn (6);
      DisplayTextFile (FileName, 'question', 'Explanation', Index, 7);
      BlankLn (12);
      PressKey (5);
      ClrScr
   END;  { of ELSE }

END;

BEGIN  END.
```

```
{
SEGMENT     PrintAll;
             { Print the text of a question, solution or explanation. }

{$I \PROPAS\PASPC}
PROCEDURE   BlankIn (number: integer); EXTERNAL;
PROCEDURE   PressKey (margin: integer); EXTERNAL;

PROCEDURE PrintAll;

BEGIN

ClrScr;
BlankIn (8);
writeln (' ':7,'The procedure PrintAll has not yet been written.');
BlankIn (15);
PressKey (7);
ClrScr

END;

BEGIN  END.
```

```
{
SEGMENT     Solution;
             { Selects the operations to be performed on the solution files. }

{$I \PROPAS\PASPC}
PROCEDURE   Blankln (number: integer); EXTERNAL;
PROCEDURE   SolutionSelection (Selector: integer; VAR Flag: boolean); EXTERNAL;

PROCEDURE   SolutionMenu;
      VAR   Flag: boolean;
             Selector: integer;

BEGIN

ClrScr;
Flag:= true;

WHILE Flag = true DO BEGIN
   writeln;
   writeln (' ':18,'Solutions Text Menu.');
   writeln (' ':18,'********************');
   Blankln (2);
   writeln (' ':7,'Do you want to;');
   writeln;
   writeln (' ':10,'1.  Write the text of a solution?');
   writeln (' ':10,'2.  Write the explanation of a solution?');
   writeln (' ':10,'3.  Edit the text of a solution?');
   writeln (' ':10,'4.  Edit the explanation of a solution?');
   writeln (' ':10,'5.  Delete a solution from the solutions file?');
   writeln (' ':10,'6.  Delete the explanation of a solution from the file?');
   writeln (' ':10,'7.  Display the text of a solution?');
   writeln (' ':10,'8.  Display the explanation of a solution?');
   writeln (' ':10,'9.  Print the text of a solution?');
   writeln (' ':10,'10. Print the explanation of a solution?');
   writeln (' ':10,'11. Return to the knowledge base menu?');
   Blankln (2);
   writeln ('Make your choice by entering a key number.');
   writeln ('Then press RETURN.');
   read (Selector);
   SolutionSelection (Selector,Flag);
   END;  { of WHILE }

   END;

   BEGIN  END.
```

```
{
SEGMENT     SolSelec;
             { Manages the calling of procedures by the solutions menu. }

{$I \PROPAS\PASPC}
PROCEDURE   WriteSolutionText; EXTERNAL;
PROCEDURE   WriteSolutionExplanation; EXTERNAL;
PROCEDURE   EditSolution; EXTERNAL;
PROCEDURE   EditSolutionExplanation; EXTERNAL;
PROCEDURE   DeleteSolution; EXTERNAL;
PROCEDURE   DeleteExplanation (title: string); EXTERNAL;
PROCEDURE   DisplaySolution; EXTERNAL;
PROCEDURE   DisplayExplanation (title: string); EXTERNAL;
PROCEDURE   PrintAll (title: string); EXTERNAL;
PROCEDURE   BlankIn (number: integer); EXTERNAL;
PROCEDURE   MenuError (range: integer); EXTERNAL;

PROCEDURE   SolutionSelection (Selector: integer; VAR Flag: boolean);

BEGIN

CASE Selector OF
   1:  WriteSolutionText;
   2:  WriteSolutionExplanation;
   3:  EditSolution;
   4:  EditSolutionExplanation;
   5:  DeleteSolution;
   6:  DeleteExplanation ('solution');
   7:  DisplaySolution;
   8:  DisplayExplanation ('solution');
   9:  PrintAll ('solution');
   10: PrintAll ('solution');
   11: Flag:= false;
   OTHERWISE BEGIN
      ClrScr;
      BlankIn (8);
      MenuError (11)
   END;  { of OTHERWISE }
END;  { of CASE }

END;

BEGIN  END.
```

```
{
SEGMENT     WritSoln;
             { Write the text of a solution and store it on disk. }

{$I \PROPAS\PASPC}
{$I \PROPAS\PASDOS}
PROCEDURE   Blankln (number: integer); EXTERNAL;
PROCEDURE   FormFileName (index: integer; title: string; VAR FileName: string); EXTERNAL;
PROCEDURE   PushPen (title, FileName: string); EXTERNAL;

PROCEDURE   WriteSolutionText (VAR CountOfSolutions: integer);
      VAR   FileName: string[30];
            SolutionOnFile: string[30];
            Counter: integer;

BEGIN

ClrScr;
ChDir ('\shell\solution');                    { Solutions are filed on solution sub-directory }
Counter:= 0;

REPEAT
   Counter:= Counter + 1;
   FormFileName (Counter,'solution',FileName);
   FindFile (FileName,SolutionOnFile);
UNTIL SolutionOnFile = '';                    { Until SolutionOnFile returns empty }

Blankln (2);
writeln (' ':7,'Enter the text of the solution.');
writeln;
writeln ('Solution no ',Counter:3);
PushPen ('solution', FileName);       { Write solution , and store in 'solution' sub-directory }
CountOfSolutions:= CountOfSolutions + 1;
ChDir ('\shell')

END;

BEGIN  END.
```

```
{
SEGMENT     EditSoln;
             { Edit the text of an existing solution. }

{$I \PROPAS\PASPC}
PROCEDURE   PressKey (margin:integer); EXTERNAL;
PROCEDURE   Blankln (number:integer); EXTERNAL;

PROCEDURE   EditSolution;

BEGIN

ClrScr;
Blankln (8);
writeln (' ':7,'The procedure EditSolution has not yet been written.');
Blankln (15);
PressKey (7);
ClrScr;

END;

BEGIN  END.
```

```
{
SEGMENT     DelSoln;
             { Delete solution file. }

{$I \PROPAS\PASPC}
PROCEDURE   PressKey (margin:integer); EXTERNAL;
PROCEDURE   Blankln (number:integer); EXTERNAL;

PROCEDURE  DeleteSolution;

BEGIN

ClrScr;
Blankln (8);
writeln (' ':7,'The procedure DeleteSolution has not yet been written.');
Blankln (15);
PressKey (7);
ClrScr;

END;

BEGIN  END.
```

```
{
SEGMENT     DispSoln;
             { Display the text of a question on screen. }

{$I \PROPAS\PASPC}
PROCEDURE   PressKey (margin: integer); EXTERNAL;
PROCEDURE   Blankln (number: integer); EXTERNAL;
PROCEDURE   FormFileName (filenumber:integer; title: string; VAR FileName: string); EXTERNAL;
PROCEDURE   DisplayTextFile (FileName, directory, heading: string; Index, displayline: integer);
EXTERNAL;

PROCEDURE   DisplaySolution;
VAR         FileName: string[30];
             Index: integer;

BEGIN

ClrScr;
Blankln (6);
writeln (' ':7,'Enter the number of the solution');
writeln (' ':7,'that you want to display.');
writeln;
writeln (' ':7,'Then press RETURN.');
GoToXY (8,11);
read (Index);
FormFileName (Index,'solution',FileName);
FileName:= concat ('\shell\solution\',FileName);

IF fstat (FileName) = false THEN BEGIN
   Blankln (2);
   writeln (' ':7,'No solution with this key number is on file.');
   PressKey (7);
   ClrScr;
END  { of IF }
ELSE BEGIN
   ClrScr;
   Blankln (6);
   DisplayTextFile (FileName, 'solution', 'Solution', Index, 7);
   Blankln (12);
   PressKey (5);
   ClrScr
END;  { of ELSE }

END;

BEGIN  END.
```

```
{
SEGMENT     WrtSexpl;
             { Write the text of the explanation of a solution. }

{$I Common1.Typ}
{$I \PROPAS\PASPC}
{$I \PROPAS\PASDOS}
PROCEDURE   Blankln (number:integer); EXTERNAL;
PROCEDURE   PressKey (margin:integer); EXTERNAL;
FUNCTION    YesNo: boolean; EXTERNAL;
PROCEDURE   FormFileName (index:integer;title:string;VAR FileName:string); EXTERNAL;
PROCEDURE   PushPen (title,FileName:string); EXTERNAL;

PROCEDURE   WriteSolutionExplanation;
      VAR   SolutionName, FileName: string;
             Index: integer;
             OK: boolean;

BEGIN
ClrScr;
Blankln (3);
ChDir ('solution');
writeln (' ':7,'Enter the number of the solution');
writeln (' ':7,'whose explanation you want to write.');
writeln;
writeln (' ':7,'Then press RETURN.');
GoToXY (8,8);
read (Index);
FormFileName (Index,'solution',SolutionName);
FormFileName (Index,'explanation',FileName);
OK:= true;
IF fstat (SolutionName) = false THEN BEGIN
   Blankln (2);
   writeln (' ':7,'No solution with this key number is on file.');
   PressKey (7);
   ClrScr
END;  { of IF }
IF (fstat(SolutionName) = true) AND (fstat(FileName) = true) THEN BEGIN
   Blankln (2);
   writeln (' ':7,'There is already an explanation for this solution on file.');
   writeln (' ':7,'Do you want to overwrite it? If so, press ''y'' or ''Y''.');
   GoToXY (8,13);
   OK:= YesNo;
   ClrScr
END;  { of IF }
IF (fstat(SolutionName) = true) AND (OK = true) THEN BEGIN
   ClrScr;
   Blankln (2);
   writeln (' ':7,'Enter the text of the explanation.');
   writeln;
   writeln ('Explanation no ',Index:3);
   PushPen ('solution',FileName);      { Write explanation and store in Solution sub-directory }
   ClrScr
END;  { of IF }
ChDir ('\shell');
END;
BEGIN  END.
```

```
{
SEGMENT     EdtSExpl;
             { Edit the explanation of a solution. }

{$I \PROPAS\PASPC}
PROCEDURE   PressKey (margin:integer); EXTERNAL;
PROCEDURE   Blankln (number:integer); EXTERNAL;

PROCEDURE   EditSolutionExplanation;

BEGIN

ClrScr;
Blankln (8);
writeln (' ':7,'The procedure EditSolutionExplanation has not yet been written.');
Blankln (15);
PressKey (7);
ClrScr;

END;

BEGIN  END.
```

```
{
SEGMENT     Classify;
             ( Select the operations to be performed on the classifier file. )

{$I \PROPAS\PASPC}

PROCEDURE   BlankIn (number:integer); EXTERNAL;
PROCEDURE   MenuError (range:integer); EXTERNAL;
PROCEDURE   SetUpClassifier (CountOfQuestions:integer); EXTERNAL;
PROCEDURE   EditClassifier; EXTERNAL;
PROCEDURE   DeleteClassifier (CountOfQuestions:integer); EXTERNAL;

PROCEDURE   ClassifierMenu (CountOfQuestions: integer);
      VAR   Flag: boolean;
             Selector: integer;

BEGIN

ClrScr;
Flag:= true;

WHILE Flag = true DO BEGIN
   BlankIn (5);
   writeln (' ':20,'Classifier Menu.');
   writeln (' ':20,'****************');
   BlankIn (2);
   writeln (' ':11,'Do you want to;');
   writeln;
   writeln (' ':14,'1. Set up a classifier?');
   writeln (' ':14,'2. Edit a classifier?');
   writeln (' ':14,'3. Delete a classifier?');
   writeln (' ':14,'4. Return to the Knowledge Engineering Menu?');
   writeln;
   writeln (' ':7,'Make your choice by typing a keynumber.');
   writeln (' ':7,'Then press RETURN.');
   read (Selector);

   CASE Selector OF
      1: SetUpClassifier (CountOfQuestions);
      2: EditClassifier;
      3: DeleteClassifier (CountOfQuestions);
      4: Flag:= false;
      OTHERWISE BEGIN
          ClrScr;
          BlankIn (8);
          MenuError (4)
        END;  { of OTHERWISE }
     END;  { of CASE }

END;  { of WHILE }

END;

BEGIN  END.
```

```
SEGMENT   StUpClas;
          { Write the classifier, essential mask and usual mask for a solution. }
{$I Common1.Typ}
{$I \PROPAS\PASPC}
{$I \PROPAS\PASDOS}
PROCEDURE   BlankIn (number:integer); EXTERNAL;
PROCEDURE   FormFileName (filenumber:integer;title:string;VAR FileName:string); EXTERNAL;
PROCEDURE   DisplayTextFile (FileName,directory,heading:string;
                             filenumber,firstline:integer); EXTERNAL;
PROCEDURE   InitialiseClassifierRecord (SolutionNumber:integer;VAR Temp:solutiontype); EXTERNAL;
PROCEDURE   SetClassifierBits (CountOfQuestions:integer;VAR Temp:solutiontype;
                               VAR Flag:boolean); EXTERNAL;
FUNCTION    Ask (leftmargin:integer;question:string):boolean; EXTERNAL;
FUNCTION    YesNo: boolean; EXTERNAL;

PROCEDURE   SetUpClassifier (CountOfQuestions:integer);
     VAR    TempFile: FILE OF solutiontype;
            Temp: solutiontype;
            FileName: string;
            SolutionNumber: integer;
            Answer, Flag: boolean;
BEGIN
CursorOff;
Flag:= true;
WHILE Flag = true DO BEGIN
   ClrScr;   BlankIn (5);
   writeln (' ':7,'What is the number of the solution');
   writeln (' ':7,'for which you want to write a classifier?');
   GoToXY (8,8);
read (SolutionNumber);
InitialiseClassifierRecord (SolutionNumber,Temp);        { Initialise the fields of Temp }
ClrScr;
FormFileName (SolutionNumber,'solution',FileName);
FileName:= concat ('\shell\solution\',FileName);
IF fstat (FileName) = false THEN
   writeln (' ':7,'No solution with this key number is on file.')
ELSE BEGIN
   TextWindow (1,2,80,9);
   TextFrame (false);
   DisplayTextFile (filename,'solution','Solution',SolutionNumber,2); { Disp solution text }
   SetClassifierBits (CountOfQuestions,Temp,Flag);
   assign (TempFile,'Classif');      { Connect file variable TempFile to disk file Classif }
   append (TempFile);                              { Move pointer to end of file }
   write (TempFile,Temp);                          { Add Temp to end of TempFile }
   Close (TempFile,true);
END;  { of ELSE }
TextWindow (1,1,80,25);
ClrScr;  GoToXY (8,3);
Answer:= Ask (1,'Do you want to write another classifier. Y/N?');
IF Answer = true THEN BEGIN
   Flag:= true;  ClrScr;  END
ELSE BEGIN
   Flag:= false; ClrScr;  END;
END;  { of WHILE }
ClrScr;  END;  BEGIN  END.
```

```
{
SEGMENT     InitClas;
             { Initialise the fields of the record Temp. }

{$I Common1.Typ}
PROCEDURE   ZeroiseBitString (VAR Bits:bitstring); EXTERNAL;


PROCEDURE   InitialiseClassifierRecord (SolutionNumber:integer;VAR Temp:solutiontype);

BEGIN

Temp.keynumber:= SolutionNumber;
ZeroiseBitString (Temp.essentialmask);
ZeroiseBitString (Temp.usualmask);
Temp.totalusual:= 0;
Temp.usualtrue:= 0;
Temp.probability:= 0;
ZeroiseBitString (Temp.classifier);

END;

BEGIN  END.
```

```
{
SEGMENT     ClasBits;
             { Set the bits of the classifier and its masks. }

{$I Common1.Typ}
{$I \PROPAS\PASPC}
{$I \PROPAS\PASDOS}
PROCEDURE   FormFileName (filenumber:integer;title:string;VAR FileName:string); EXTERNAL;
PROCEDURE   DisplayTextFile (FileName,directory,heading:string;
                             filenumber,firstline:integer); EXTERNAL;
PROCEDURE   SetUsualBits (Index:integer;VAR Temp:solutiontype); EXTERNAL;
FUNCTION    Ask (leftmargin:integer;question:string):boolean; EXTERNAL;
FUNCTION    YesNo: boolean; EXTERNAL;

PROCEDURE   SetClassifierBits (CountOfQuestions:integer;VAR Temp:solutiontype;VAR Flag:boolean);
      VAR   TempFile: FILE OF solutiontype;
             FileName: string;
             Index: integer;
             Answer, Dummy: boolean;

BEGIN

CursorOff;

FOR Index:= 1 TO CountOfQuestions DO BEGIN                  { Loop through all the questions }
   TextWindow (1,10,80,25);
   FormFileName (Index,'question',FileName);
   FileName:= concat ('\shell\question\',FileName);
   ClrScr;
   DisplayTextFile (FileName,'question','Question',Index,2);      { Display question text }
   GoToXY (8,7);
   Answer:= Ask (1,'Is it essential that this question receives a correct answer. Y/N?');

   IF Answer = true THEN BEGIN
      Dummy:= setbit (Temp.essentialmask,Index);            { Set essential mask bit to true }
      GoToXY (8,9);
      writeln ('Is the answer that is always needed');
      writeln (' ':7,'to this question ''yes'' or ''no''?');
      Answer:= YesNo;
      IF Answer = true THEN
          Dummy:= setbit (Temp.classifier,Index);           { Set classifier bit to true }
   END  { of IF }
   ELSE
      SetUsualBits (Index,Temp);
   ClrScr;                                          { Clear only the lower part of the screen }
END;  { of FOR }

END;

BEGIN  END.
```

```
{
SEGMENT     Usulbits;
             { Set the usual bits in a classifier. }

{$I Common1.Typ}
{$I \PROPAS\PASPC}
{$I \PROPAS\PASDOS}

FUNCTION    YesNo: boolean; EXTERNAL;
FUNCTION    Ask (leftmargin:integer;question:string): boolean; EXTERNAL;

PROCEDURE   SetUsualBits (Index:integer;VAR Temp:solutiontype);
      VAR   Answer, Dummy: boolean;

BEGIN

TextWindow (1,12,80,25);
ClrScr;
GoToXY (8,5);
writeln ('Does the solution usually require');
writeln (' ':7,'a correct answer to this question. Y/N?');
Answer:= YesNo;

IF Answer = true THEN BEGIN
   Dummy:= setbit (Temp.usualmask,Index);              { Set usual mask bit to true }
   Temp.totalusual:= Temp.totalusual + 1;
   ClrScr;
   GoToXY (8,5);
   writeln ('Is the answer that is usually needed');
   writeln (' ':7,'to this question ''yes'' or ''no''. Y/N?');
   Answer:= YesNo;

   IF Answer = true THEN
      Dummy:= setbit (Temp.classifier,Index);          { Set classifier bit to true }
END;  { of IF }

END;

BEGIN  END.
```

```
{
SEGMENT     EditClas;
             { Edit a classifier. }

{$I Common1.Typ}
{$I \PROPAS\PASPC}
{$I \PROPAS\PASDOS}
PROCEDURE   BlankIn (number:integer); EXTERNAL;
PROCEDURE   FormFileName (filenumber:integer;title:string;VAR FileName:string); EXTERNAL;
FUNCTION    Ask (leftmargin:integer;question:string): boolean; EXTERNAL;

PROCEDURE   EditClassifier;
      VAR   TempFile: FILE OF solutiontype;
            SolutionFile, QuestionFile: text;
            TempRecord: solutiontype;
            FileName: string[30];
            Line: string[100];
            SolutionNumber, QuestionNumber, Counter: integer;
            Answer, ClassifierSetting, Dummy: boolean;

BEGIN
ClrScr;
CursorOff;
BlankIn (8);
writeln (' ':7,'Enter the number of the solution whose classifier you want to edit.');
read (SolutionNumber);
ClrScr;
BlankIn (5);
writeln (' ':7,'The text of the solution whose classifier you are editing is;');
FormFileName (SolutionNumber,'solution',FileName);
FileName:= concat ('\shell\solution\',FileName);
assign (SolutionFile,FileName);
reset (SolutionFile);
Counter:= 8;
WHILE NOT eof(SolutionFile) DO BEGIN
   readln (SolutionFile, Line);
   GoToXY (15,Counter);
   writeln (Line);
   Counter:= Counter + 1;
END;  { of WHILE }
writeln;
writeln (' ':7,'Enter the number of the question that you want to change.');
read (QuestionNumber);
writeln;
writeln (' ':7,'The question whose bit you are editing is;');
FormFileName (QuestionNumber,'question',FileName);
FileName:= concat ('\shell\question\',FileName);
assign (QuestionFile,FileName);
reset (QuestionFile);
Counter:= 16;
WHILE NOT eof(QuestionFile) DO BEGIN
   readln (QuestionFile, Line);
   GoToXY (15,Counter);
   writeln (Line);
   Counter:= Counter + 1
END;  { of WHILE }
assign (TempFile,'Classif');
```

```
update (TempFile);
seek (TempFile,(SolutionNumber - 1));
read (TempFile, TempRecord);
writeln;
ClassifierSetting:= testbit(TempRecord.classifier,QuestionNumber);
writeln (' ':7,'The setting of this question bit is ',ClassifierSetting);
Answer:= Ask (7,'Do you want to change its setting. Y/N?');
IF Answer = true THEN BEGIN
   Dummy:= flipbit(TempRecord.classifier,QuestionNumber);
   seek (TempFile,(SolutionNumber - 1));
   write (TempFile,TempRecord);
   close (TempFile,true)
END;  { of IF }
ClrScr
END;
BEGIN  END.
```

```
{
SEGMENT  DelClass;
         { Delete a specified classifier and close up the succeeding file elements. }

{$I Common1.Typ}
{$I \PROPAS\PASPC}
{$I \PROPAS\PASDOS}

PROCEDURE   DeleteClassifier (CountOfQuestions:integer);
      VAR   TempFile: FILE OF classifiertype;
            Temp: classifiertype;
            Selector, Counter, Index: integer;

BEGIN
ClrScr;
writeln (' ':7,'What is the number of the solution');
writeln (' ':7,'whose classifier you want to delete?');
read (Selector);
assign (TempFile,'Classif');
reset (TempFile);
WHILE NOT eof (TempFile) DO BEGIN
   IF TempFile^.keynumber <> Selector THEN          { File pointer not at selected element }
      get (TempFile);                                                  { Go to next element }
   IF eof (TempFile) = true THEN
      writeln (' ':7,'No classifier has been written for this solution.')  { Selected classifier
                                                                             not found }
ELSE BEGIN                                          { File pointer is at selected element }
   Counter:= Selector;
   FOR Index:= Selector TO CountOfQuestions DO BEGIN
      seek (TempFile,(Counter+1));                                  { Go to next file element }
      Temp.keynumber:= TempFile^.keynumber;       { Set Temp's fields to this element's values }
      Temp.essentialmask:= TempFile^.essentialmask;
      Temp.usualmask:= TempFile^.usualmask;
      Temp.classifier:= TempFile^.classifier;
      Temp.totalusual:= TempFile^.totalusual;
      Counter:= Counter - 1;                        { Go back to selected file element }
      write (TempFile,Temp);                       { Overwrite element with values of next element }
      seek (TempFile,(Counter+1))
   END;  { of FOR }
END;  { of ELSE }
END;  { of WHILE }
close (TempFile,true)
END;
BEGIN  END.
```

```
{
SEGMENT     FindRslt;
             { Main procedure for writing the message and obtaining the result. }

{$I Common1.Typ}
PROCEDURE   ZeroiseBitString (VAR Bit:Bitstring); EXTERNAL;
PROCEDURE   SetUpSolutionList (CountOfSolutions:integer;VAR Head:solutionpointer); EXTERNAL;
PROCEDURE   SetMessageAndMask (LiveQuestion:integer;VAR Message,MessageMask:bitstring); EXTERNAL;
PROCEDURE   RemoveContradictedSolutions (Message,MessageMask:bitstring;MFQ:integer;
                                                    VAR Head:solutionpointer); EXTERNAL;
PROCEDURE   CalculateProbability (Message,MessageMask:bitstring;CountOfQuestions:integer;
                                          Head:solutionpointer); EXTERNAL;
PROCEDURE   DisplayResult (Head:solutionpointer); EXTERNAL;
PROCEDURE   ClearHeap (Head:solutionpointer); EXTERNAL;
FUNCTION    FindMostFrequentQuestion (MessageMask:bitstring;CountOfQuestions:integer;
                                              Head:solutionpointer): integer; EXTERNAL;
FUNCTION    FindUsualQuestion (MessageMask:bitstring;CountOfQuestions:integer;
                                    Head:solutionpointer): integer; EXTERNAL;

PROCEDURE   FindResult (CountOfQuestions,CountOfSolutions:integer);
      VAR   Message, MessageMask: bitstring;
             Head: solutionpointer;
             UQ, MFQ: integer;

BEGIN

ZeroiseBitString (Message);                                    { Set Message and Mask to zero }
ZeroiseBitString (MessageMask);
SetUpSolutionList (CountOfSolutions,Head);                     { Create solutions list }
MFQ:= FindMostFrequentQuestion (MessageMask,CountOfQuestions,Head);  { Find number of MFQ }

WHILE MFQ <> 0 DO BEGIN                          { Search essential masks & prune solutions list }
   SetMessageAndMask (MFQ,Message,MessageMask);          { Set Message and MessageMask for MFQ }
   RemoveContradictedSolutions (Message,MessageMask,MFQ,Head); { Remove contradicted solution }
   MFQ:= FindMostFrequentQuestion (MessageMask,CountOfQuestions,Head); { Find no MFQ remaining }
END;  { of WHILE }

UQ:= FindUsualQuestion (MessageMask,CountOfQuestions,Head);                { Find number of UQ }

WHILE UQ <> 0 DO BEGIN
   SetMessageAndMask (UQ,Message,MessageMask);                                { Set Message and
MessageMask for UQ }
   UQ:= FindUsualQuestion (MessageMask,CountOfQuestions,Head);      { Find number of next UQ }
END;  { of WHILE }

CalculateProbability (Message,MessageMask,CountOfQuestions,Head);
DisplayResult (Head);
ClearHeap (Head);

END;

BEGIN  END.
```

```
{
SEGMENT     SetUpSol;
             ( Set up a linked list of solutions and return the Head pointer. )

{$I Common1.Typ}
{$I \PROPAS\PASPC}

PROCEDURE   SetUpSolutionList (CountOfSolutions:integer;VAR Head:solutionpointer);
      VAR   Classifier: FILE OF solutiontype;
            TempRecord: solutiontype;
            Temp, Current: solutionpointer;
            Index: integer;

BEGIN

new (Current);                                  { Make space in heap for last element in the list }
Current^.keynumber:= CountOfSolutions;          { Number of last element := total number of solutions }
Current^.next:= NIL;                            { Set pointer field of last element to NIL }

FOR Index:= (CountOfSolutions - 1) DOWNTO 1 DO BEGIN  { Work backward from last element in list }
   new (Temp);                                    { Make space in heap for a new element in the list }
   Temp^.keynumber:= Index;                                  { Set keynumber to loop index }
   Temp^.next:= Current;                        { Set pointer to current element in the list }
   Current:= Temp;                              { Make the current element the new element }
END;  { of FOR }

Head:= Current;   { Move head pointer to first solution - current pointer -  on exiting FOR loop }
assign (Classifier,'Classif');                          { Connect Classifier to disk file 'Classif' }
reset (Classifier);
Temp:= Head;                                  { Position Temp at beginning of solutions list }

FOR Index:= 1 TO CountOfSolutions DO BEGIN
   read (Classifier,TempRecord);
   Temp^.essentialmask:= TempRecord.essentialmask;
   Temp^.usualmask:= TempRecord.usualmask;
   Temp^.classifier:= TempRecord.classifier;
   Temp^.totalusual:= TempRecord.totalusual;
   Temp^.usualtrue:= 0;
   Temp^.probability:= 0;
   ClrScr;
   Temp:= Temp^.next;                                 { Move pointer to next record }
END;  { of FOR }

END;

BEGIN  END.
```

```
{
SEGMENT     MFrqQust;


{$I Common1.Typ}
{$I \PROPAS\PASDOS}

FUNCTION    FindMostFrequentQuestion (MessageMask:bitstring;CountOfQuestions:integer;
                                      Head:solutionpointer):integer;
     VAR    Current: solutionpointer;
             TopQuestion, QuestionCounter: questiontype;
             Index: integer;
             AllFalseFlag, AllTrueFlag: boolean;

BEGIN

TopQuestion.questionnumber:= 0;
TopQuestion.countofoccurrance:= 0;

FOR Index:= 1 TO CountOfQuestions DO BEGIN
   AllFalseFlag:= false;
   AllTrueFlag:= false;
   QuestionCounter.questionnumber:= Index;
   QuestionCounter.countofoccurrance:= 0;
   Current:= Head;

   WHILE Current <> NIL DO BEGIN
      IF (testbit(Current^.essentialmask,Index) = true) AND (testbit(MessageMask,Index) = false) THEN BEGIN
          QuestionCounter.countofoccurrance:= (QuestionCounter.countofoccurrance + 1);
          IF testbit(Current^.classifier,Index) = true THEN
             AllTrueFlag:= true
          ELSE
             AllFalseFlag:= true;
      END;  { of IF }
   Current:= Current^.next
   END;  { of WHILE }

   IF AllFalseFlag AND AllTrueFlag = true THEN
      IF QuestionCounter.countofoccurrance > TopQuestion.countofoccurrance THEN
          TopQuestion:= QuestionCounter
END;  { of FOR }

   FindMostFrequentQuestion:= TopQuestion.questionnumber;

END;
BEGIN  END.
```

```
SEGMENT     MessMask;
             { Display the most frequently occurring question and obtain the user's answer to it. }
{$I Common1.Typ}
{$I \PROPAS\PASPC}
{$I \PROPAS\PASDOS}
PROCEDURE   BlankIn (number:integer); EXTERNAL;
PROCEDURE   PressKey (margin:integer); EXTERNAL;
PROCEDURE   FormFileName (filenumber:integer;title:string;VAR FileName:string); EXTERNAL;
PROCEDURE   DisplayTextFile (DiskFile,directory,heading:string;filenumber,firstline:integer);
EXTERNAL;

PROCEDURE   SetMessageAndMask (LiveQuestion:integer;VAR Message,MessageMask:bitstring);
      VAR   FileName: string[30];
            Gate: char;
            Answer, Dummy: boolean;
BEGIN
CursorOff;
TextWindow (1,1,80,7);
TextFrame (true);
ClrScr;
FormFileName (LiveQuestion,'question',FileName);
DisplayTextFile (FileName,'question','Question',LiveQuestion,2); { Display question text }
TextWindow (1,1,80,25);
GoToXY (1,10);
writeln (' ':7,'Please answer the question ''Yes'' or ''No''.');
TextWindow (1,23,80,25);
TextFrame (false);
GoToXY (1,23);                                          { Position display at bottom of screen }
writeln (' ':3,'Press ''W'' for ''Why'' if you want to see an explanation of this question.');
Gate:= ConSilent;                                       { Wait for Y, N or W keyboard input }
IF (Gate = 'W') OR (Gate = 'w') THEN BEGIN
   TextWindow (1,11,80,25);                             { Set text window to lower part of screen }
   FormFileName (LiveQuestion,'explanation',FileName);
   FileName:= concat('\shell\question\',FileName);
   IF fstat (FileName) = false THEN BEGIN
      BlankIn (3);
      writeln (' ':7,'No explanation for this question is on file.');
      PressKey (7);
      ClrScr;
   END  { of IF }
   ELSE BEGIN
      TextWindow (1,18,80,22);
      TextFrame (False);
      DisplayTextFile (FileName,'question','Explanation',LiveQuestion,1); { Display explan text }
   END;  { of ELSE }
   Gate:= ConSilent;                                    { Wait for new Y or N keyboard input }
   ClrScr;
END;  { of IF }
IF (Gate = 'Y') OR (Gate = 'y') THEN
   Dummy:= setbit (Message,LiveQuestion);      { Set Message to true if answer is 'yes'}
Dummy:= setbit (MessageMask,LiveQuestion); { Set MessageMask to true if answer is 'yes' or 'no' }
TextWindow (1,1,80,25);                        { Restore window to whole screen }
ClrScr;
END;   BEGIN  END.
```

```
{
SEGMENT     RemSoln;
{ Remove from list of solutions any whose essential masks is contradicted by an answer to a ques-
tion. }

{$I Common1.Typ}

PROCEDURE   RemoveContradictedSolutions (Message,MessageMask:bitstring;MFQ:integer;
                                         VAR Head:solutionpointer);
      VAR   Current, Previous: solutionpointer;
            IntermediateResult1, IntermediateResult2: boolean;

      BEGIN

      Current:= Head;                          { Current set to first element of solutions list }

      WHILE Current <> NIL DO BEGIN
         IntermediateResult1:= NOT (testbit(Message,MFQ) XOR testbit(Current^.classifier,MFQ));
         IntermediateResult2:= testbit(Current^.essentialmask,MFQ);
         IF IntermediateResult1 AND IntermediateResult2 = false THEN  { If the result is false,
                                                                        delete no-longer-possible solution element. }
            IF Current = Head THEN BEGIN                                 { To delete first solution record }
               Head:= Head^.next;                                           { Move Head pointer to next element }
               dispose (Current);                                           { Delete first element }
               Current:= Head;                                   { Set current pointer to what is now the first element }
            END  { of IF }
            ELSE BEGIN                                                   { Delete if non-head element }
               Previous^.next:= Current^.next;                            { By-pass current element }
               dispose (Current);                                         { Delete by-passed element }
               Current:= Previous^.next;                                { Set current pointer equal to next element }
            END  { of ELSE }

         ELSE BEGIN                                  { Not deleting element because still possible solution }
            Previous:= Current;                      { Move previous pointer to current element }
            Current:= Current^.next                  { Move current pointer to next element }
         END;  { of ELSE }                           { IF Current^.next is NIL, WHILE loop is exited }

      END;  { of WHILE }

      END;

      BEGIN  END.
```

```
{
SEGMENT     UsulQust;
             ( Return the number of next unanswered usual question for all the possible solutions. )

($I Common1.Typ)

FUNCTION    FindUsualQuestion (MessageMask:bitstring;CountOfQuestions:integer;
                               Head:solutionpointer): integer;
     VAR    Current: solutionpointer;
            Found: boolean;
            Index, questionnumber: integer;

BEGIN

Current:= Head;                                    { Go to beginning of solutions list }
Found:= False;

WHILE (Current <> NIL) AND (Found = false) DO BEGIN
   Index:= 1;

   WHILE (Index <= CountOfQuestions) AND (Found = false) DO
      IF (testbit(Current^.usualmask,Index) = true) AND (testbit(MessageMask,Index) = false) THEN
BEGIN
         Found:= true;
         Questionnumber:= Index;
      END  ( of IF )
      ELSE
         Index:= Index + 1;

   Current:= Current^.next;
END;  { of WHILE }

IF Found = true THEN
   FindUsualQuestion:= QuestionNumber
ELSE
   FindUsualQuestion:= 0;

END;

BEGIN  END.
```

```
{
SEGMENT     Probable;
             { Calculate the probability of the possible solutions being correct. }

{$I Common1.Typ}
PROCEDURE   PressKey (margin:integer); EXTERNAL;

PROCEDURE   CalculateProbability
(Message,MessageMask:bitstring;CountOfQuestions:integer;Head:solutionpointer);
      VAR   Current: solutionpointer;
             IntermediateResult1, IntermediateResult2, UsualResult: boolean;
             Index: integer;

BEGIN

Current:= Head;                                    { Go to beginning of solutions list }

WHILE Current <> NIL DO BEGIN

   FOR Index:= 1 TO CountOfQuestions DO BEGIN
      IntermediateResult1:= NOT (testbit(Message,Index) XOR testbit(Current^.classifier,Index));
      IntermediateResult2:= testbit(MessageMask,Index) AND testbit(Current^.usualmask,Index);
      UsualResult:= IntermediateResult1 AND IntermediateResult2;
      IF UsualResult = true THEN     { True when the answer to the usual question is correct }
         Current^.usualtrue:= (Current^.usualtrue + 1);
   END;  { of FOR }

Current^.probability:= (Current^.usualtrue / Current^.totalusual) * 100;
Current:= Current^.next                            { Go to next solutions list element }

END;  { of WHILE }

END;

BEGIN  END.
```

```
{
SEGMENT     DispRslt;
             { Display the most likely solution with its probability on screen. }

{$I Common1.Typ}
{$I \PROPAS\PASPC}
{$I \PROPAS\PASDOS}
PROCEDURE   BlankIn (number:integer); EXTERNAL;
PROCEDURE   PressKey (margin:integer); EXTERNAL;
PROCEDURE   FormFileName (filenumber:integer;title:string;VAR FileName:string); EXTERNAL;
PROCEDURE   DisplayTextFile (DiskFile,directory,heading:string;
                             filenumber,firstline:integer); EXTERNAL;
FUNCTION    YesNo (margin:integer): boolean; EXTERNAL;

PROCEDURE   DisplayResult (Head:solutionpointer);
      VAR   Disk: text;
             Current: solutionpointer;
             Line: string[100];
             FileName: string[30];
             Gate: char;
             FrontRunner, Counter: integer;
             Probability: real;

BEGIN
ClrScr;
IF Head = NIL THEN BEGIN
   BlankIn (8);
   writeln (' ':7,'I know of no solution that matches these answers.');
   PressKey (7);
END  { of IF }
ELSE BEGIN
   Current:= Head;                                          { Go to first solution list element }
   FrontRunner:=Current^.keynumber;
   Probability:= Current^.probability;
   WHILE Current <> NIL DO BEGIN
      IF Current^.probability > Probability THEN BEGIN   { Next element is the more probable }
          FrontRunner:= Current^.keynumber;               { Update FrontRunner }
          Probability:= Current^.probability;             { Update Probability }
      END;  { of IF }
      Current:= Current^.next;
   END;  { of WHILE }
   FormFileName (FrontRunner,'solution',FileName);
   assign (Disk,FileName);                              { Connect Disk with FrontRunner on disk file }
   ChDir ('solution');
   reset (Disk);
   BlankIn (3);
   writeln (' ':7,'The most likely solution is;');
   Counter:= 6;
   WHILE NOT eof(Disk) DO BEGIN                             { Write solution text into screen box }
      readln (Disk,Line);
      GoToXY (8,Counter);
      writeln (Line);
      Counter:= Counter + 1
   END;  { of WHILE }
   close (Disk,true);
   ChDir ('\shell');
   GoToXY (7,13);
```

```
   writeln (' ':1,'The probability that the solution is correct is ',trunc(Probability):3,' per-
cent.');
   TextWindow (1,1,80,15);  TextFrame (true);          { Emphasise displayed solution text }
   TextWindow (1,1,80,25);
   GoToXY (5,23);
   writeln ('Press ''W'' for ''Why'' if you want to see an explanation of this solution.');
   GoToXY (5,24);
   writeln ('Press any other key to clear the screen and begin another session.');
   TextWindow (1,22,80,25);
   TextFrame (false);
   Gate:= ConSilent;
   IF (Gate = 'W') OR (Gate = 'w') THEN BEGIN
      TextWindow (1,17,80,21);
      TextFrame (false);
      FormFileName (FrontRunner,'explanation',FileName);
      FileName:= concat('\shell\solution\',FileName);
      IF fstat (FileName) = false THEN BEGIN
          GoToXY (7,2);
          writeln ('No explanation for this solution is on file.');
          PressKey (7);
          ClrScr;
      END  { of IF }
      ELSE BEGIN
          DisplayTextFile (FileName,'question','Explanation',FrontRunner,1);
      END;  { of ELSE }
   Gate:= ConSilent;
      END;  { of IF }
   TextWindow (1,1,80,25);                          { Restore text window to whole screen }
   ClrScr
   END;  { of ELSE }
   END;

   BEGIN  END.
```

```
{
SEGMENT     ClrHeap;
             { Removes the solutions linked list from memory. }

{$I Common1.Typ}

PROCEDURE   ClearHeap (VAR Head:solutionpointer);
      VAR   Current, Succeeding: solutionpointer;

BEGIN

Current:= Head;
WHILE Current <> NIL DO BEGIN
   Succeeding:= Current^.next;
   dispose (Current);
   Current:= Succeeding
END;  { of WHILE }
Head:= NIL

END;
BEGIN  END.
```

Appendix 3. LISTING OF HOUSE.BAS

```
REM    ***    HOUSE ARCHITECTURE WITH CLASSIFIERS    ***
REM    ***    LIST OF VARIABLES    ***
REM    C    Array of integers representing classifiers
REM            first index = classifier number
REM            second index = 16 bit field within each classifier
REM    CR    Array of integers indicating relevant bits in classifier
REM             first index = type A or B or C
REM             second index = classifier number
REM             third index = 16 bit field within each classifier
REM    Q$    Array of diagnostic questions (string variables)
REM    NQ    Number of questions (integer variable)
REM    H$     Array of house architectual types (string variables)
REM    FG    Boolean array for presence of graphic info for each feature
REM    FP    Integer array with bits set for features which are present
REM    FT    Integer array with bits set for features tested
REM    CT    Integer array of threshold value for each classifier
REM    MK    Integer array of words used as bit masks
REM    CV    Integer array of values for each classifier
REM    NC    Number of classifiers
REM    NE    Number of houses for which all features are known

REM    ***    INITIALIZATION    ***
DEFINT B-Z:OPTION BASE 1
WINDOW CLOSE 1
DIM C(60,10), CR(3,60,10), CT(60), CV(60)
DIM  FP(10), FT(10), MK(16), FG(160)
DIM Q$(160),H$(60)
FG(23)=1:FG(24)=1:FG(25)=1
T$(1)="A":T$(2)="B":T$(3)="C"
FOR J = 1 TO 10
    FP(J) = 0:FT(J)=0
NEXT J
FOR I = 1 TO 59
    CV(I) = 0
NEXT I
NE=0:BS=60:CV(BS)=-500
B = MOUSE(0)

REM    ***    READ 16 BIT-MASKS    ***
FOR J = 1 TO 16:READ MK(J):NEXT J

REM    ***    DISPLAY INFORMATION    ***
WINDOW 1,,(40,70)-(472,280),2
PRINT:PRINT TAB(12);"EXPERT SYSTEM FOR HOUSE ARCHITECTURE"
PRINT:PRINT TAB(3); "This program is designed to help you identify the architectural"
PRINT "style of family homes.  The computer will ask you questions about"
PRINT "specific attributes of the house you are examining.  Respond to these"
PRINT "questions by clicking the mouse on the appropriate answer.    If you"
PRINT "are not sure about the proper response, choose the alternative which"
PRINT "is most nearly correct."

REM    ***    READ LIST OF QUESTIONS    ***
L1:            READ N:IF N = 999 THEN L2
READ Q$(N):NQ=N:GOTO L1
```

```
REM    ***     READ CLASSIFIERS & RELEVANCE MASKS     ***
L2:             READ N:IF N = 999 THEN L3
READ H$(N), CT(N):NC = N
FOR H = 1 TO 3
C1:                 READ N:IF N=999 THEN C2
    NA = ABS(N)
    J = INT((NA-1)/16) + 1
    K = ((NA-1) MOD 16) + 1
    CR(H,NC,J) = CR(H,NC,J) OR MK(K)
    IF N > 0 THEN C(NC,J) = C(NC,J) OR MK(K)
    GOTO C1
C2:             NEXT H
GOTO L2

L3:             PRINT:PRINT TAB(18);"CLICK THE MOUSE TO BEGIN"
D2:             IF MOUSE(0) = 0 THEN D2
 WINDOW CLOSE 1

REM    ***     CREATE MENUS     ***
MENU 6,0,1,"Debug"
MENU 6,1,1,"Message"
MENU 6,2,1,"Classifiers"
MENU 7,0,1,"Crosstabs"
MENU 7,1,1,"House Types"
MENU 7,2,1,"Features"
ON MENU GOSUB PME
MENU ON

REM    ***     DETERMINE DATE OF CONSTRUCTION     ***
Q$ = "When was the house built ?"
WINDOW 1,Q$,(120,60)-(390,310),1
    BUTTON 1,1,"before 1820",(80,20)-(180,40)
    BUTTON 2,1,"1820 to 1880",(80,65)-(180,85)
    BUTTON 3,1,"1880 to 1940",(80,110)-(180,130)
    BUTTON 4,1,"after 1940",(80,155)-(180,175)
    BUTTON 5,1,"unknown",(80,200)-(180,220)
    GOSUB QUERY
    IF B = 5 THEN L4
    J=1:K=B:FP(J) = FP(J) OR MK(K)
    FOR K=1 TO 16:FT(J) = FT(J) OR MK(K):NEXT K
WINDOW CLOSE 1
FOR K = 1 TO 4:GOSUB ADJ:NEXT K

REM    ***     GET SLOPE OF THE ROOF     ***
L4:             Q$ = "What is the slope of the roof ?"
WINDOW 1,Q$,(90,60)-(420,290)
    BUTTON 1,1,"flat",(60,20)-(330,45),2
    BUTTON 2,1,"less than 30 degrees",(60,60)-(330,85),2
    BUTTON 3,1,"30 to 45 degrees",(60,100)-(330,125),2
    BUTTON 4,1,"more than 45 degrees",(60,140)-(330,165),2
    BUTTON 5,1,"combination of the above",(60,180)-(330,200),2
    GOSUB QUERY
    J=1:K=B+4:FP(J) = FP(J) OR MK(K)
WINDOW CLOSE 1
FOR K = 5 TO 9:GOSUB ADJ:NEXT K
```

```
REM    ***    COMPOSITION OF EXTERIOR WALLS    ***
Q$ = "The exterior walls are made of ?"
WINDOW 1,Q$,(90,60)-(420,300)
    BUTTON 1,1,"wood",(60,20)-(260,45),2
    BUTTON 2,1,"stone",(60,55)-(260,80),2
    BUTTON 3,1,"brick",(60,90)-(260,115),2
    BUTTON 4,1,"stucco or adobe",(60,125)-(260,150),2
    BUTTON 5,1,"combination of the above",(60,160)-(260,185),2
    BUTTON 6,1,"other",(60,195)-(260,220),2
    GOSUB QUERY
    J=1:K=B+9:FP(J) = FP(J) OR MK(K)
WINDOW CLOSE 1
FOR K=10 TO 15:GOSUB ADJ:NEXT K

REM    ***    ROOF-WALL JUNCTION    ***
Q$ = "Junction between roof and exterior wall ?"
WINDOW 1,Q$,(40,55)-(465,315)
    BUTTON 1,1,"little or no overhang (no eaves)",(50,30)-(380,50),2
    BUTTON 2,1,"exterior wall extends above roof (parapet)",(50,60)-(380,80),2
    BUTTON 3,1,"slight overhang with exposed rafters",(50,90)-(380,110),2
    BUTTON 4,1,"slight overhang with boxed eaves",(50,120)-(380,140),2
    BUTTON 5,1,"wide overhang with exposed rafters",(50,150)-(380,170),2
    BUTTON 6,1,"wide overhang with boxed eaves",(50,180)-(380,200),2
    BUTTON 7,1,"other",(50,210)-(380,230),2
    GOSUB QUERY:BIT = B + 15
    J = INT((BIT-1)/16) + 1
    K = ((BIT-1) MOD 16) + 1
    FP(J) = FP(J) OR MK(K)
    J=2:FOR K =1 TO 6:FT(J) = FT(J) OR MK(K):NEXT K
WINDOW CLOSE 1
J=1:K=16:GOSUB ADJ
J=2:FOR K=1 TO 6:GOSUB ADJ:NEXT K
GOSUB TAP

REM    ***    MAIN LOOP    ***
REM    ***    GET NEXT QUESTION    ***
L9:            IF CV(M1) > CT(M1) THEN L40
IF CV(BS)+470 > CV(M1) THEN M1=BS:GOTO L40
H = 0
L10:         H = H + 1:IF H > 3 THEN L20
J = 1
L11:         J = J + 1:IF J > 10 THEN L10
               Q=CR(H,M1,J):IF Q=0 THEN L11
               K = 0
L12:         K = K + 1:IF K > 16 THEN L11
               T = Q AND MK(K) AND NOT FT(J)
               IF T = 0 THEN L12
N = 16*(J-1) + K
Q$ = Q$(N)
GOSUB YN
FT(J) = FT(J) OR MK(K)
IF B=1 THEN FP(J) = FP(J) OR MK(K)
GOSUB ADJ
GOSUB TAP
GOTO L9
```

```
L20:          CV(M1) = CV(M1)-500
IF CV(M1) > CV(BS) THEN BS=M1
NE = NE + 1:IF NE>5 THEN L30
WINDOW 3,"BEST SO FAR",(320,170)-(500,220)
MOVETO 20,20:PRINT H$(BS);SPC(2);CV(BS)+500;
GOSUB TAP
GOTO L9

L30:          IF CV(R1)+500 > 15 THEN M1=R1:GOTO L40
WINDOW CLOSE 2:WINDOW CLOSE 3
WINDOW 1,,(80,120)-(430,220),2
MOVETO 30,50:PRINT "This house does not fit any of my categories"
GOTO TRAP

L40:          WINDOW CLOSE 2:WINDOW CLOSE 3
WINDOW 1,,(80,120)-(430,220),2
MOVETO 30,50:PRINT "The architectural style is ";H$(M1);

TRAP:         GOTO TRAP

              REM    ***    ADJUST CLASSIFIER VALUES    ***
ADJ:          FOR I = 1 TO NC
   IF CV(I) = -99 THEN A3
   TR = NOT(C(I,J) XOR FP(J))
   H = 1:RB = CR(H,I,J) AND MK(K)
   IF RB = 0 THEN A1
   TB = TR AND RB
   IF TB = 0 THEN CV(I)=-99 ELSE CV(I)=CV(I)+5
   GOTO A3
A1:              H = 2:RB = CR(H,I,J) AND MK(K)
   IF RB = 0 THEN A2
   TB = TR AND RB
   IF TB = 0 THEN CV(I)=CV(I)-5 ELSE CV(I)=CV(I)+5
   GOTO A3
A2:                 H = 3:RB = CR(H,I,J) AND MK(K)
    IF RB = 0 THEN A3
    TB = TR AND RB
    IF TB = 0 THEN CV(I)=CV(I)-1 ELSE CV(I)=CV(I)+5
A3:           NEXT I
RETURN

REM    ***    SUBROUTINE TO CHECK DESKTOP    ***
QUERY:       D = DIALOG(0): IF D <> 1 THEN QUERY
B = DIALOG(1):BUTTON B,2
RETURN

REM    ***    SUB TO ASK YES-NO QUESTION    ***
YN:           IF FG(N) = 0 THEN Y1
WINDOW 4,,(70,135)-(260,235),3
TQ = N-22:ON TQ GOSUB G1,G2,G3,G4,G5
Y1:           WINDOW 1,"Does the house have",(30,50)-(480,130),1
Y2:               NT=0:NL = LEN(Q$):LB = 200 - NL*3
    MOVETO LB,20:PRINT Q$;" ?";
    BUTTON 1,1,"Yes",(150,40)-(190,60)
    BUTTON 2,1,"No",(230,40)-(270,60)
```

```
Y3:D = DIALOG(0):NT = NT + 1:IF NT > 600 THEN Y2
    IF D <> 1 THEN Y3
    B = DIALOG(1):BUTTON B,2
WINDOW CLOSE 1:WINDOW CLOSE 4
RETURN

REM    ***    PROCESS MENU SELECTION    ***
PME:          M = MENU(0):IF M = 7 THEN CRT
IF M <> 6 THEN RETURN
MENU 6,S,1:S = MENU(1)
MENU 6,S,2:MENU
ON S GOSUB MESS, CLAS
RETURN

REM    ***    DISPLAY MESSAGE    ***
MESS:        WINDOW CLOSE 2
WINDOW 4,"",(20,35)-(490,305),2
PRINT:PRINT TAB(22);"CURRENT MESSAGE"
PRINT TAB(13);"BITS TESTED";SPC(14);"BITS SET"
FOR J2 = 1 TO 10:PRINT TAB(10);
    FOR K2 = 1 TO 16:T = FT(J2) AND MK(K2)
        IF T <> 0 THEN PRINT "1"; ELSE PRINT "0";
    NEXT K2:PRINT SPC(6);
    FOR K2 = 1 TO 16:T = FP(J2) AND MK(K2)
        IF T <> 0 THEN PRINT "1"; ELSE PRINT "0";
    NEXT K2:PRINT
NEXT J2
PRINT:PRINT TAB(20);"HIT ANY KEY TO CONTINUE";
MZ:           R$=INKEY$:IF R$ = "" THEN MZ
WINDOW CLOSE 4
RETURN

CLAS:        WINDOW CLOSE 2
WINDOW 4,"",(10,35)-(500,320),2
PRINT:PRINT TAB(20);"LEADING ACTIVE CANDIDATE"
PRINT TAB(22);H$(M1)
PRINT TAB(7);"Classifier";SPC(14);"Mask A";SPC(13);"Masks B & C"
FOR J3 = 1 TO 10
    PRINT TAB(3);
    FOR K3 = 1 TO 16
        T = C(M1,J3) AND MK(K3)
        IF T <> 0 THEN PRINT "1"; ELSE PRINT "0";
    NEXT K3:PRINT SPC(4);
    FOR K3 = 1 TO 16
        T = CR(1,M1,J3) AND MK(K3)
        IF T <> 0 THEN PRINT "1"; ELSE PRINT "0";
    NEXT K3:PRINT SPC(4);
    FOR K3 = 1 TO 16
        T = (CR(2,M1,J3) OR CR(3,M1,J3)) AND MK(K3)
        IF T <> 0 THEN PRINT "1"; ELSE PRINT "0";
    NEXT K3:PRINT
NEXT J3
PRINT:PRINT TAB(21);"HIT ANY KEY TO CONTINUE";
CZ:           R$=INKEY$:IF R$ = "" THEN CZ
WINDOW CLOSE 4
RETURN
```

```
TAP:          M1=60:M2=60:M3=60:CV(M1) = -500
FOR I = 1 TO NC
    IF CV(I) > CV(M1) THEN M3=M2:M2=M1:M1=I:GOTO T1
    IF CV(I) > CV(M2) THEN M3=M2:M2=I:GOTO T1
    IF CV(I) > CV(M3) THEN M3=I
T1:            NEXT I
WINDOW 2,"ACTIVE CANDIDATES",(20,260)-(480,330),1
 MOVETO 40,15
 PRINT CV(M1);"/";CT(M1);SPC(3);H$(M1);
 MOVETO 40,35
 PRINT CV(M2);"/";CT(M2);SPC(3);H$(M2);
 MOVETO 40,55
 PRINT CV(M3);"/";CT(M3);SPC(3);H$(M3);
 RETURN

REM    ***    PROCESS MENU INTERRUPT    ***
CRT:          WINDOW CLOSE 2
MENU:NX=0:IF MENU(1) = 2 THEN PM5
IF MENU(1) <> 1 THEN RETURN
PM1:          WINDOW 3,"WHICH HOUSE TYPE ?",,1
MXI=12:IF MXI>NC-NX THEN MXI=NC-NX
FOR II=1 TO MXI
    BUTTON II,1,H$(II+NX),(40,20*II-5)-(240,20*II+10),2
NEXT II:IF NC-NX<13 THEN PM2
MXI=24:IF MXI>NC-NX THEN MXI=NC-NX
FOR II = 13 TO MXI
    BUTTON II,1,H$(II+NX),(260,20*II-245)-(470,20*II-230),2
NEXT II
PM2:          IF NX+24<NC THEN BUTTON 25,1,"MORE",(80,260)-(200,280),3
BUTTON 26,1,"EXIT",(300,260)-(420,280),3
PM3:          IF DIALOG(0) <> 1 THEN PM3
KCH=DIALOG(1):BUTTON KCH, 2
IF KCH=25 THEN NX=NX+24:GOTO PM1
IF KCH=26 THEN PM9
HC=KCH+NX:R$="RELEVANT FEATURES FOR "+H$(HC)
WINDOW 3,R$,,1
NF=0
FOR HH = 1 TO 3
    FOR JJ = 1 TO 10
        FOR KK = 1 TO 16
            TCB = CR(HH,HC,JJ) AND MK(KK)
            IF TCB = 0 THEN PM4
            NF=NF+1:NN=(JJ-1)*16 + KK
            PRINT SPC(1);T$(HH);SPC(3);Q$(NN)
            IF NF = 16 THEN GOSUB PAU
PM4: NEXT KK
    NEXT JJ
NEXT HH
BUTTON 1,1,"EXIT",(430,270)-(490,290),3
PM4A:         IF DIALOG(0) <> 1 THEN PM4A
KCH=DIALOG(1):IF KCH<>1 THEN PM4A
BUTTON 1,2:GOTO PM9
```

```
PM5:        WINDOW 3,"WHICH FEATURE ?",,1
MXI=12:IF MXI>NQ-NX THEN MXI=NQ-NX
FOR II=1 TO MXI
   BUTTON II,1,Q$(II+NX),(10,20*II-5)-(480,20*II+10),2
NEXT II
IF NX+12<NQ THEN BUTTON 13,1,"MORE",(80,260)-(200,280),3
BUTTON 14,1,"EXIT",(300,260)-(420,280),3

PM6:        IF DIALOG(0) <> 1 THEN PM6
KCH = DIALOG(1):BUTTON KCH,2
IF KCH=13 THEN NX=NX+12:GOTO PM5
IF KCH=14 THEN PM9
FC=KCH+NX:WINDOW 3,Q$(FC),,1
BUTTON 1,1,"EXIT",(430,270)-(490,290),3
JJ = INT((FC-1)/16) + 1
KK = ((FC-1) MOD 16) + 1
FOR HH = 1 TO 3
    FOR II = 1 TO NC
        TBC = CR(HH,II,JJ) AND MK(KK)
        IF TBC = 0 THEN PM6A
        IF H$(II) = PH$ THEN PM6A
        PRINT SPC(1);T$(HH);SPC(1);H$(II)
        PH$ = H$(II)
PM6A:           NEXT II
NEXT HH
PM7:        IF DIALOG(0) <> 1 THEN PM7
KCH=DIALOG(1):IF KCH<>1 THEN PM7
BUTTON 1,2
PM9:         WINDOW CLOSE 3
RETURN

PAU:        BUTTON 1,1,"MORE",(430,270)-(490,290),3
PA1:        IF DIALOG(0)<>1 THEN PA1
KCH=DIALOG(1):IF KCH<>1 THEN PA1
BUTTON 1,2:BUTTON CLOSE 1:RETURN

REM    ***    GRAPHICS FOR BOARD-AND-BATTEN FRONT DOOR    ***
G1:            LINE (75,13)-(117,87),,B
FOR I = 1 TO 6
   LINE (75+I*6,13)-(75+I*6,87)
NEXT I
RETURN

REM    ***    GRAPHICS FOR RECESSED PANELS IN FRONT DOOR    ***
G2:            LINE (75,12)-(117,88),,B
FOR I = 1 TO 4
   LINE (84,6+16*I)-(92,16+16*I),,B
   LINE (98,6+16*I)-(106,16+16*I),,B
NEXT I
RETURN
```

```
REM    ***    PILASTERS ON EACH SIDE OF FRONT DOOR    ***
G3:           LINE (80,20)-(112,80),,B
LINE (69,20)-(74,80),,B
LINE (118,20)-(123,80),,B
LINE (67,80)-(76,83),,B
LINE (116,80)-(125,83),,B
LINE (78,80)-(114,83),,B
LINE (67,17)-(76,20),,B
LINE (116,17)-(125,20),,B
RETURN

          REM    ***    MASKS FOR BIT MANIPULATION    ***
          DATA &H8000,&H4000,&H2000,&H1000
          DATA &H800,&H400,&H200,&H100
          DATA &H80,&H40,&H20,&H10
          DATA &H8,&H4,&H2,&H1

          REM    ***    LIST OF ARCHITECTURAL FEATURES    ***
          REM    ***    DATE OF CONSTRUCTION    ***
          DATA 1,"before 1820"
          DATA 2,"1820 to 1880"
          DATA 3,"1880 to 1940"
          DATA 4,"after 1940"
          REM    ***    ROOF SLOPE    ***
          DATA 5,"flat roof"
          DATA 6,"low slope roof"
          DATA 7,"moderate slope roof"
          DATA 8,"steep slope roof"
          DATA 9,"several different roof slopes"
          REM    ***    COMPOSITION OF EXTERIOR WALLS    ***
          DATA 10,"wood exterior"
          DATA 11,"stone exterior"
          DATA 12,"brick exterior"
          DATA 13,"stucco or adobe exterior"
          DATA 14,"combination of wood and masonry or stucco"
          DATA 15,"unconventional exterior cladding"
          REM    ***    JUNCTION OF ROOF AND EXTERIOR WALL    ***
          DATA 16,"no roof overhang"
          DATA 17,"parapet at roof-line"
          DATA 18,"slight overhang with exposed rafters"
          DATA 19,"slight overhang with boxed eaves"
          DATA 20,"wide overhang with exposed rafters"
          DATA 21,"wide overhang with boxed eaves"
          DATA 22,"unusual roof-wall junction"
          REM    ***    ENTRYWAY    ***
          DATA 23,"a board-and-batten front door"
          DATA 24,"six or eight recessed panels in the front door"
          DATA 25,"pilasters on each side of the front door"
          DATA 26,"a pediment (crown) above the front door"
          DATA 27,"a front door split into upper and lower halves"
          DATA 28,"more than one external front door"
          DATA 29,"paired entry doors"
          DATA 30,"a semi-circular or elliptical fanlight over the front door"
          DATA 31,"slender columns supporting a forward-extending pediment"
          DATA 32,"small rectangular windows on either side of the front door"
          DATA 33,"a round-arched front doorway"
```

```
DATA 34,"ornate decorations on or around the front door"
DATA 35,"a recessed or obscurred main entrance"
DATA 36,"a row of small, rectangular glass panes above the front door"
DATA 37,"an entryway dominated by a large, formal portico (entry porch)"
DATA 38,"a fancy metal canopy extending forward above the front door"
DATA 39,"cantilevered (unsupported) section of house, roof, or balcony"
REM    ***    FRONT PORCH    ***
DATA 40,"a full-height (ground to roof-line) entry porch"
DATA 41,"a large, one-story front porch"
DATA 42,"a porch wrapping around more than one side of the house"
DATA 43,"classical (Roman) columns"
DATA 44,"porch roof supported by heavy, squared columns"
DATA 45,"porch roof supported by delicate, turned columns"
DATA 46,"spindled porch railings"
DATA 47,"porch roof supports which look like bundles of sticks flared at the top"
DATA 48,"a second-story porch (balcony) with balustrade"
DATA 49,"lacy spandrels (gingerbread) along porch roof-line"
DATA 50,"no front porch"
DATA 51,"porch roof supported by plain, slender, wooden columns"
DATA 52,"a porch which covers the entire front facade"
DATA 53,"rough-hewn porch supports, roof beams, and window lintels"
DATA 54,"visor-shaped, horizontal extension along front of house"
REM    ***    WINDOWS    ***
DATA 55,"one or more palladian windows"
DATA 56,"one or more oriel windows"
DATA 57,"one or more bay windows"
DATA 58,"a large, rectangular picture window"
DATA 59,"metal casement windows set flush with exterior wall"
DATA 60,"a window with a large pane bounded by many smaller panes"
DATA 61,"double-hung windows with multi-pane glazing"
DATA 62,"windows grouped in side-by-side pairs"
DATA 63,"tall, narrow windows with multi-pane glazing"
DATA 64,"three or more contiguous windows"
DATA 65,"upper-story windows less elaborate than first-story ones"
DATA 66,"horizontal window openings with many rectangular panes"
DATA 67,"windows constructed of glass blocks"
DATA 68,"windows with many, small, diamond-shaped panes"
DATA 69,"windows with blank lower panes and patterned upper panes"
DATA 70,"segmental arches above windows"
DATA 71,"rounded arches above windows"
DATA 72,"pointed arches above windows"
DATA 73,"label molding above windows"
DATA 74,"hood molding above windows"
DATA 75,"bracketed awnings above windows"
DATA 76,"pedimented windows"
DATA 77,"small iron balconies at the base of window openings"
DATA 78,"flat lintels above window openings"
DATA 79,"round or elliptical windows"
REM    ***    GENERAL ARCHITECTURAL FEATURES    ***
DATA 80,"an irregular roof shape"
DATA 81,"a second story which partially overhangs the first story"
DATA 82," a round or polygonal tower at one corner of the facade"
DATA 83,"symmetrically placed windows about a centered front door"
DATA 84,"two or more front-facing gables"
DATA 85,"a prominent gable on the front facade"
DATA 86,"upper and lower stories with different exteriors"
```

```
DATA 87,"one or more pedimented dormers"
DATA 88,"a sculptured (fancy shape) dormer"
DATA 89,"ground to roof-line pilasters"
DATA 90,"cross gables (90 degree angle from each other)"
DATA 91,"a gambrel roof (dual pitched gables)"
DATA 92,"a mansard roof (hipped with differing upper and lower slopes)"
DATA 93,"a hipped roof"
DATA 94,"flared eaves"
DATA 95,"rounded ceramic roof tiles"
DATA 96,"flat ceramic roof tiles"
DATA 97,"wooden roof shingles"
DATA 98,"a thatched or false-thatched roof"
DATA 99,"exterior walls arranged in an octagonal shape"
DATA 100,"wooden shingles covering a curved or rounded exterior wall"
DATA 101,"a prominent round tower with a conical roof"
DATA 102,"a prominent square, hexagonal, or octagonal tower"
DATA 103,"a long, sprawling floor plan (ranch style)"
DATA 104,"an attached garage"
DATA 105,"wide masonry columns supporting the house"
DATA 106,"a simple rectangular floor plan and a side-gabled roof"
DATA 107,"a multi-directional shed roof"
DATA 108,"three different floor levels in a two-story house (split level)"
DATA 109,"a central wing projecting forward from the front facade"
DATA 110,"two small wings at either end with a recessed central entryway"
DATA 111,"two or more stories"
DATA 112,"wall cladding which extends up into the gable without a break"
DATA 113,"gradually curved vertical corners"
DATA 114,"a long horizontal ribbon of connecting windows"
REM    ***    ROOF-LINE ORNAMENTATION    ***
DATA 115,"ornamental brackets under the eaves"
DATA 116,"modillions or dentils under the eaves"
DATA 117,"decorated verge boards"
DATA 118,"trusses in the gables"
DATA 119,"false beams at the end of the gables"
DATA 120,"spindlework detailing (gingerbread) in the gables"
DATA 121,"decorative terra cotta panels on the face of the gables"
DATA 122,"decorative half-timbering in the gables"
DATA 123,"a roof-line balustrade"
DATA 124,"a wide band of trim under the eaves"
DATA 125,"a small, horizontal ledge (coping) at the roof line"
DATA 126,"parapeted gables without half-timbering"
DATA 127, "fancy, ornate decorative detailing along the roof-line"
DATA 128, "horizontal rectangular openings just below the roof-line"
DATA 129,"an eyebrow dormer"
REM    ***    EXTERIOR WALL DECORATIONS    ***
DATA 130,"decorative half-timbering on upper-stories"
DATA 131,"exterior details which avoid a smooth-walled appearance"
DATA 132,"patches of patterned or textured shingles"
DATA 133,"masonry walls with patterned brickwork or stonework"
DATA 134,"brackets accentuating simulated upper-story overhang"
DATA 135,"wood shingle wall cladding"
DATA 136,"quoins decorating corners of masonry exterior"
DATA 137,"a belt course on masonry exterior"
DATA 138,"garlands or other floral decorations on exterior"
DATA 139,"rectangular shutters along side the windows"
DATA 140,"patterned stickwork decorations on exterior walls"
```

```
DATA 141,"wooden roof beams projecting from top of exterior wall"
DATA 142,"zigzag, chevron, or lozenge decorations on exterior"
DATA 143,"cornice and facade detailing emphasizing horizontal lines"
DATA 144,"dormer windows on the steep lower slope of a mansard roof"
DATA 145,"hipped dormer"
DATA 146,"shed dormer"
DATA 147,"exterior detailing with a vertical emphasis"
DATA 148,"small towers and other vertical projections on the roof"
DATA 149,"floor-to-ceiling windows"
REM    ***    ROOF-TOP DECORATIONS    ***
DATA 150,"a roof-top cupola"
DATA 151,"a pinnacle on the roof"
DATA 152,"castellations on the roof"
DATA 153,"metal roof cresting"
DATA 154,"spires projecting above one or more gables"
DATA 155,"a large onion-shaped (Turkish) dome on the roof"
DATA 156,"decorative chimney pots"
DATA 157,"a prominent, tall, decorative chimney"
DATA 158,"a wide, flat, plain chimney"
DATA 159,"a roof-top balustrade"
DATA 160,"large chimneys at both ends of the house"
DATA 999

REM    ***    LIST OF CLASSIFIERS    ***
DATA 1, "Queen Anne Victorian", 35
DATA  3,8,80,-83,999
DATA  41,131,999
DATA  14,42,43,45,48,49,55,57,60,69,82,84,116,117,157
DATA  118,120,121,122,132,133,134,153,999
DATA  2, "Tudor", 30
DATA  3,8,-41,999
DATA  90,157,112,999
DATA  12,23,33,63,64,98,126,130,156,999
DATA  3, "Italian Renaissance", 30
DATA  3,21,93,999
DATA  6,-10,115,999
DATA  25,26,65,71,76,109,110,136,137,999
DATA  4, "Italian Renaissance", 30
DATA  3,5,11,999
DATA  83,116,123,999
DATA  25,26,43,44,65,71,76,136,137,999
DATA  5, "Northern Postmedieval English", 25
DATA  1,8,16,999
DATA  10,23,68,106,111,999
DATA  81,135,157,999
DATA  6, "Southern Postmedieval English", 25
DATA  1,8,16,999
DATA  23,68,106,999
DATA  12,111,160,999
DATA  7, "Urban Dutch Colonial", 30
DATA  1,999
DATA  8,12,17,106,-111,146,999
DATA  27,61,91,160,999
DATA  8, "Rural Dutch Colonial", 30
DATA  1,999
DATA  -10,106,999
```

```
DATA  27,61,91,94,-111,146,999
DATA  9, "Urban French Colonial", 35
DATA  50,999
DATA  1,8,13,106,-111,999
DATA  17,28,29,30,63,94,999
DATA  10, "Rural French Colonial", 30
DATA  1,999
DATA  8,13,41,999
DATA  29,51,63,92,105,999
DATA  11, "Spanish Colonial", 25
DATA  6,-10,999
DATA  1,28,-34,95,999
DATA  23,48,999
DATA  12, "Spanish Colonial", 30
DATA  5,-10,17,999
DATA  1,28,-34,-111,999
DATA  23,41,141,999
DATA  13, "Georgian", 40
DATA  1,-30,-62,111,999
DATA  7,19,61,83,999
DATA  24,25,26,36,54,76,85,87,89,91,106,116,136,137,159,999
DATA  14, "Adam", 40
DATA  1,-62,999
DATA  7,19,30,61,999
DATA  24,25,31,32,55,77,78,83,85,116,123,127,137,138,139,999
DATA  15, "Early Classical Revival", 35
DATA  37,43,83,999
DATA  1,7,19,30,40,999
DATA  24,25,48,79,116,123,999
DATA  16, "Greek Revival", 35
DATA  2,-30,124,999
DATA  6,19,32,43,999
DATA  25,36,40,48,52,128,999
DATA  17, "Gothic Revival", 40
DATA  2,8,999
DATA  10,18,41,72,90,112,999
DATA  56,57,73,84,85,102,117,118,999
DATA  18, "Gothic Revival", 30
DATA  2,-10,-13,999
DATA  102,152,999
DATA  32,72,73,151,999
DATA  19, "Italianate", 35
DATA  2,999
DATA  6,111,115,999
DATA  25,29,32,41,43,62,72,74,75,85,102,128,150,999
DATA  20, "Egyptian Revival", 25
DATA  2,-10,47,999
DATA  6,111,999
DATA  25,62,78,115,116,999
DATA  21, "Oriental Revival", 25
DATA  -10,93,155,999
DATA  2,72,133,999
DATA  115,999
DATA  22, "Swiss Chalet Revival", 25
DATA  6,10,20,999
DATA  2,127,999
```

```
DATA  48,140,999
DATA  23, "Octagon", 25
DATA  2,99,999
DATA  6,21,93,999
DATA  41,42,115,150,999
DATA  24, "Second Empire Victorian", 35
DATA  2,144,999
DATA  9,92,115,999
DATA  25,29,41,57,62,71,74,75,79,102,109,136,137,150,999
DATA  25, "Stick Victorian", 35
DATA  10,140,999
DATA  2,8,18,90,999
DATA  41,57,75,94,102,118,999
DATA  26, "Shingle Victorian", 35
DATA  3,135,999
DATA  7,41,80,-83,999
DATA  55,57,64,82,100,129,145,999
DATA  27, "Richardsonian Romanesque", 30
DATA  3,11,999
DATA  71,-83,101,999
DATA  43,64,126,129,151,999
DATA  28, "Folk Victorian", 30
DATA  3,10,999
DATA  7,41,49,999
DATA  19,45,46,83,115,999
DATA  29, "Colonial Revival", 33
DATA  61,999
DATA  3,7,19,999
DATA  25,26,31,32,54,57,62,71,81,83,87,91,109,116,135,136,146,999
DATA  30, "Neoclassical", 33
DATA  40,43,999
DATA  3,7,19,61,83,999
DATA  25,26,32,116,123,999
DATA  31, "Chateauesque", 35
DATA  3,8,-10,999
DATA  157,999
DATA  11,33,71,74,101,126,151,154,999
DATA  32, "Beaux Arts", 25
DATA  3,92,138,999
DATA  11,83,87,999
DATA  38,76,77,136,999
DATA  33, "Beaux Arts", 20
DATA  3,5,138,999
DATA  11,43,83,123,999
DATA  71,136,999
DATA  34, "Beaux Arts", 20
DATA  3,6,93,138,999
DATA  11,83,999
DATA  71,136,999
DATA  35, "French Eclectic", 30
DATA  3,8,-10,999
DATA  19,-90,93,999
DATA  33,77,83,94,145,157,999
DATA  36, "French Eclectic", 35
DATA  3,8,-10,101,999
DATA  19,-90,93,999
```

```
DATA  33,64,81,94,130,157,999
DATA  37, "Mission", 25
DATA  3,999
DATA  7,13,95,999
DATA  17,20,33,41,44,54,88,102,999
DATA  38, "Spanish Eclectic", 25
DATA  3,999
DATA  6,13,16,-83,95,999
DATA  23,33,34,71,77,999
DATA  39, "Monterey", 25
DATA  6,-93,111,999
DATA  3,48,999
DATA  51,86,97,999
DATA  40, "Pueblo Revival", 25
DATA  3,5,999
DATA  13,17,141,999
DATA  23,53,-83,999
DATA  41, "Pueblo Revival", 25
DATA  4,5,999
DATA  13,17,141,999
DATA  23,53,-83,999
DATA  42, "Prairie", 35
DATA  3,6,999
DATA  21,111,143,999
DATA  34,41,44,64,93,95,119,145,158,999
DATA  43, "Craftsman", 30
DATA  3,6,999
DATA  20, 999
DATA  41,44,64,-93,94,115,118,119,130,146,999
DATA  44, "Art Moderne", 30
DATA  3,5,999
DATA  13,-83,125,143,999
DATA  67,79,113,999
DATA  45, "Art Deco", 20
DATA  3,5,142,999
DATA  13,999
DATA  127,147,148,999
DATA  46, "International", 35
DATA  -1,-2,5,-83,999
DATA  -17,59,-125,999
DATA  13,35,39,51,114,149,999
DATA  47, "Minimal Traditional", 25
DATA  4,-5,-8,999
DATA  16,85,999
DATA  -111,157,999
DATA  48, "Ranch", 20
DATA  4,103,999
DATA  6,-16,-83,-111,999
DATA  58,104,139,999
DATA  49, "Split-Level", 15
DATA  4,108,999
DATA  6,-16,999
DATA  14,104,999
DATA  50, "Contemporary", 20
DATA  4,999
DATA  6,20,999
```

```
DATA  14,147,149,999
DATA  51, "Shed", 15
DATA  4,107,999
DATA  16,999
DATA  10,35,999
DATA  52, "Neoeclectic Mansard", 25
DATA  4,9,92,999
DATA  -10,63,71,999
DATA  29,30,50,999
DATA  53, "Neocolonial", 25
DATA  4,-5,999
DATA  7,61,111,999
DATA  81,83,139,999
DATA  54, "Neo-French", 25
DATA  4,8,93,999
DATA  -10,63,999
DATA  71,-83,-104,999
DATA  55, "Neo-Tudor", 30
DATA  4,8,999
DATA  -10,63,-83,90,999
DATA  64,80,84,130,157,999
DATA  56, "Neo-Mediterranean", 25
DATA  4,6,-10,999
DATA  71,999
DATA  12,13,20,21,29,95,-104,999
DATA  57, "Neoclassical Revival", 25
DATA  4,40,999
DATA  37,43,83,-104,999
DATA  48,124,136,999
DATA  58, "Neo-Victorian", 35
DATA  4,10,-50,111,999
DATA  45,46,-83,131,999
DATA  42,49,61,999
DATA  999
END
```

(Frey, 1986b)

REFERENCES

Items are listed and referenced according to the Harvard convention. However, in the case of the three works by Wittgenstein, I have followed the custom of referencing the item by means of its intials. In text references page numbers appear after the date and separated from it by a colon.

Aalto, A. (1940) **The Humanizing of Architecture**, in THE TECHNOLOGY REVIEW. Vol 36.

Akin, O. (1978) **How Do Architects Design** in Latombe, J.C. (Ed.) Artificial Intelligence and Pattern Recognition in Computer Aided Design, North Holland, New York.

Akin, O. (1986) **Psychology of Architectural Design**, Pion Ltd, London.

Akman, V., ten Hagen, P.J.W. and Tomiyama T. (1990) **A Fundamental and Theoretical Framework for an Intelligent CAD System** in COMPUTER AIDED DESIGN. Vol 22, No 6.

Alexander, C. (1964) **Notes on the Synthesis of Form**, McGraw Hill, New York.

Alexander, C. (1979) **The Timeless Way of Building**, Oxford University Press, New York.

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I. and Angel, S. (1977) **A Pattern Language: Towns, Buildings, Construction**, Oxford University Press, New York.

Alty, J.L. and Coombs, M.J. (1984) **Expert Systems**, National Computing Centre Publications, Manchester.

Archer, L.B. (1969) **The Structure of the Design Process** in Broadbent, G., and Ward, A. (Eds.) Design Methods in Architecture, Lund Humphries Publishers Ltd, London.

Archer, L.B. (1970) **An Overview of the Structure of the Design Process** in Moore, G.T. (Ed.) Emerging Methods in Environmental Design and Planning, MIT Press, Cambridge, MA.

Arnold, M. (1865) **Functions of Criticism at the Present Time**, Macmillan, London.

Asimow, M. (1962) **Introduction to Design**, Prentice Hall, Englewood Cliffs, NJ.

Austin, D. (1968) **2001: A Space Odyssey**, in FILMS AND FILMING. Vol 14, No 10.

Barr, A. and Feigenbaum, E.A. (Eds.) (1981) **The Handbook of Artificial Intelligence**, Pitman Books Ltd, London.

Bennett, J., Buchanan, B.G., Cohen, P. and Fisher, F. (1982) **Applications Oriented AI Research: Science** in Barr, A., and Feigenbaum, E.A. (Eds.) Handbook of Artificial Intelligence. Vol II., W. Kaufman Inc, Los Altos, CA.

Bijl, A. (1986) **Designing with Words and Pictures in a Logic Modelling Environment** in Pipes, A. (Ed.) Computer-Aided Architectural Design Futures, Butterworths, London.

Block, I. (1987) **Wittgenstein** in Turner, R. (Ed.) Thinkers of the Twentieth Century, St James Press, Chicago, IL.

Bobrow, D.G., and Hayes, P.J. (1985) **Artificial Intelligence: Where are We?**, in ARTIFICIAL INTELLIGENCE. Vol 25.

Boden, M.A. (1977) **Artificial Intelligence and Natural Man**, MIT Press, London.

Booker, L.B., Holland, J.H., and Goldberg, D.E. (1989) **Classifier Systems and Genetic Algorithms**, in ARTIFICIAL INTELLIGENCE. Vol 40, No 1-3.

Boole, G. (1854) **An Investigation of the Laws of Thought**, Walton & Maberly, London.

Born, R.P. (1987) **Split Semantics: Provocations Concerning Theories of Meaning and Philosophy in General** in Born, R. (Ed.) Artificial Intelligence: The Case Against, Croom Helm Ltd, Beckenham, Kent.

Brown, J.S., Burton, R.R., and de Kleer, J. (1982) **Pedagogical, Natural Language and Knowledge Engineering Techniques in SOPHIE I, II and III** in Sleeman, D., and Brown, J.S. (Eds.) Intelligent Tutoring Systems, Academic Press, New York.

Buchanan, B.G., Sutherland, G.L., and Feigenbaum, E.A. (1969) **Rediscovering Some Problems of Artificial Intelligence in the Context of Organic Chemistry** in Metzler, B., and Michie, D. (Eds.) Machine Intelligence 5, Edinburgh University Press, Edinburgh.

CIAM, (1928) **La Sarraz Declaration**, in DAS NEUE FRANKFURT. Translated by M. Bullock in 'Programmes and Manifestoes on 20th-Century Architecture' by U. Conrads published by Lund Humphries Ltd, London, 1970.

Carbonell, J.R. (1970) **AI in CAI: An Artificial Intelligence Approach to Computer-Assisted Instruction**, in IEEE TRANSACTIONS ON MAN-MACHINE SYSTEMS. Vol 11, No 4.

Charniak, E. and McDermott, D.V. (1985) **Introduction to Artificial Intelligence**, Addison-Wesley, New York.

Coleman, A. (1985) **Utopia on Trial**, Hilary Shipman, London.

Committee of Ministers, (1975) **European Charter of the Architectural Heritage**, Council of Europe, Strasbourg.

Cook, P. (1967) **Architecture; Action and Plan.**, Studio Vista Ltd, London.

Cooper, D. and Clancy, M. (1985) **Oh! Pascal!**, W.W. Norton & Company, New York.

Coyne, R.D., and Gero, J.S. (1986) **Semantics and the Organisation of Knowledge in Design**, in DESIGN COMPUTING. Vol 1, No 1.

Coyne, R.D., Rosenman, M.A., Radford, A.D. and Balachandran, M. (1990) **Knowledge-Based Design Systems**, Addison-Wesley Publishing Company, Reading, MA.

Cross, N., Naughton, J., and Walker, D. (1981) **Design Method and Scientific Method** in Jaques, R., and Powell, J.A. (Eds.) Design: Science: Method, Westbury House Ltd, Guildford.

Danto, A.C. (1980) **The Use and Mention of Terms and the Simulation of Linguistic Understanding** in THE BEHAVIORAL AND BRAIN SCIENCES. Vol 3, No 3.

Darby, J. (1988) **The IBM Humanities Project at Oxford University** in Proceedings of The Fifth International Conference on Technology and Education, Edinburgh.

Darke, J. (1979) **The Primary Generator in the Design Process**, in DESIGN STUDIES. Vol 1, No 1.

Davis, R.H., and Lyall, J. (1986) **Recognition of Handwritten Characters - A Review**, in IMAGE AND VISION COMPUTING. Vol 4, No 4.

Dennett, D.C. (1980) **The Milk of Human Intentionality**, in THE BEHAVIORAL AND BRAIN SCIENCES. Vol 3, No 3.

Dijkstra, E.W. (1959) **A Note on Two Problems in Connection with Graphs**, in NUMERISCHE MATEMATIK. Vol 1. 1959.

Dowsing, R.D., Rayward-Smith, V.J. and Walter, C.D. (1986) **A First Course in Formal Logic and it Applications in Computer Science**, Blackwell Scientific Pub'tions, Oxford.

Dreyfus, H.L. (1965) **Alchemy and Artificial Intelligence: Rand Memorandum P-3244.**, Rand Corporation, Santa Monica, CA.

Dreyfus, H.L. (1979) **What Computers Can't Do**, Harper Colophon Books, New York. Second Edition.

Dreyfus, H.L. and Dreyfus, S.E. (1986) **Mind Over Machine**, Blackwell, Oxford.

Duda, R.O., and Gaschnig, J.G. (1981) **Knowledge-Based Systems Come of Age**, in BYTE. Vol 6, No 2.

Eccles, J.C. (1980) **A Duallist-Interactionist Perspective**, in THE BEHAVIORAL AND BRAIN SCIENCES. Vol 3, No 3.

Eden, M. (1968) **Handwriting Recognition and Generation** in Kolers, P.A., and Eden, M. (Eds.) Recognising Patterns, MIT Press, Cambridge, MA.

Englemore, R.S., Morgan, A.J., and Nii, H.P. (1988) **Introduction** in Englemore, R., and Morgan, T. (Eds.) Blackboard Systems, Addison-Wesley Publishing Company, Wokingham.

Ernst, G.W. and Newell, A. (1969) **GPS: A Case Study in Generality and Random Problem Solving**, Academic Press, New York.

Feigenbaum, E.A. and McCorduck, P. (1983) **The Fifth Generation**, Addison Wesley Publishing Company, Reading, MA.

Finin, T., Joshi, A.K. and Webber, B.L. (1986) **Natural Language with Artificial Experts**, Univ. of Pennsylvania, Philadelphia, PA.

Fodor, J.A. (1980) **Searle on What Only Brains Can Do**, in THE BEHAVIORAL AND BRAIN SCIENCES. Vol 3, No 3.

Frege, G. (1884) **Dei Grunlagen der Arithmetik**, Koebner, Breslau. Translated by J.L.Austin as 'The Foundations of Arithmetic: A Logico-Mathematical Enquiry into the Concept of Number' published by Basil Blackwell, Oxford, 1953.

Frey, P.W. (1986) **A Bit-Mapped Classifier**, in BYTE. Vol 11, No 12.

Frey, P.W. (1986) **Listing of House.Bas**, in BYTE. Vol 11, No 12.

Fuller, R.B. (1960) **Prime Design**, in Bennington College Bulletin. Printed in 'The Buckminster Fuller Reader' edited by J. Meller and published by Jonathan Cape Ltd, London, 1970.

Garnier, T. (1925) **Une Cite Industrielle, etudes pour la construction des villes**, Vincent, Paris. Partially translated by D. Wiebenson as 'Tony Garnier: The Cite Industrielle' and published by George Baziller, New York, 1969.

Genesereth, M.R. and Nilsson, N.J. (1987) **Logical Foundations of Artificial Intelligence**, Morgan Kaufman Publishers Inc, Palo Alto, CA.

Ginzburg, M.I. (1924) **Stil'i epokha**, Gosuastvenoe Izatelestvo, Moscow. Translated by A. Senkevitch as 'Style and Epoch' and published by The MIT Press, Cambridge, MA, 1982.

Gove, H.E. (1987) **Mass Spectroscope** in Parker, S.P. (Ed.) McGraw-Hill Encyclopedia of Science and Technology, McGraw-Hill Book Company, New York.

Greenblatt, R.D., Eastlake, D.E., and Crocker, S.D. (1967) **The Greenblatt Chess Program**, in Proceedings of the Fall Joint Computer Conference.

Gropius, W. (1926) **Grundsatz der Bauhaus-Produktion**, in VIVOS VOCO. Vol V. Translated by W. Jabs and B. Gilbert as 'Bauhaus Dessau - Principles of Production' in 'The Bauhaus' by H.M. Wingler published by The MIT Press, Cambridge, MA, 1969.

Halliday, M.A.K. and Hasan, R. (1976) **Cohesion in English**, Lownes, London.

Haring, H. (1932) **Form der Leistungs-Erfulling**, in INNEN-DEKORATION. Vol 43. Translated by M. Bullock as 'The House as Organic Structure' in 'Programmes and Manifestoes on 20th-Century Architecture' by U. Conrads published by Lund Humphries Publishers Ltd, London, 1970.

Harris, L.R. (1977) **ROBOT: A High Performance Natural Language Data Base Query System**, in Proceedings of the Fifth International Joint Conference on Artificial Intelligence. Cambridge, MA.

Hastings, A. (1986) **Interactive Videodisc Project at University College Dublin**, in ART LIBRARIES JOURNAL. Vol 11, No 4.

Hayes, J.E. and Levy, D.N.L. (1976) **The World Computer Chess Championship**, Edinburgh University Press, Edinburgh.

Hayes-Roth, B. (1985) **Blackboard Architecture for Control**, in ARTIFICIAL INTELLIGENCE. Vol 26, No 3.

Hearst, E. (1967) **Psychology Across the Chess Board**, in PSYCHOLOGY TODAY. Vol 1, No 2.

Heidegger, M. (1927) **Sein und Zeit**, Max Neimeyer Verlag, Tubingen. Translated by J. Macquarrie and E. Robinson as 'Being and Time' and published by Basil Blackwell, Oxford, 1962.

Herrick, R. (1648) **Hesperides**, J. Williams & F. Eglesfield, London.

Hillier, W., Musgrove, J., and O'Sullivan, P. (1972) **Knowledge and Design** in Michell, W.J. (Ed.) Proceedings of Educational Design Research and Practice Conference 1972, Regents of the University of California, .

Hitchcock, H.R. and Johnson, P. (1932) **The International Style: Architecture Since 1922**, W.W. Norton & Co Inc, New York.

Hockey, S.M. and Scott, F.A. (1981) **The Kurzweil Data Entry Machine: Report to the Computer Board**, Oxford University Computing Service, Oxford.

Hofstadter, D.R. (1980) **Reductionism and Religion**, in THE BEHAVIORAL AND BRAIN SCIENCES. Vol 3, No 3.

Holland, J.H. (1986) **Escaping Brittleness: The Possibility of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems** in Michalski, R.S., Carbonell, J.G. and Mitchell, T.M. (Eds.) Machine Learning. An Artificial Intelligence Approach. Vol II, Morgan Kaufmann Publishers Inc, Los Altos, CA.

Hooper, R. (1977) **The National Development Programme in Computer-Assisted Learning: Final Report to the Director**, Council for Educational Technology, London.

Hundertwasser. (1958) **Schrift der Galerie Renate Boukes**, Reinhard Kaufman, Weisbaden. Translated by M. Bullock as 'Mould Manifesto Against Rationalism in Architecture' in 'Programmes and Manifestoes on Modern Architecture' by U. Conrads published by Lund Humphries Publishers Ltd, London, 1970.

Husserl, E.G.A. (1913) **Ideen zu einer reinen Phanomenologie und phanomenologischen Philosophie** in Neimeyer, (Ed.) Jahrbuch fur Philosophie und phanomenologische Forschung, Halle. Translated by W.R.B. Gibson as 'Ideas' and published by George Allen & Unwin, London, 1931

Huxtable, A.L. (1984) **The Tall Building Artistically Reconsidered**, Pantheon Books, New York.

Hyatt, R.M., Gower, A.E., and Nelson, H.L. (1986) **Cray Blitz** in Beal, D.F. (Ed.) Advances in Computer Chess 4, Pergamon Press, Oxford.

Illingworth, V., Glaser, E.L. and Pyle, I.C. (1986) **Dictionary of Computing**, Oxford University Press, Oxford.

Israel, D.J. (1987) **What's Wrong With Non-Monotonic Logic?** in Ginsberg, M.L. (Ed.) Readings in Non-Monotonic Logic, Morgan Kaufman, Los Altos, CA.

Jencks, C. (1983) **Post-Modernism: The True Inheritor of Modernism** in Murray, P. (Ed.) RIBA Transactions 3. Vol 2, No 1. RIBA Magazines, London.

Johnson, L., and Keravnou, E.T. (1988) **What Does Sophie Teach?**, in INTERNATIONAL JOURNAL OF SYSTEMS RESEARCH AND INFORMATION SCIENCE. Vol 2.

Jones, J.C. (1970) **Design Methods**, John Wiley & Sons Ltd, London.

Kahn, L.I. (1955) **Order Is**, in PERSPECTA, No 3.

de Kleer, J. (1987) **Qualitative Physics** in Shapiro, S.C. (Ed.) Encyclopedia of Artificial Intelligence, John Wiley & Sons, New York.

Krier, R. (1975) **Stadraum in Theorie und Praxis**, Karl Kramer Verlag, Stuttgart. Translated by C. Czeckowiski and G. Blaci as 'Urban Space' and published by Acadamy Editions, London, 1979.

Krier, R. (1981) **Vorwarts, Kameraden, Wir Mussen Zuruck.** Lecture at Oppositions 18 Forum. Translated by C. Hubert as 'Forward, Comrades, We Must Go Back' in OPPOSITIONS, No 24.

Krishnamurti, R. (1985) **Representing Design Knowledge**, EdCAAD, Edinburgh.

Lasdun, D. (1965) **An Architect's Approach to Architecture**, in RIBA JOURNAL. Vol 72, No 4.

Layzer, D. (1984) **Constructing the Universe**, Scientific American Books Inc, New York.

Le Corbusier. (1920) **Trois Rappels a MM les Architects**, in L'ESPRIT NOUVEAU. Translated by F. Etchells as 'Three Reminders to Architects' in 'Towards a New Architecture' published by The Architectural Press, London, 1927.

Le Corbusier. (1924) **Urbanism**, Editions Cres, Paris. Translated by F. Etchells as 'The City of Tomorrow' and published by The Architectural Press, London, 1929.

Le Corbusier. (undated) **Texte et Dessins pour Ronchamp** in Pauly, D. (Ed.) The Chapel at Ronchamps as an Example of le Corbusier's Creative Process, unpublished.

Lederberg, J. (1964) **A System for Computer Construction, Enumeration and Notation of Organic Molecules as Tree Structures and Cyclic Graphs. Parts 1-V**, in Interim Report to the National Aeronautics and Space Administration.

Libet, B. (1980) **Mental Phenomena and Behavior**, in THE BEHAVIORAL AND BRAIN SCIENCES. Vol 3, No 3.

Logan, B.S. (1987) **The Structure of Design Problems**, Unpublished PhD Thesis, Strathclyde University.

Loos, A. (1908) **Ornament und Verbrechen**, in DER STURM. Translated by H. Meek as 'Ornament and Crime' in 'Adolf Loos: Pioneer of Modern Architecture' by L. Munz and G. Kunstler published by Thames and Hudson, London, 1966.

Mackintosh, C.R. (1893) **Scotch Baronial Architecture**, unpublished typescript in the library of the Mackintosh School of Art, Glasgow.

Markus, T.A. (1969) **The Role of Building Performance Measurement and Appraisal in Design Method** in Broadbent, G., and Ward, A. (Eds.) Design Methods in Architecture, Lund Humphries Publishers Ltd, London.

Maver, T.W. (1970) **Appraisal in the Building Design Process** in Moore, G.T. (Ed.) Emerging Methods in Environmental Design and Planning, MIT Press, Cambridge, MA.

Maxwell, G. (1980) **Intentionality: Hardware, not Software**, in THE BEHAVIORAL AND BRAIN SCIENCES. Vol 3, No 3.

McCarthy, J. (1979) **Ascribing Mental Qualities to Machines** in Ringle, M. (Ed.) Philosophical Perspectives in Artificial Intelligence, Harvester Press, Brighton.

McCarthy, J. (1980) **Circumscription - A Form of Non-Monotonic Reasoning**, in ARTIFICIAL INTELLIGENCE. Vol 13, No 1 & 2.

McDermott, D., and Doyle, J. (1980) **Non-Monotonic Logic**, in ARTIFICIAL INTELLIGENCE. Vol 13, No 1 & 2.

Merleau-Ponty, M. (1945) **Phenomenologie de Perception**, Gallimard, Paris. Translated by C. Smith as 'Phenomenology of Perception' and published by Routledge & Kegan Paul, London, 1965.

Meyer, H. (1928) **Bauen**, in BAUHAUS. Vol 2, No 4. Translated by W. Jabs and B. Gilbert as 'Building' in 'The Bauhaus' by H.M. Wingler published by the MIT Press, Cambridge, MA, 1969.

Michie, D. (1968) **Memo Functions and Machine Learning**, in NATURE. Vol 218.

Michie, D. and Johnson, R. (1984) **The Creative Computer**, Viking Books, London.

Miller, G.A. & Chomsky, A.N. (1963) **Finitary Models of Language Users** in Luce, R.D., Bush, R. and Galanter, E. (Eds.) Handbook of Mathematical Psychology, John Wiley & Sons, New York.

Minsky, M.L. (1961) **Steps Toward Artificial Intelligence** in PROCEEDINGS OF THE INSTITUTE OF RADIO ENGINEERS. Vol 49.

Minsky, M.L. (1966) **Artificial Intelligence** in Flannagan, D. and Belto, E. (Eds.) Information, Scientific American Inc, New York.

Minsky, M.L. (1967) **Computation: Finite and Infinite Machines**, Prentice-Hall Inc, Englewood Cliffs, NJ.

Minsky, M.L. (1968) **Introduction** in Minsky, M.L. (Ed.) Semantic Information Processing, MIT Press, Cambridge, MA.

Minsky, M. (1975) **A Framework for Representing Knowledge** in Winston, P. (Ed.) The Psychology of Computer Vision, McGraw-Hill Book Company, New York.

Minsky, M. (1980) **Decentralised Minds**, in THE BEHAVIORAL AND BRAIN SCIENCES. Vol 3, No 3.

Moneo, R. (1978) **On Typology**, in OPPOSITIONS, No 13.

Moor, J.H. (1988) **The Pseudorealisation Fallacy and the Chinese Room Argument** in Fetzer, J.H. (Ed.) Aspects of Artificial Intelligence, Kluwer Academic Publishers, Dordrecht.

Moore, C. and Allen, G. (1976) **Dimensions**, Architectural Record Books, New York.

Morris, W. (1893) **Gothic Architecture**, Kelmscott Press, Hammersmith.

Mounce, H.O. (1981) **Wittgenstein's Tractatus**, Basil Blackwood, Oxford.

Muthesius, H. & Van de Velde H. (1914) **Werkbund Theses and Antitheses**, in BAUWELT, No 27. Translated by M. Bullock in 'Programmes and Manifestoes on 20th-Century Architecture' by U. Conrads published by Lund Humphries Publishers Ltd, London, 1970.

Natsoulas, T. (1980) **The Primary Sources of Intentionality**, in THE BEHAVIORAL AND BRAIN SCIENCES. Vol 3, No 3.

Neutra, R. (1954) **Survival Through Design**, Oxford University Press, Oxford.

Newell, A. (1966a) **On the Analysis of Human Problem Solving Protocols**, in Proceedings of the International Symposium on Mathematical and Computational Methods in the Social Sciences, Rome.

Newell, A. (1966b) **On the Representation of Problems**, Carnegie Institute of Technology, Pittsburg.

Newell, A. (1973) **Artificial Intelligence and the Concept of Mind** in Schank, R.C., and Colby, K.M. (Eds.) Computer Models of Thought and Language, W.H. Freeman & Company, San Francisco.

Newell, A. and Simon, H.A. (1972) **Human Problem Solving**, Prentice-Hall Inc, Englewood Cliffs, NJ.

Newman, O. (1961) **A Short Review of CIAM Activity** in Joedicke, J. (Ed.) Documents of Modern Architecture, Alec Tiranti Ltd, London.

Nilsson, N.J. (1980) **Principles of Artificial Intelligence**, Springer-Verlag, Berlin.

NLP (1984) **Tutorial Number 3 on Natural Language Processing**, in Proceedings of the AAAI National Conference on Artificial Intelligence, Austin, TX.

Norman, R.B. (1987) **Intuitive Design and Computation** in Kalay, Y.E. (Ed.) Computability of Design, John Wiley & Sons, New York.

O'Brien, F. (1967) **The Third Policeman**, McGibbon & Kee Ltd, London.

Obermeier, K.K. (1983) **Wittgenstein on Language and Artificial Intelligence: The Chinese-Room Thought Experiment Revisited**, in SYNTHESE. Vol 56.

Oettinger, A.G. (1955) **The Design of an Automatic Russian-English Technical Dictionary** in Locke, W.N., and Booth, A.D. (Eds.) Machine Translation of Languages, MIT Press, Cambridge MA.

Page, J.K. (1963) **A Review of the Papers Presented at the Conference** in Jones, J.C., and Thornley, D.G. (Eds.) Conference on Design Methods, Pergamon Press, Oxford.

Page, J.K. (1964) **Environmental Research Using Models**, in ARCHITECTS JOURNAL. Vol 139, No 11.

Passmore, J. (1957) **A Hundred Years of Philosophy**, Gerald Duckworth & Company, London.

Peirce, C.S. (1901) **Application of the Method** in Burks, A. (Ed.) Collected Papers of C.S. Peirce, Harvard University Press, Cambridge, MA, 1957.

Peirce, C.S. (1903) **The Three Cotary Propositions** in Burks, A. (Ed.) Collected Papers of C.S. Peirce, Harvard University Press, Cambridge, MA, 1957.

Piaget, J. (1936) **La Naissance de l'Intelligence Chez l'Enfant**, Delachaux & Neistle, Paris. Translated by M. Cook as 'The Origin of Intelligence in the Child' and published by Routledge & Kegan Paul, London, 1953.

Popper, K.R. (1934) **Logic der Forschung** in Frank, P., and Schlick, M. (Eds.) Schriften zur wissenschaftlichen Weltauffassunig, Verlag von Julius Springer, Vienna. Translated by K.R. Popper, J. Freed and L. Freed as 'The Logic of Scientific Discovery' and published by Hutchinson and Co Ltd, London, 1959.

Popper, K.R. (1972) **Conjectures and Refutations**, Routledge and Kegan Paul, London.

Popper, K. (1974) **Replies to My Critics: The Problem of Demarcation** in Schlipp, P.A. (Ed.) The Philosophy of Karl Popper, The Open Court Publishing Co, La Salle, IL.

Post, E.L. (1943) **Formal Reductions of the General Combinatorial Decision Problem**, in AMERICAN JOURNAL OF MATHEMATICS. Vol 65.

Preziosi, D. (1979) **The Semiotics of the Built Environment**, Indiana University Press, Bloomington.

Puccetti, R. (1980) **The Chess Room: Further Demythologising of Strong AI**, in THE BEHAVIORAL AND BRAIN SCIENCES. Vol 3, No 3.

Pylyshyn, Z.W. (1974) **Mind, Machines and Phenomenology. Some Reflections on Dreyfus' 'What Computers Can't Do'**, in COGNITION. Vol 3, No 1.

Quillian, M.R. (1968) **Semantic Memory** in Minsky, M. (Ed.) Semantic Information Processing, The MIT Press, Cambridge, MA.

Rankin, T.L. (1988) **When is Reasoning Nonmontonic?** in Fetzer, J.H. (Ed.) Aspects of Artificial Intelligence, Kluwer Academic Publishers, Dordrecht.

Reiter, R. (1980) **A Logic for Default Reasoning**, in ARTIFICIAL INTELLIGENCE. Vol 13, No .

Rich, E. (1983) **Artificial Intelligence**, McGraw-Hill Inc., Singapore.

Ringle, M. (1980) **Mysticism as a Philosophy of Artificial Intelligence**, in THE BEHAVIORAL AND BRAIN SCIENCES. Vol 3, No 3.

Rittel, H.W.J. (1972) **Son of Rittelthink**, in Design Methods Group Occasional Paper No 1.

van der Rohe, M. (1960) **Letter to Douglass V. Freret, 13 February 1960**. Printed in 'Mies van der Rohe: Architect as Educator' edited by R. Achilles, K. Harrington and C. Myhrum and published by the University of Chicago Press, Chicago, IL, 1986.

Rosch, E. (1977) **Human Categorisation** in Warren, N. (Ed.) Studies in Cross-cultural Psychology, Academic Press, London.

Rossi, A. (1981) **A Scientific Autobiography**, MIT Press, Cambridge, MA.

Russell, B. and Whitehead, A.N. (1913) **Principia Mathematica**, Cambridge University Press, Cambridge.

Russell, B. (1918) **The Philosophy of Logical Atomism**, in THE MONIST. Vol 28 & 29.

Russell, B. (1944) **My Mental Development** in Schilpp, P.A. (Ed.) The Philosophy of Bertrand Russell, Harper & Row, New York.

Russell, P.J. (1986) **Genetics**, Little, Brown & Co, Boston.

Ryle, G. (1949) **The Concept of Mind**, Hutchinson, London.

Sandars, N.K. (1968) **Prehistoric Art in Western Europe**, Penguin Books Ltd, Harmondsworth.

Sant'Elia, A. (1914) **L'Architettura Futurista: Manifesto**, in LACERBA. Vol 2, No 15. Translated by R.W. Flint as 'Manifesto of Futurist Architecture' in 'Futurist Manifestoes' edited by U. Apollonio published by Thames and Hudson, London, 1973.

Schank, R.C. (1973) **Identification of Conceptualizations Underlying Natural Language** in Schank, R.C. and Colby, K.M. (Eds.) Computer Models of Language and Thought, W.H. Freeman & Company, San Francisco.

Schank, R.C. (1975) **Conceptual Information Processing**, North Holland Publishing Co, Amsterdam.

Schank, R.C. (1979) **Natural Language, Philosophy and Artificial Intelligence** in Ringle, M. (Ed.) Philosophical Perspectives in Artificial Intelligence, Harvester Press, Brighton.

Schank, R. and Abelson, R. (1977) **Scripts, Plans, Goals and Understanding**, Lawrence Erlbaum Assoc., Hillsdale, NJ.

Scheerbart, P. (1914) **Glasarchitektur**, Verlag Der Sturm, Berlin. Translated by M. Bullock as 'Glass Architecture' in 'Programs and Manifestoes on Modern Architecture' by U. Conrads published by Lund Humphries Publishers Ltd, London, 1970.

Schlick, M. (1928) **Preface to 'Logic, Sprache, Philosophie. Kritik der Philosophie.' by Freidrich Waismann.** in Mulder, H.L., and Van de Velde-Schlick, B.F.B. (Eds.) Moritz Schlick, Philosophical Papers. Vol II (1925-1936), D. Reidel Publishing Company, Dordrecht, 1979.

Schrodinger, E. (1935) **Die gegenwartige Situation in der Quantenmechanik**, in DIE NATURWISSENSCHAFTEN. Vol 23. Translated by J.D. Trimmer as 'The Present Situation in Quantum Mechanics' in Proceedings of the American Philosophical Society, Vol 124, 1980.

Scott, G. (1914) **The Architecture of Humanism**, Constable & Co Ltd, London.

Searle, J.R. (1979) **What is an Intentional State?**, in MIND. Vol 88.

Searle, J.R. (1980a) **Minds, Brains and Programs**, in THE BEHAVIORAL AND BRAIN SCIENCES. Vol 3, No 3.

Searle, J. (1980b) **Author's Response**, in THE BRAIN AND BEHAVIORAL SCIENCES. Vol 3, No 3.

Searle, J.R. (1984) **Minds, Brains and Science: The Reith Lectures**, in THE LISTENER. Vol 112, No 2883, 2884, 2885, 2886, 2887, 2888.

Sell, P.S. (1985) **Expert Systems - A Practical Introduction**, Macmillan Publishers Ltd, London.

Shankland, G., Willmott, P. and Jordan, D. (1977) **Inner London: Policies for Dispersal and Balance**, H.M.S.O., London.

Shannon, C.E. (1950) **Programming a Digital Computer for Playing Chess**, in PHILOSOPHICAL MAGAZINE. Vol 41, No 314.

Shapiro, S.C., Eckroth, D. and Vallasi, G.A. (1987) **Encyclopedia of Artificial Intelligence**, John Wiley & Sons, New York.

Sheffer, H.M. (1913) **A Set of Five Independent Postulates for Boolean Algebras, with Application to Logical Constants**, in TRANSACTIONS OF THE AMERICAN MATHEMATICAL SOCIETY. Vol 14.

Shortliffe, E.H. and Fagan, L.M. (1972) **Expert Systems Research: Modelling of Medical Decision Making Processes**, Stanford University, Stanford, CA.

Simon, H. (1977) **Artificial Intelligence Systems that Understand**, in 5th International Joint Conference on Artificial Intelligence. Vol II.

Smith, B.C. (1986) **Varieties of Self-Reference** in Halpern, J.Y. (Ed.) Theoretical Aspects of Reasoning About Knowledge. Proceedings of the 1986 Conference, Morgan-Kaufman, Los Altos, CA.

Soleri, P. (1971) **The Sketchbooks of Paolo Soleri**, MIT Press, Cambridge, MA.

Sommerville, I (1985) **Software Engineering**, Addison-Wesley Publishing Co, Wokingham.

Stiny, G.N. (1985) **Computing with Form and Meaning in Architecture**, in JOURNAL OF ARCHITECTURAL EDUCATION.

Sullivan, L. (1924) **Autobiography of an Idea**, AIA Press, New York.

Szanser, A.J. (1967) **Machine Translation - The Evaluation of an Experiment**, THE INCORPORATED LINGUIST. Vol 6, No 4.

Terry, Q. (1983) **Genuine Classicism** in Murray, P. (Ed.) RIBA Transactions 3. Vol 2, No 1. RIBA Magazines, London.

Thines, G. (1987) **Husserl** in Gregory, R.L. (Ed.) The Oxford Companion to the Mind, Oxford University Press, Oxford.

Vaux, J. (1988) **Minds and Machines**, in NEW SCIENTIST. Vol 117, No 1600.

Venturi, R. (1966) **Complexity and Contradiction in Architecture**, Museum of Modern Art, New York.

Viollet-le-Duc, E.E. (1863) **Sur l'Architecture au Dix-Neuvieme Siecle - Sur la Methode** in Viollet-le-Duc, E.E. (Ed.) Entretiens sur l'Architecture, A. Morel & Cie, Paris. Translated by B. Backnell as 'Architecture in the Nineteenth Century - Importance of Method' in 'Discources on Architecture', Vol I, published by Ticknor & Co, Boston, MA, 1889. Republished by Allen & Unwin, London, 1959.

Voysey, C.F.A. (1915) **Individuality**, Chapman & Hall Ltd, London.

Wachsmann, K. (1961) **The Turning Point of Architecture**, Reinhold Publishing Corp, New York.

Waismann, F. (1967) **Ludwig Wittgenstein und der Weiner Kreis**, Basil Blackwell, Oxford. Translated by J. Schultz and B. McGuiness as 'Wittgenstein and the Vienna Circle' and published by Basil Blackwell, Oxford, 1979.

Waltz, D. (1975) **Understanding Line Drawings of Scenes with Shadows** in Winston, P.H. (Ed.) The Psychology of Computer Vision, McGraw-Hill Book Company, New York.

Waterman, D.A. (1970) **Generalized Learning Techniques for Automating the Learning of Heuristics**, in ARTIFICIAL INTELLIGENCE. Vol 1, NO 1.

Wheatley, J. (1990) **Commercial Propositions**, in BUILDING DESIGN. No 983.

Whitehead, A.N. (1933) **Adventures of Ideas**, Cambridge University Press, Cambridge.

Winograd, T. (1975) **Frame Representations and the Procedural/Declarative Controversy** in Bobrow, D.G., and Collins, A. (Eds.) Representation and Understanding, Studies in Cognitive Science, Academic Press, New York.

Winograd, T. (1980) **Extended Inference Modes of Reasoning by Computer Systems**, in ARTIFICIAL INTELLIGENCE. Vol 13, No 1 & 2.

Winston, P.H. (1980) **Learning and Reasoning by Analogy**, in Communications of the Association for Computing Machinery. Vol 23, No 12.

Winston, P.H. and Horn, B.K.P. (1981) **LISP**, Addison-Wesley Publishing Co, Reading, MA.

Winston, P.H. (1984) **Artificial Intelligence**, Addison-Wesley Publishing Co., Reading, MA.

Wittgenstein, L. (1922) **Tractatus Logico-Philosophicus**, Routledge & Kegan Paul Ltd, London. Translated by D. Pears and B. McGuiness and published by Routledge & Kegan Paul, London, 1961.

Wittgenstein, L. (1953) **Philosophical Investigations**, Basil Blackwell, Oxford. Edited and translated by G.E.M. Anscombe.

Wittgenstein, L. (1961) **Notebooks**, Basil Blackwell, Oxford. Edited by G.H. von Wright and G.E.M. Anscombe and translated by G.E.M. Anscombe.

Wright, F.L. (1931) **To the Young Man in Architecture**, in Chicago Art Institute Lectures, Chicago Art Institute, Chicago, IL.

Wright, F.L. (1935) **Broadacre City: A New Community Plan**, in ARCHITECTURAL RECORD. Vol 104.

Zucker, S.W. (1987) **Early Vision** in Shapiro, S.C., Eckroth, D. and Vallasi, G.A. (Eds.) Encyclopedia of Artificial Intelligence, John Wiley & Sons, New York.