CRITERIA FOR CHOOSING REPRESENTATION LANGUAGES AND CONTROL REGIMES FOR EXPERT SYSTEMS.

Han Reichgelt
Frank van Harmelen

Department of Artificial Intelligence
University of Edinburgh
Scotland

ABSTRACT

Shells and high-level programming language environments suffer from a number of shortcomings as knowledge engineering tools. We conclude that a variety of knowledge representation formalisms and a variety of controls regimes are needed. In addition guidelines should be provided about when to choose which knowledge representation formalism and which control regime. The guidelines should be based on properties of the task and the domain of the expert system. In order to arrive at these guidelines we first critically review some of the classifications of expert systems in the literature. We then give our own list of criteria. We test this list applying our criteria to a number of existing expert systems. As a caveat, we have not yet made a systematic attempt at correlating the criteria and diferent knowledge representations formalisms and control regimes, although we make some preliminary remarks throughout the paper.

INTRODUCTION

Two types of tools are currently available for constructing expert systems, namely expert system shells and high level programming language environments. Both of these types of tool suffer from a number of shortcomings which limit their usefulness.

The first type of available tools, shells, are usually constructed by abstraction from a concrete expert system. One takes a concrete expert system and takes out the contents of the knowledge base which is specific to the problem at hand to derive a shell. Thus, a shell normally consists of an inference engine and an empty knowledge base and some rather primitive debugging and explanation facilities. Buyers of shells often believe, and manufacturers often claim, that the shell is appropriate for a number of different tasks in different domains. However, a large number of people have expressed dissatisfaction with expert system shells. The two most frequently heard complaints about shells (e.g. [1]) are first that the view that the inference engine which was successful in one application will also be successful in other applications is unwarranted, and second that the knowledge representation scheme often makes the expression of knowledge in another domain awkward if not impossible.

On the other hand, proponents of high level language programming environments, such as LOOPS [27], KEE [28], or ART [30], can be seen as taking a more pluralistic position:- instead of providing knowledge engineers with a single pre-fabricated inference engine one provides knowledge engineers with a large set of tools each of which has proved useful in other applications. LOOPS, for example, provides object-oriented programming with inheritance, a production-rule interpreter, active values as well as LISP.

While we accept that systems such as KEE are useful as tools for program development we claim that they are less useful as tools for building expert systems for non-AI programmers. Their main problem is that they provide the knowledge engineer with a bewildering array of possibilities and little, if any, guidance under what circumstances which of these possibilities should be used. Unless used by experienced programmers, high level programming environments encourage an ad hoc programming style in which no attention is paid to a principled analysis of the problem at hand to see which strategy is best suited for its solution.

The conclusion that we draw from the problems associated with shells and high level programming language environments is that there are a number of different 'models of rationality' and that different tasks and different domains require different models of rationality. When constructing expert systems, knowledge engineers then have to decide which model of rationality is appropriate to the domain and the task at hand. It is our belief that it is possible to give some guidelines which should make this decision easier for the knowledge engineer.

The notion of a model of rationality, although perhaps intuitively appealing, is rather vague and needs to be made more precise. We see a model of rationality as having both a static and a dynamic aspect. It consists of domain knowledge plus knowledge about how to use this knowledge when solving a problem. We will also use the term 'problem solving strategy' for the dynamic aspect of a model of rationality.

It is important to realise that we define the notion of a model of rationality at the epistemological level in the sense of Breuker and Wielinga [5]. Generalising work by Brachman [4], they distinguish between five levels at which one can discuss the representation and manipulation of knowledge in expert systems: the linguistic, the conceptual, the epistemological, the logical and the implementational level. The linguistic level corresponds to simply representing what an expert reports on his knowledge. At the conceptual level, the question is addressed which primitives are needed to represent the knowledge formally. The analysis at the epistemological level uncover structural properties of the knowledge expressed at the conceptual level, using types of concept and types of inference strategy. The logical level of analysis applies to the (logical) formalisms in which implementational mechanisms are uncovered on which higher levels are based. The notion of a model of rationality is an epistemological one because it is intended to show structural properties of the knowledge, both in terms of the actual knowledge about some domain and in terms of how this knowledge is to be used in reasoning. Because of this, the epistemological level also seems to be the appropriate level for formulating criteria for choosing a model of rationality.

The distinction between the static and the active aspects of a model rationality corresponds to a distinction which one can make between two different aspects of an expert system. First, there is the domain which the knowledge to be embedded in the expert system is about. Thus, the domain of an expert system may be electronics or internal medicine. Secondly, there is the task which the knowledge engineer wants the expert system to perform. Thus, the task of a system can be diagnosing a faulty electronic circuit or monitoring a circuit. It is interesting to note that the problems with shells reflect these two aspects of an expert system. The first complaint about shells concerning the expressiveness of the knowledge representation language is related to the structure of the domain. The second complaint about shells concerning the rigidity of the inference engine is related to the task of the expert system. As pointed out by Chandrasekaran [10, 11], typical expert system tasks such as diagnosis, planning, monitoring etc. seem to be related to particular control regimes.

The correspondence between the two aspects of a model of rationality and the problems with shells suggests that a model of rationality should be computationally realised as a knowledge representation formalism and a control regime for using this knowledge representation formalism. Therefore, if we are to provide knowledge engineers with the tools for building an expert system, we have to give them advice on which knowledge representation formalism and which control regime to choose. In this paper, we will formulate two sets of criteria which are relevant in making this choice. We will call the criteria which are relevant in choosing a knowledge representation formalism 'domain criteria', while the criteria for control regimes will be called 'task criteria'.

Before we give the list of criteria, three more introductory remarks have to be made. First, the criteria we give are intended as guidelines rather than as prescriptions. Thus, whenever we say that a particular knowledge representation formalism is most suited for an application with a certain property, then this should not be taken as a categorical statement. It is only intended as a piece of advice and knowledge engineers may want to reject it and opt for another knowledge representation formalism.

A second remark is that we assume that the different criteria which we give here are independent of each other. We realise that this is an idealisation which may have to be given up later. For the time being, however, we will not make any attempt to relate the different criteria. A further idealisation lies in the fact that we will present our criteria as being discrete, while in practice most of our criteria will represent continua in which many intermediate values exist in between the extremes.

The third introductory remark is a terminological point. We distnguish between two different types of information used in an expert system. First, there is the permanent knowledge encoded in the knowledge base. An example is a rule like "If someone has a headache and a dry mouth, and they drank excessively the previous night, then they have a hangover and no essential medication is called for". We will use the term 'Knowledge' for this type of information. A second type of information is session dependent information for which we will use the term 'data'. We distinguish between two types of data, namely external and inferred data. Internal data is information which the system receives from outside sources, whether they be users of the system, or sensory equipment the system is hooked up to. Inferred data is information which the system infers itself from other data. Thus, if the user tells the system that Peter has a headache and a dry mouth and that Peter drank excessively the previous night, then we are dealing with external data. If the system then uses this data together with the rule mentioned before and derives that Peter has a hangover, then this is inferred data. Note that there is a continuum between knowledge and data. Data can become part of a permanent knowledge base. Thus, if we store the fact that Peter had a hangover on June 12th in a patient record, then what was data originally becomes knowledge. Recording data can of course be very useful for gathering statistical information about for example the incidence of diseases during a particular period, or about the effectiveness of a drug.

RELATED WORK

The idea of using different knowledge representation formalisms and different control regimes for different expert system aplications is used by various other researchers in the field of expert systems as well. In this section, we will briefly mention some of them.

In the introduction we already mentioned Chandrasekaran and his group. Their work is based on the same intuition underlying our work, namely that there are task specific models of rationality and that the control regime and the knowledge representation which are most appropriate for a particular application, depend on the type of task at hand. (Cf [9, 10, 11]). The underlying claim is that complex knowledge-based reasoning tasks can often be decomposed into a number of generic tasks each with associated types of knowledge and family of control regimes. However, as far as we know, although they have come up with some examples of these generic tasks, they have not (yet) made a complete catalogue of generic tasks.

Another piece of related work is the classification in Stefik et al. [18]. They claim that one of the most variable characteristics of expert systems is the way they search for solutions, and they then propose a classification of expert systems which is based on the search strategy adopted in the system. In the next section we will discuss their classification in more detail.

Other related work is work done more specifically in the area of knowledge acquisition. Bennett's ROGET [3] is a program which assists in the contruction of expert systems by helping identify the conceptual structure of the system, a representation of the kinds of domain-specific inferences that the system will need to perform and the facts that support these inferences. It finds the appropriate conceptual structure by using a classification of problem-solving tasks, each entry of which is associated with a conceptual structure. ROGET tries to identify the problem at hand with an entry in this classification and then uses the associated conceptual structure to guide the acquisition dialogue with a domain expert. KADS, a system currently under development at the University of Amsterdam is based on the same intuition. ([5, 6]).

CLASSIFICATIONS IN THE LITERATURE

There are various classifications of types of expert system and types of expert system task in the literature. Rather than discuss all the different classifications in great detail, we will only discuss what are probably the two best known, both of which can be found in [16].

The Hayes-Roth, Waterman, Lenat classification

The introduction to Building Expert Systems [16, pp. 13-16] contains a classification of types of expert systems. Hayes-Roth et al. distinguish between ten different types of expert systems. They are:

— interpretation systems
— prediction systems
— diagnosis systems
— design systems
— planning systems
— monitoring systems
— debugging systems
— repair systems
— instruction systems
— control systems

Interpretation systems infer situation descriptions from observables. They are used to explain observed data by assigning to them symbolic meanings describing the situation accounting for the data. Examples are surveillance systems and speech understanding systems. Prediction systems infer likely consequences from given situations. The obvious example is weather forecasting. Diagnosis systems infer system malfunctions from observed data. This category includes medical diagnosis and electronic fault finding. Design systems develop configurations of objects that satisfy the constraints on the design problem. Examples are circuit layout and building design. Planning systems design actions. They specialise in design for objects that perform functions. Monitoring systems compare observations of system behaviour to features that are crucial to successful plan outcomes. These features correspond to potential flaws in the plan. Debugging systems prescribe remedies for malfunctions. Repair systems develop and execute plans to administer a remedy for some diagnosed problem. Instruction systems diagnose and debug student behaviour. Finally, control systems adaptively govern the overall behaviour of a system.

The above classification of expert systems is unsatisfactory in two respects. Firstly, a number of the expert systems which are included in the classification are built up from more primitive expert systems which are

included in the classification are built up from more primitive expert systems. No reason is given why these systems are included whereas other possible combinations of 'primitive' expert systems are not included. Second, at least some of the types of expert systems in it are special cases of more general types of expert systems included in the hierarchy. In general, the more specific cases are used to perform the same general task as the more general expert system but in a more specific domain.

The first complaint we had against the above classification was that some of the types of expert system in the list were not primitive in the sense that they are composed of a number of simpler expert systems. In the definition of debugging systems, it is not quite clear whether the system itself diagnoses the problem for which it prescribes a remedy. If it does, then we are dealing with a complex system which first does some diagnosis to infer some system malfunction, and subsequently constructs a plan to remedy the diagnosed malfunction. Repair systems certainly do both these tasks and are therefore always complex systems. Unlike debugging systems, also they must have execution capabilities, as they are used to execute the plan that they constructed. Instruction systems diagnose and debug student behaviours, i.e. the malfunctions in the student behaviour are inferred and then remedied. Control systems, finally, adaptively govern the behaviour of a system. They therefore combine most of the above primitive expert systems tasks. From the ten original expert systems, only seven can be seen as potentially primitive, namely interpretation, prediction, diagnosis, design, planning, monitoring and possibly debugging systems.

We are principally interested in 'primitive' tasks, i.e. expert systems tasks which cannot be treated as consisting of other, smaller, expert system tasks. It is at least initially plausible to concentrate on the primitive systems, and hope that the more complex expert systems can be constructed by combining the primitive ones.

The second complaint concerned the fact that at least some of the types of expert system are specialisations of more general types of system included in the classificaton. Consider for example interpretation and diagnosis systems. Interpretation systems infer situation descriptions from observables. Diagnosis systems perform exactly the same task, except that the situation description inferred in a diagnosis system is a system malfunction, whereas there is no such restriction on the type of situation description in the case of an interpretation system. It is therefore reasonable to see diagnosis systems as a sub-class of interpretation systems. Similarly, planning systems can be regarded as special cases of design systems. Design systems are used to develop configurations of objects that satisfy certain constraints. The configurations that are constructed in planning systems are diagnoses themselves are a special case of planning systems:- they design plans for remedying malfunctions. We thus get a hierarchy of design systems, planning systems and debugging systems.

If we restrict the classification proposed by Hayes-Roth et al to the types of expert system which are primitive and general, we are left with only four of the original ten types of expert system. We thus have the following four tasks left:

— interpretation tasks
— prediction tasks
— design tasks
— monitoring tasks

In the remainder of this paper, we will refer to this list of expert systems as the 'revised Hayes-Roth classification'.

The Stefik classification of search strategies

Stefik, Aikins, Balzer, Benoit, Birnbaum, Hayes-Roth and Sacerdoti [18] give a prescriptive guide to building expert systems. They propose a classification of expert systems. On the basis of this classification, they make prescriptions for appropriate search techniques.

The preoccupation with search techniques in the Stefik classification is by no means unusual. After all, Stefik et al. rightly claim that the way in which expert systems search their solution space is one of their most important and variable aspects. However, for our purposes, it is too limited. While the search strategy is an important part of the control regime, the question of knowledge representation is hardly touched.

Stefik et al. try to determine what properties of the domain and the task are relevant in determining which search technique would be appropriate by starting with a simple application for which a simple search technique can be used, and gradually relax the restrictions on the application, thus leading to more complex search techniques. However, when it comes to drawing fine distinctions, they soon give up the attempt to correlate features of the domain and task to search techniques, and seem to base their distinctions more on implementational issues. We will return to this later.

Initially then, Stefik et al. consider a simple application for which simple exhaustive search would be appropriate. The restrictions that an application has to meet for exhaustive search to be appropriate are a small solution space, reliable and fixed data, and reliable knowledge. The important criteria thus are the size of the solution space, the temporal nature of the constraints on the solution, the reliability of data and knowledge. We will discuss them in the reverse order.

The first restriction Stefik et al. relax is the reliability of knowledge and data. Although they initially seem to make a distinction between data and knowledge that is similar to ours, they do not persist with this but rather treat the slightly different problems of unreliable data or knowledge in the same way. The techniques that one can use in applications with uncertain knowledge or data are the use of probability models, the use of fuzzy logic, or belief revision.

A second constraint that Stefik et al. use is whether the data used in a session vary with time. If they do, then one appropriate technique would be the use of state-triggered expectations and dynamic belief-revision. Stefik et al. conjecture that systems for dealing with time-varying data need more elaborate representations of events and time, and that while such representations are within reach, their construction is still a research enterprise.

The final constraint that Stefik et al. relax is the size of the solution space. The most appropriate search technique for large but factorable solution spaces is hierarchical generate-and-test, but there are still some problems that remain even in this case. Examples are cases where there is no evaluator for partial solutions, cases where a single line of reasoning or single knowledge source is too weak, and cases where the representation method is too inefficient. We would argue however that at least some of these problems are based on implementational issues and not on the sort of epistemological considerations relevant for our purpose.

The Stefik classification of expert systems and the prescriptions which are made for choosing a search technique most suited to the different types of expert system are an example of the type of guidelines that we have in mind as well. Our main criticism against it is that the guidelines are only for choosing search techniques and that the problem of choosing an appropriate knowledge representation formalism is not addressed. Also, some of the criteria which are used are at the implementational rather than the epistemological level

Knowledge representation formalisms

In the introduction of this paper, we distinguished between two aspects of any expert system, namely the domain the system has knowledge about, and the task which the system is to perform with this knowledge. We argued that the type of domain the system is dealing with would be relevant for choosing an appropriate knowledge representation formalism, whereas the type of task would be relevant for choosing an appropriate control regime for the formalism. We will first deal with the domain-related criteria, which are relevant for choosing a knowledge representation formalism. But before we do so, we will briefly digress about the type of formalism which we intend to use.

There are three main types of knowledge representation language which have been used in expert systems:-production rules, structured objects (i.e. semantic networks and frame systems), and logic (see [21], chapters 3-5 for a detailed discussion of each of these). We will use logic as our knowledge representation language, at least at the epistemological level at which the criteria are formulated.

This is not the place to discuss logic or extol its virtues as a knowledge representation language in any great detail. However, as the word "logic" is used in a confusing way, we will define precisely what we mean by a logic. We distinguish between three aspects of a logic. First, there is the grammar of the logic in which one defines what expressions are part of the logical language. Thus, usually one defines a set of basic expressions and some recursive rules which determine how one can construct complex expressions out of basic or other complex expressions. Second, there is the semantics of the logic, in which one defines what the meanings of the expressions are. Usually, one defines the meanings of the basic expressions first. Then, corresponding to each grammatical rule, one defines a semantic rule which determines the meaning of the complex expression from the meanings of the simpler expressions out of which it has been constructed. Third, there is the proof-theory of the logic. In it, one defines several formal rules which can be used to derive new sentences from given sentences. In general, one tries to construct a proof theory which is sound and complete. A proof theory is sound if, whenever a sentence A can be derived from a set of sentences S, A is also a valid consequence of S, that is if all the sentences in S are true, then A is true as well. A proof theory is complete if, whenever A is a valid consequence of S, A can also be derived from S. In general, only soundness is considered to be essential for a proof theory to be acceptable, although a complete proof theory is of course preferred over an incomplete one.

Given this definition of logics, we can now give two reasons for preferring logics over other knowledge representation formalisms. First, logics have a semantics i.e. for every well-formed expression in the formalism the conditions under which it is true are precisely specified. Apart from the intrinsic value of a truth-conditional interpretation, one can also use it to guarantee that the inferences which are drawn are correct in the sense that from true premises only true conclusions can be drawn. Thus, by using logic one can guarantee the correctness of the results, assuming the correctness of the knowledge and the data from which the system is reasoning.

A second advantage of using logic is the high flexibility which it gives. There are many well-understood logics, each of which can be used to model special kinds of reasoning. Starting with classical predicate calculus, the formalism which is discussed in most elementary text books (see for example [44], or [49]), we will follow [42] and distinguish between two types of alternative logic. The first type of alternative logic that Haack mentions are deviant logics. Deviant logics have the same grammar as first-order classical predicate calculus but a different semantics. An example of a deviant logic would be intuitionistic predicate calculus (see [49], chapter 5, and [47], chapter 4.) Another example of a deviant logic which has been used quite often in the context of expert systems is fuzzy logic [41, 37].

The second type of alternative logic that Haack distinguishes are extended logics. Extended logics are logics whose expressions properly include the expressions of classical predicate calculus, and whose inferences properly include the set of inferences allowed in classical first-order predicate calculus. Extended logics thus take the classical predicate calculus as their bases and add new logical operators to the language and other inferences to the set of already allowed inferences. There are many examples of extended logics. We will mention only two types here. First, there are modal logics in which the language of classical predicate calculus is extended with a necessity and a possibility operator. (The standard book on modal logics is [43]). Thus, one can say that a particular proposition is necessary or possible. What makes modal logic an extended rather than a deviant logic is the fact that all the inferences which are allowed in classical predicate logic are also allowed in modal logic. Because there are many different ways of specifying the meaning of the necessity and the possibliity operator, there are many different kinds of modal logic. A second type of extended logic is temporal or tense logic. In temporal logic one adds operators which have the intended meaning that the formula within their scope are true on a particular point in time. For a very technical descussion of temporal logics, see [46].

Note that the definition of a logic as consisting of a logical language and an interpretation for that language means that the so-called non-monotonic logics do not count as logics, at least not until they have been given an adequate semantics. For a partial attempt at doing this, see [47, chapter 5] and references therein.

We would like to repeat that the analyses which we propose are at the epistemological level. That also applies to the above discussion of logic. Although we believe that the advantages which we mention are correct, this should not be taken to imply that the different logics which we think are useful should be implemented directly. It is certainly possible to implement theories of modal logics in first-order predicate caluclus, as Moore [45] proposes. Also, structured object representations may be a good technique to use in the implementation of logic-based expert systems (Cf [17]). However, we will not pursue this question here.

DOMAIN-RELATED CRITERIA

Now we turn to the domain criteria, which are to be used in choosing a knowledge representation formalism in a particular specification. We have not yet made a systematic attempt at relating the various criteria which we mention here to the various logics. Although we will make some preliminary remarks, it should be borne in mind that these remarks are only preliminary and are not the result of a careful analysis of the needs of the various domains.

Nature of the knowledge in the system

A first domain related criterion is the nature of the knowledge encoded in the knowledge base. One can distinguish between a model of the domain based on empirical associations, and a causal model of the domain. The terms 'shallow' versus 'deep' knowledge have been used [15].

Systems with shallow knowledge. The knowledge in systems with shallow knowledge is based on empirical associations which have been observed between certain phenomena. In general, there is no complete theory of the domain. As a consequence, the link between the phenomena is usually unclear. These systems therefore often have to reason with uncertainty as well.

Systems with deep knowledge. In systems with deep knowledge, there is often some causal model of the domain. Systems of this type may explicitly reason about causal relations although, as far as we know, there is no completely satisfactory logical analysis of causality, it seems certain that causality implies a necessary connection between cause and effect. Therefore, systems with deep models probably require a modal logic, a logic in which reasoning about necessity and possiblity are allowed.

The temporary nature of the data.

Another criterion which is important in determining the nature of the problem, is the temporary nature of the data. In the introduction data were defined as the session dependent knowledge the system is equipped with. If the data are completely determined before a session with the system, and the user could in principle give the system all the potentially relevant data before the expert system draws any inference, then we have a static problem. Otherwise, we have a dynamic problem.

Static problems. In static problems all the constraints which the solution has to meet can be specified before a session with the system. It is assumed that the problem at hand does not change during the session. The data which is potentially relevant for the solution is known beforehand by the user, who can supply the system with the relevant information as and when requested.

Dynamic problems In dynamic problems one cannot specify all the constraints on the solution beforehand. They may change as time goes on, as in monitoring systems such as VM [20]. An expert system dealing with dynamic problems may need to be able to reason explicitly about time, and it is therefore likely that we need temporal logics for this type of problem. We distinguish between two possibliities.

Predictable changes
The changes in the constraints on the solution might be predictable. Thus, given that we have all the information we need at time t, we can predict with certainty what the situation at t+n will be. Thus, although the constraints change over time, we are still able to predict what the constraints are going to be at some later stage if we know the constraints at an earlier time. In other words, the information one has at time t is enough to predict exactly what will happen in the future.

Unpredictable changes
A second possibility is that the constraints which hold at a later time cannot be completely reliably predicted from the information available at an earlier moment in time. Thus, the information at time is compatible with different future courses of events. One therefore needs a more complicated temporal logic than in cases where the changes are predictable.

Certainty of the information

The certainty of the information plays an important role as well. One can distinguish between perfect and imperfect information. There is a strong correlation between this criterion and the nature of the knowledge in the system, i.e. between this and the first domain related criterion.

Certain information. Certain knowledge is information which can be assumed to be true without qualifications. There is no doubt about either the truth of the knowledge, or about the applicability of the knowledge to the case in hand.

Uncertain terminology
One form of imperfection of knowledge is due to uncertainty about the exact meanings of the terms used in the domain in question. In the domain of marriage counselling for example, meanings of such basic terms as 'mental cruelty' and 'neglect of the partner' are hard to make precise. A related problem is that different types of people often use the same term in different ways. Thus, in the medical domain patients and doctors often differ in their understanding of medical terms, such as e.g. palpitations [2].

Uncertain data
A second type of uncertainty is due to uncertainty in the data. In signal processing applications for example, the data are often noisy and hence their identification is problematic, resulting in another source of uncertainty. Another source of uncertainty in signal processing applications is the fact that data will have to be reduced into a more compact format in order for the computer to be able to store them. This will lead to a loss of some information and hence uncertainty in the data.

Uncertain Knowledge
A third type of uncertainty is uncertainty in the knowledge itself. Often the connection between the antecedent and the consequent in an inference rule leaves room for some doubt. Most methods for dealing with uncertainty in expert systems are designed to deal specifically with this type of uncertainty.

TASK RELATED CRITERIA

We will now discuss task-related criteria, which we believe are relevant in choosing a control regime for the logic which was chosen on the basis of domain considerations. As in the case of domain criteria, we have not yet made a systematic attempt at relating task criteria and control regimes. Again, we will make some preliminary remarks but the same qualifications apply. In fact, we believe that it is even more difficult to formulate the correct correlations for task-related criteria because there is very little work comparing the relative merits of various problem-solving strategies.

The structure of the task

A first critertion which is relevant in the choice of the control regime is the type of task which the system is expected to perform. We distinguish between four types of task, namely classification or interpretation, monitoring, design and simulation or prediction. The list of tasks we propose is basically identical to the revised Hayes-Roth classification mentioned in section 2 of this paper. Some of the preliminary suggestions for control regimes are taken from [10, 11].

Classification. In a classification or interpretation task, the expert system is expected to analyse some data to arrive at a higher-level description of the situation in which the data was observed. The system is asked to identify some data with a specific element in a pre-determined set of higher-level situation descriptions. It is important to point out that in classification tasks it is practical to enumerate the solution space in advance (Cf [13, 14]). Although for a number of existing expert systems a complete enumeration of the solution space would be possible in theory, the solution space is so large that enumeration is not practical. An example which may illustrate this point is chess. Although it is in principle possible to enumerate all possile solutions, chess is after all a finite game, it is unlikely that it will ever be practical to actually do so.

The 'input' in a classification task is a number of data, some 'low-level' descriptions of observed events. There is of course the question of how to define what counts as a 'low-level' description of an observed event, and philosophers have discussed this question extensively. For our purposes however, the precise definition of this option is irrelevant. We can at least say that the notion is not an absolute one but is relative to a number of factors. Consider for example the speech domain. The 'low-level' descriptions of observed events are representations of the actual speech sounds produced by the speaker. The output of this module, i.e. the higher level situation description, would be, say a phoneme lattice. But on a higher level, one can see the phoneme lattice as the low-level situation description and a word of English as the higher-level situation description. The moral is that what counts as a low-level description is task dependent. After all, in the speech understanding system the output of one stage of the problem solving process is a low level situation description for the next stage. However, in general, the data in a classification task always constitutes a description of the situation which is at a lower level than the situation description which is the output.

Given that the solution space can be completely enumerated in advance, and given the fact that often the solutions can be hierarchically ordered, we follow [10, 11, 12] and suggest that an appropriate control regime for classification tasks might be top-down refinement or establish-refine:- each concept at a given level is tried to see if it explains the data. If it does then its subconcepts are tried until one fits the data, etc. If a concept does not fit the data, then it and all its subconcepts are rejected.

Monitoring. In a monitoring task, an expert system iteratively observes the behaviour of some system to extract features that are crucial for successful execution. Often the features are potential flaws in the plan according to which the system is performing.

Monitoring tasks have some characteristics in common with classification tasks. In particular, in both cases the solution space can be completely enumerated in advance. The main diffference between monitoring and straightforward classification is the fact that monitoring has an iterative aspect, and that the outcome on the nth iteration might depend on the outcome of earlier iterations.

The control regime for monitoring systems is likely to be bottom up:- incoming data is interpreted in order to determine if they correspond to possible problems for the performance of a system.

Design. In design tasks, the expert system constructs a complex entity which satisfies certain conditions and constraints. The complex entity under construction can be of very many different kinds. For example, the entity under construction might be some spatial configuration of some components. In fact, this is the task that R1/XCON performs ([22]; see also section on R1 below). But the entity under construction might also be a temporal configuration of actions, i.e. a plan. It follows that planning systems can be regarded as a special type of design system, namely those in which the object to be designed is a (temporal) sequence of actions. So although planning systems might appear to be very different from a spatial configuration system such as R1, they are very similar in structure and only differ in the domain over which they reason.

The conditions which a successful solution has to fit fall into two categories. The first category are those conditions or constraints which describe the present situation, as the present situation may constrain what solutions are possible and acceptable. The second category describes some requirements which the solution has to satisfy, i.e. they may specify a number of conditions the user may wish the solution to meet. Thus, in a planning system we would specify both the present situation and the desired situation.

The main difference with classification and monitoring systems lies in the fact that the solution has to be constructed and cannot be found in some pre-determined set of possible solutions. Sometimes, the solution space could in principle be enumerated, but the solution space is so large that enumeration is impractical. Earlier, we gave the example of chess.

Simulation. Another task which an expert system can be asked to perform is simulation or prediction. In tasks of this kind, the expert system is given a state of some system and a change in it, and asked to infer whatever other changes will happen or can be expected to happen. In simulation systems, as in design systems, it is usually impractical or impossible to enumerate the space of possible solutions completely in advance.

The control regime which is to be used in simulation systems is likely to be bottom up:- changes in state of the system are interpreted as changes in the behaviour of sub-systems, and this information can then be used to predict other changes in the behaviour of the overall system.

The role of the system in the interaction

A second factor which is important in the selection of a control regime is the role the expert system is to play in the interaction with the user and the status of the solution of the problem posed to it by the user.

Advisory expert systems. In advisory expert systems, the system puts forward possible solutions for the problem at hand. The user is then asked whether he or she accepts the conclusion in question. If the solution is accepted, then the particular problem is considered to be solved. If the user rejects the solution, then the system goes back and tries to find another solution. In advisory expert systems then, the user is the final authority on the acceptability of the solution put forward by the system. Users may reject the advice the system puts forward because they do not agree with the rules the system uses or because they do not agree with the rules the system uses or because they do not agree with the facts the system assumes to hold.

The fact that the user may reject the solution that the system proposes means that the expert system has to be built in such a way that it can look for a second solution. This has many consequences for the control regime used. For example, the system has to to be able to keep track of solutions which it has proposed earlier and which were rejected by the user. Also for efficiency reasons it would be preferable if certain statements which remained true because they were independent of the rules or the data that the user rejected, did not need to be derived again. The system should therefore be able to keep track of justifications for at least some of the propositions in its knowledge base.

Imperative expert system. In imperative expert systems, the system itself is the final authority on the acceptability of the solution. The system works on the assumption that the solution it puts forward is the only correct solution. It therefore does not try to find an alternative solution if the user rejects a particular solution the system has come up with. An example of an imperative system would be a system monitoring a power plant in a closed loop.

Criticising expert systems. Domain experts who are faced with a difficult problem may under certain circumstances consult another expert. Often this consultation will involve presenting to the other expert the problem and the preliminary solution to the problem which the first expert has come up with. The consulted expert will make critical remarks on the proposed solution. A recent development in the field of expert systems is the construction of such criticising expert systems. In systems of this type, the user presents the system with a problem and a solution and the system plays the role of a critic. The system analyses and comments on the proposed solution. An example of a system of this type is ATTENDING [23] in which plans for anaesthetic management are critically commented upon by the expert system. Another example is the ONCOCIN project at Stanford [26].

There are two possibliities which come to mind as the appropriate control regime for criticising expert systems. The choice between the two depends on what the basis should be for the system's critique. One can imagine that the system constructs a solution of its own and that it uses this solution to comment on the proposed solution. ONCOCIN follows this strategy. Or one can imagine that the system only looks for certain typical mistakes which are made by domain experts when they construct a solution to the problem. Thus, a typical mistake might be that domain experts forget to take a particular factor into account. A criticising system could then be told about typical mistakes and it could look at the proposed solution to see

if any of these mistakes are made. Obviously, the basis on which the system is to comment on the proposed solution is a major factor in choosing an appropriate control regime.

Time limitation in the task

Another factor which we mention here, even though it is almost certain that it will interact with some of the earlier criteria mentioned before and needs more research, is whether the problem solving process is time dependent or not. We can make a distinction between time-critical applications, where the system has to come up with a solution within a certain time limit, and non-time-critical applications. In time-critical applications one question which becomes important is whether the system should spend more time pursuing the line of reasoning it is on at the moment, or whether it should give up and look for another. In non-time-critical applications, the system could in principle pursue all possible paths until it has found out whether the path in question yields a solution or not although for reasons of efficiency this is often not feasible.

It is important to stress that the sort of time limitation we have in mind is limitations on processing time and not limitations on user time. Considerations on user-time, such as asking a minimal number of questions during a consultation of the system, are important for any system. What we have in mind are limitations on processing time. An example of the type of limitation we have in mind is VM (see below). VM is hooked up to some machinery for measuring certain vital functions of a patient in an intensive care unit. It takes certain measurements every 2 or 10 minutes. So, VM has to do all its reasoning about the data collected at one point in time before it takes a new set of measurements. Therefore, VM's task is time-critical in our sense. In general, it can be expected that expert systems which are hooked up to sensory equipment are time-critical.

One way which has been proposed in the literature for improving the efficiency of a system in general and which is particularly useful for dealing with time-critical applications is the use of meta-level reasoning. (See [31], and [50]; see for an extensive overview, [51]. By making the system able to reason about its own reasoning, it becomes able to answer the type of questions which it has to answer in time-critical applications.

It has been suggested to us that the above criterion might just be a special case of the more general phenomenon of lack of resources, where the resources might be internal (processing time, memory) or external (user's time, information available, money). Although we argued above that the criterion which we have in mind, might be slightly different from these other types of lack of resources, this question obviously needs more research. For the moment it is sufficient to say that we believe that the use of meta-level reasoning will turn out to be relevant for the other types resource limit as well.

OTHER RELEVANT FACTORS

There are two other factors which are relevant in choosing an inference engine for an expert system, but which are neither properties of the domain nor properties of the task. They are (i) whether the system can receive more information while it is reasoning, and (ii) whether the information that it has already received and that it is using can change during a run.

In systems where more information can become available during a run, one question which has to be answered is whether it is worthwhile to spend energy acquiring or waiting for this new information or to carry on anyway. An example is the decision whether the system should carry on or whether it should ask more questions of the user. We believe that meta-level reasoning is the technique which is also most suited for dealing with this type of problem, because the problem is also a question of resource management by the system.

Another problem is that some of the data that the system is currently using in its reasoning may change during the reasoning process, possibly forcing the system to reason non-monotonically. The situation is different from a monitoring system, because in a monitoring system, the data is sampled once in a given time period and it can then be used to reason with. Because incoming data is usually stored together with some indication about the time it arrived, incoming data provide more information rather than different information. New data does not invalidate old data. But in a system of the type we are considering here, new data may overwrite old data. Another way of formulating this is as follows:- let p be some parameter for which values are measured. Then, in a monitoring system one stores information of the following form:- the value of p at time t was x. Clearly, this information remains true if the same parameter is measured again and its value found to have changed at time t+n. But in a system of the type we are considering here, the information which is stored is of the form:- the value of parameter p is x. When one later measures parameter p again and p's value has changed, then the earlier information about the value of p is no longer valid.

There are various ways of dealing with the problem of changing data. A first and obvious way would be for the system to retract all the inferences which it had made using the old data. However, that would be very inefficient and unnecessary. After all, although some of the data may change, other parts will remain the same, and there is no reason to retract the inferences that the system has drawn on the basis of pieces of data which have not changed. The critical question therefore is which inferences the system has to withdraw when the data changes.

The best way to do this would be to store with every proposition in the knowledge base on the basis of what other pieces of information, if any, it was included. One could then use these justifications to determine whether a given proposition depended on a piece of data which has changed. Truth-maintenance or belief revision systems do exactly that. There are various papers which extend this idea. [33] is an early attempt at providing a truth maintenance system. [32] is a more modern system which avoids some of the problems associated with Doyle's system. Reichgelt [34] argues that truth maintenance systems csan also provide the answer to the problem of default reasoning.

CLASSIFYING EXISTING EXPERT SYSTEMS

In this section we will apply our criteria to a few existing expert systems. In this way we test our criteria and give an indirect argument for its potential usefulness. After all, if the criteria are going to be useful in the construction of new expert systems, then they must apply to already existing systems. The expert systems to which we apply our classification are chosen more or less at random. It is important to stress that in the discussion below we will do some 'rational reconstruction' of the expert systems in question.

MYCIN

One of the best known expert systems is MYCIN, a program which helps clinicians in diagnosing certain kinds of infections and prescribing courses of treatment [25]. MYCIN goes through four different steps in its reasoning:

— Determine whether there is a significant infection
— Determine the (possible) organism(s) involved
— Select a set of drugs
— Determine the best (combination of) drug(s) and the dosage.

These four steps fall into two groups of different types of tasks. The first two tasks are classification tasks, while the last two tasks are design tasks. Furthermore, although the domains for the four subtasks are obviously related, they have slightly different properties.

The first task, determining whether there is a significant infection, is necessary because the data which MYCIN works with are uncertain. There are two main problems. First, even the healthy body contains certain bacteria. Second, even if one finds foreign organisms in cultures grown from samples taken from the patient, then there is still the possibility thast the organism was introduced into the culture in the laboratory. Thus, the first task has to be undertaken because MYCIN deals with uncertain data.

The second task is again an example of a classification task. Based on a number of data MYCIN determines which set of organisms is most likely to be responsible for the infection in question. MYCIN does this by trying to find supporting evidence for each of the organisms it knows about. It only finds a combination of organisms when the evidence is such that it supports both organisms separately. Therefore MYCIN does not construct a diagnosis in the way that INTERNIST/CADUCEUS does [24].

The third and the fourth task are design tasks. Based on the list of organisms on whose existence the system has decided and based on certain properties of the patients, MYCIN constructs a scheme of drug(s), dosage(s) and method(s) of administration.

MYCIN is a dictatorial system when it is performing its first and second task; the user cannot reject any of its conclusions. However, MYCIN takes the role of an advisory system in the third and fourth part of its reasoning process. It allows the user to reject certain of its recommendations. If this happens, MYCIN will backtrack and try again to find an acceptable course of antibiotics.

MYCIN's problems are static problems. The data do not change during a run of the program. Also, MYCIN is not time-critical in the sense that VM is.

MYCIN uses a shallow model of its domain. It does not contain any information about the causal relations between the presence of certain organisms and certain infections, and between administering certain drugs and their effect on the diagnosed organisms. Since the connection between certain data and certain organisms is often not more than an empirical association, MYCIN not only uses uncertain data, but also contains uncertain knowledge.

HEARSAY-II

HEARSAY-II is one of the better known speech understanding programs. It interprets spoken requests for information from a database about books and papers in Artificial Intelligence. In September 1976, configuration C2, HEARSAY-II had a lexicon of 1011 words, and a rather complicated grammar. We will not go into the architecture of HEARSAY-II, or into the way in which the system interprets sentences in any great detail. The interested reader is referred to [18b].

Unlike earlier speech understanding systems, HEARSAY-II does not just interpret single words. The system is also able to interpret continuous speech. When HEARSAY-II receives a speech signal, it derives four parameters directly which it uses and transfers it from an analogue representation into a digital one. Following this, it splits the signal into non-overlapping segments and classifies them. On the basis of the classificationof these segments, it forms hypotheses about possible words in the signal. Finally, once it has what one might call a word matrix, it tries to construct an interpretable sentence out of it, using syntactic and semantic knowledge.

From the above description, it is clear that HEARSAY-II performs a number of different tasks. One can roughly divide the tasks which HEARSAY-II goes through into two stages:- the formation of word hypotheses, and the combination of the word hypotheses into interpretable sentences. The first stage is an example of a classification task because the possible solutions are all pre-enumerated in the lexicon. The second stage is an example of a design task because the system has to construct interpretable sentences out of the set of word hypotheses. The set of interpretable sentences is not explicitly stored.

HEARSAY-II is a dictatorial system:- one cannot reject the solution it comes up with and ask it to find an alternative.

The type of problem which HEARSAY-II is built to solve is a static problem. Once the speech wave is stored, the data does not change during the session. HEARSAY-II does have some conception of time: it makes hypotheses about which particular word occurred between certain points in the speech sound. However. it is not sensitive to temporal changes in the data as VM is.

HEARSAY-II does not have a causal model of its domain but uses a shallow model.

HEARSAY-II uses imperfect knowledge. Its data in particular is uncertain and part of the difficulty of speech understanding lies in the uncertainty of the data. Terminology and the knowledge which are used to transform the speech wave into a higher level description are not uncertain however.

HEARSAY-II does not interpret the speech signals in real time. Rather, it buffers the speech wave and then tries to understand it. HEARSAY-II therefore does not perform a time-critical task.

VM

Fagan, Kunz, Feigenbaum and Osborn [19] describe VM, a program intended to provide diagnostic and therapeutic suggestions about patients in intensive care units. It is able to recognise untoward events and suggests corrective action. It is also able to suggest adjustments to the therapy based on a long-term assessment of patient status and therapeutic goals. VM detects possible measurement errors. Finally, VM maintains a set of patient-specific expectations and goals for future evaluation.

VM contains five different sets of rules which it considers in order. First, VM characterises measured data as reasonable or spurious. Secondly, it determines the therapeutic state of the patient. Thirdly, it establishes expectations of future values of measured variables. Fourthly, it checks the physiological status of the patient. Finally, it checks compliance with long-term therapeutic goals.

The basic cycle of VM performs a classification task. Patient measurements are analyzed in order to arrive at a description of the patient's status. After it has done this, VM gives some suggestions about corrective actions to be taken in response to untoward events or adjustments to the proposed therapy. We can thus say that the basic loop of VM, as so many expert systems in medicine, first performs a classification task and then a design task. First, it interprets patient data and then constructs suggestions about how to change the therapy. However, since VM repeats this basic cycle to keep track of changing patient behaviour, the full system performs a monitoring task, rather than a classification task.

VM produces periodic summaries of the patient's status and what the authors call suggestions to clinicians. It is not clear however what happens if the clinician rejects these suggestions and it is therefore impossible to say whether VM is an advisory or a dictatorial expert system.

VM clearly works with dynamic problems. It is not possible to specify all the constraints before the session. Moreover, although VM makes predictions about likely future values for the measurements one of the

crucial features of the domain is that the values are not completely predictable. In fact, VM uses discrepancies between expected values and real values to base its suggestions on.

The type of change VM encounters is different measurements rather than more measurements. So, VM could in principle look at the data which was collected at an earlier point in time, compare those with the data it has collected now, and only withdraw those inferences which depend on data which has changed. However, VM does not use this strategy. Rather it starts from scratch, so to speak, whenever new data comes in. However, note that VM stores earlier data and uses them in its reasoning process.

The type of model which VM uses is not entirely clear from the description of VM in Fagan et al. However, it seems clear that VM does not reason from first principles and one can safely assume that the model which is used is shallow.

Since VM can determine possible measurement errors, its data is uncertain. If the data in a system is always certain, then there would be no need for a mechanism which enabled the system to reject some of them. VM's terminology is not uncertain and its knowledge base consists of straightforward production rules which do not contain certainty factors or any other way of coping with uncertainty.

Since VM has to make its recommendations before the next data arrives, the application is time-critical. However, it seems to be the case that VM does not contain any special features to deal with interrupts as new data come in. Data are measured every 2 or 10 minutes, and VM seems to be able to do most of its reasoning between two data reading points.

R1

R1 [22] is a program for configuring VAX computer systems. Its input is a customer's order and its output is a set of diagrams displaying the spatial relationships among the components on the order. The main task has two sub-tasks. First, the customer's order must be checked to make sure that it is complete; if it is not, the missing components must be added. Secondly, the spatial relationships between all these components must be determined.

The first subtask R1 performs is data checking. It will be remembered that we defined data as session dependent information. The data which R1 uses therefore is the customer's order. Since the order can be incomplete, R1 is able to add the missing components. R1 can thus not accept the data without any qualification and therefore the data that it uses is uncertain.

The second sub-task is R1's central task. It is determining the spatial relationships between the different components. Since R1 does not select from a number of pre-enumerated possibliities but rather constructs the spatial specification, we are dealing with a design system.

It does not seem to be the case that the user can reject R1's solutions and ask it to construct another configuration. R1 is therefore a dictatorial system.

Once R1 has made the customer's order complete, the data no longer changes, i.e.nothing is added or taken away from the list of components which make up the customer's order. R1 thus solves a static problem.

R1's rules are not based (at least not explicitly) on some causal model of the domain. It therefore uses a shallow model.

As said before, R1's data is uncertain in the sense that they can be incomplete. However, both the terminology used and the knowledge are perfect in the sense that they are not uncertain.

Finally, R1' task is not time-critical.

CONCLUDING REMARKS

In this paper, we argued that the presently available tools for expert systems are not sufficient. Shells are too rigid while high-level programming are too unstructured. We suggested that what was needed were a number of logical languages for knowledge representation and a number of control regimes. We then formulated criteria which we believe will be relevant in the formulation of these guidelines. We suggested that there were certain criteria related to the domain about which the system was to contain knowledge. These criteria would be useful in formulating guildelines about which logical lanaguage to choose as the knowledge representation language. Other criteria were related to the task the system was to perform. They should be used in formulating guidelines for choosing control regimes. Obviously, the criteria proposed here have to be followed by more explicit advice on what logics are most suited given the properties of the domain, and which control regimes are best given the properties of the task. Although we have made some preliminary remarks in this paper, this question needs to be tackled in a more systematic way, and we hope to do so in the near future.

Acknowledgements

The research reported in this paper was done in the Alvey-funded project "A Flexible toolkit for Building Expert Systems". We would like to thank Maarten van Someren, Peter Jackson and John Fox for their comments on an earlier version of the paper, as well as our industrial collaborators at GEC Research.

BIBLIOGRAPHY

Related Work

[1] Alvey, P. (1983). Problems of designing a medical expert system. *Expert Systems 83,* pp.20-42.

• Alvey describes the problems encountered with simple expert systems shells.

[2] Beck, E., J. Francis & R. Souhami (1983) *Tutorials in differential diagnosis (2nd edition)* London: Pitman.

• The book describes the problems encountered by physicians during the diagnosis of a patient.

[3] Bennett, J. (1985) ROGET: A knowledge-based consultant for acquiring the conceptual structure of an expert system. *Journal of Automated Reasoning,* 1, 19-74.

• ROGET is a system in the area of knowledge acquisition built on the assumption that different kinds of knowledge need different conceptual structures to support them.

[4] Brachman, R. (1979) On the epistemological status of semantic networks. In N. Findler (ed.) *Associative networks: representation and use of knowledge by computer.* New York: Academic Press

• Brachman gives an analysis of the different levels that can be used to describe semantic networks.

[5] Breuker, J. & B. Wielinga (1985) Use of models in the interpretation of verbal data. University of Amsterdam. To appear in: A. Kidd (ed) *Knowledge elicitation for expert systems: A practical handbook.* New York: Plenum Press.

[6] Breuker, J. & B. Wielinga (1985) KADS: Structured Knowledge Acquisition for Expert Systems. VF-Memo 40, University of Amsterdam

• Like ROGET this is a system in the area of knowledge acquisition based on the premiss that different kinds of knowledge need different conceptual structures to support them.

[7] Brown, D., Chandrasekaran, B., Expert Systems for a class of mechanical design activity *Proceedings of IFIP Working Group 5.2 Working Conference on Knowledge Engineering in Computer Aided Design,* Budapest, Hungary, Sept '84.

[8] Brown, D., Chandrasekaran, B., Knowledge and control for design problem solving. Technical report from the Ohio State University Dept. of Computer and Information Science, Laboratory for AI Research, 1985.

• These two papers contain a detailed description of one of the prototypical control regimes that Chandrasekaran et. al. have identified, and which they use for design tasks.

[9] Chandrasekaran, B. (1983) Towards a taxonomy of problem-solving types. AI Magazine, 4, 9-17.

[10] Chandrasekaran, B. (1985) Generic tasks in expert system design and their role in explanation of problem solving. Paper presented to the NAS/ONR Workshop on AI and distributed problem solving.

[11] Chandrasekaran, B., (1985) Expert Systems: Matching Techniques to Tasks. in: *Artificial Intelligence Applications for Business,* W. Reitman (ed), Ablex Corp.

• In these papers Chandrasekaran makes an attempt to relate the architectures and representations for expert systems to the types of tasks for which they are appropriate. His approach is closely related to the one taken in the current paper.

[12] Stiklen, J., Chandrasekaran, B., Josephson, J. (1985) Control Issues in Classificatory Diagnosis *Proceedings of the Ninth International Joint Conference on Artificial Intelligence,* IJCAI '85, Los Angeles, pp. 300-306.

• This paper is to the prototypical task of diagnosis what the previous two papers are for design: a detailed description of the control regime proposed by Chandrasekaran et. al.. for executing this kind of task.

[13] Clancey, W. (1984) Classification problem solving. *AAAI-84,* 49-55.

• Clancey describes the prototypical task of classification, and the appropriate control regime.

[14] Clancey, W. (1985) Heuristic Classification. Stanford University Knowledge Systems Labaoratory, Technical report KSL-85-5.

• Clancey describes how a broad range of problems are solved by a particular problem solving method, which he calls heuristic classification. Heuristic classification can be seen as a prototypical task, and Clancey describes the characteristic inference structure for it.

[15] Hart, P. (1982) direction for AI in the eighties. *SIGART Newsletter, 79.*

[16] Hayes-Roth, F., D. Waterman & D. Lenat (eds.) (1983). *Building expert systems.* Reading, Mass.: Addison-Wesley.

• One of the standard text-books on expert systems. It contains two classifications of expert systems according to different sets of criteria.

[17] Jackson, P. (1985) reasoning about belief in the context of advice-giving systems. In M. Bramer (ed.) *Research and development in expert systems.* pp 73-83, Cambridge: Cambridge University Press.

• A paper on the requirements for control and representation in advice-giving systems.

[18] Stefik, M., J. Aikins, R. Balzer, J. Benoit, L. Birnbaum, F. Hayes-Roth, & E. Sacerdoti (1982) The organization of expert systems: A tutorial. *Artificial Intelligence,* 18, 135-172. Also in F. Hayes-Roth, D. Waterman & D. Lenat (1983).

Expert Systems

[18b] Erman, L. and Lesser, V. (1980) The HEARSAY-II Speech Understanding System: A tutorial. In: *Trends in speech recognition.* Wayne and Lea (eds.)

[19] Fagan, L.M., Kunz, J.C., Feigenbaum, E.A. and Osborn, J. (1979) Representation of dynamic clinical knowledge: Measurement interpretation in the intensive care unit. *Proceedings of the Sixth International Joint conference on Artificial Intelligence,* IJCAI '79, pp. 260-262.

[20] Fagan, L. (1980) *VM, Representing time-dependent relations in a medical setting.* Ph.D. Diss. Computer Science Department, Stanford University.

• Two papers that describe VM, an expert system for monitoring ventilation management.

[21] Jackson, P. (1986) *Introduction to expert systems.* Reading, Mass.: Addison-Wesley.

• A good textbook on expert systems.

[22] McDermott, J. (1982) R1: A rule-based configurer of computer systems. *Artificial Intelligence,* 19, 39-88.

[23] Miller, P. (1983) Medical plan analysis: The attending system. *IJCAI-83,* 239-41.

• A description of one of the first expert systems with a so called critical attitude.

[24] Pople, H. (1977) The formation of composite hypotheses in diagnostic problem-solving: An exercise in synthetic reasoning. *IJCAI-77,* 1030-37.

• A description of the work on INTERNIST/CADUCEUS.

[25] Shortliffe, E. (1976) *Computer-based medical consultation: MYCIN.* New York: American Elsevier.

[26] Shortliffe, E. (1985) Update on ONCOCIN, A chemotherapy advisor for clinical oncology. *Medical Informatics*

Expert Systems Tools

[27] Bobrow, D. & M. Stefik (1983) *The LOOPS manual,* Xerox.

[28] Intellicorp, (1984) *The Knowledge Engineering Environment.*

[29] Richer, M., Five Commercial Expert System Tools: An Evaluation *The Artifical Intelligence Report,* Vol. 2, No. 8, August 1985.

• A comparison of five of the major commercial knowledge based programming environments: S.1, KEE, ART, DUCK, and Knowledge Craft.

[30] Williams, C. (1983) ART, the advanced reasoning tool, conceptual overview. Inference Corporation.

Truth Maintenance Systems

[31] de Kleer, J., J. Doyle, G. Steele Jr, & G. Sussman (1977) AMORD: Explicit control of reasoning. *SIGART Newsletter,* 64,116-25.

[32] de Kleer, J. (1986) An assumption-based TMS. *Artificial Intelligence,* 28,127-62.

[33] Doyle, J. (1979) A truth-maintenance system. *Artificial Intelligence,* 12, 231-72.

• Two papers that describe different approaches to the problems of truth-maintenance and non-monotonicity.

[34] Reichgelt, H. (1985) *Reference and quantification in the cognitive view of language.* PhD Thesis. University of Edinburgh.

Uncertainty

[35] Fox, J., (1980) Alternatives to Bayes? A quantitative comparison with rule-based diagnostic inference. *Methods of Information in Medicine,* Vol. 19, pp. 210-215.

[36] Mamdani, E.H., Gaines, B.R. (eds.), *Fuzzy Reasoning and its Applications* Academic Press, 1981.

• An interesting book with a collection of papers on handling uncertainty by the main researchers in the field.

[37] Negoita, C. (1985) Expert systems and fuzzy systems. Menlo Park, Cal.: Benjamin/Cummings.

[38] Parker, R. (1985) The treatment of uncertainty in expert systems. Technical Report 21, Internal project paper for a flexible toolkit for building expert systems.

[39] Whalen, Th., Decision making Under Uncertainty with Various Assumptions about Available Information. *IEEE Transaction on Systems, Man and Cybernetics,* Vol. SMC-14, No. 6. Nov/Dec 1984, pp. 888-900

[40] Whalen, Th, Schot, B., (1985) Alternative Logics for approximate reasoning in expert systems: a comparative study. *International Journal of Man-Machine studies,* Vol. 22, (1985), pp. 327-346.

• All these papers are good overviews of a number of the most well known methods for reasoning with uncertainty.

[41] Zadeh, L. (1979) A theory of approximate reasoning. In J. Hayes, D. Michie & L. Mikulich (eds) *Machine Intelligence 9.* New York: John Wiley.

• One of the standard papers on fuzzy logics.

Logic

[42] Haack, S. (1974) *Deviant logic: Some philosophical issues.* Oxford: Oxford University Press.

• A good description of the notion of deviant logics.

[43] Hughes, G. & M. Cresswell (1968) *An introduction to modal logic.* London: Methuen.

[44] Mates, B. (1972) *Elementary logic (2nd edition)* Oxford: Oxford University Press.

• An introductory textbook on classical logic.

[45] Moore, R. (1985) A formal theory of knowledge and action. In J. Hobbs & R. Moore (eds) *Formal theories of the commonsense world.* Norwood, N.J.: Ablex.

• A proposal for an implementation of modal logics in first-order predicate calculus.

[46] Rescher, N. & A. Urquhart (1971) *Temporal logic.* Berlin: Springer-Verlag.

[47] Turner, R. (1984) *Logics for artificial intelligence.* Chichester: Horwood.

• A good introduction on the use of non-standard logic in AI systems.

[48] van Benthem, J. (1982) *The Logic of time.* Dordrecht: Reidel.

[49] van Dalen, D. (1983) *Logic and struture (2nd edition)* Berlin: Springer-Verlag

Meta-level Reasoning

[50] Davis, R & B. Buchanan. (1977) Meta-level Knowledge: Overview and applications. *IJCAI-77,* 920-27.

• A general overview of the possibilities of using meta-level knowledge in the context of knowledge based systems.

[51] van Harmelen, F. (1986) Improving the efficiency of meta-level reasoning. Discussion Paper, Department of Artificial Intelligence, University of Edinburgh.

• An overview and discussion of a number of systems in the literature that use meta-level reasoning.