THE MODIFIABILITY OF RULE-BASED EXPERT SYSTEMS

A thesis submitted for the degree of Doctor of Philosophy.

by

Leonardo Bottaci

Brunel University, Dept. of Electrical Engineering, London, 1985.

Acknowledgements

I would like to gratefully acknowledge the kind assistance of my supervisor Dr. Les Johnson. In addition I gained much from discussions with Professor Edwin Borg-Costanzi and fellow students, Simon Murdoch, Peter Elleby and Elpida Keravnou.

Leonardo Bottaci

Brunel University, 1985.

ABSTRACT

This thesis examines the claim that rule representations of knowledge are conveniently modified. The thesis falls into two parts and in the first, a precise notion of a convenient modification, called an extension, is developed and it is shown that extensible knowledge-bases are very convenient to modify and develop. We show that rule representations of knowledge are extensible only if they incorporate a suitable organisation of knowledge. Furthermore, we show that non-rule representations with a suitable organisation of knowledge are also extensible. We therefore conclude that rule representations of knowledge are no more or less extensible than non-rule representations. In the second part, we consider the more pragmatic aspects of knowledge-base modifiability. In each of two detailed case studies, we compare the modifiability of a rule-based expert system with its "second generation" counterpart which incorporates non-rule representations of knowledge. We conclude that in practice the modifiability of extensible knowledge-bases can be compromised if the organisation of knowledge is represented obscurely. Above all, the thesis emphasizes the importance of the organisation of knowledge in an expert system.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

RESEARCH TOPIC

There have been a number of claims asserting that rule representations of knowledge are, amongst other things, convenient to modify. It is these claims that this thesis investigates.

In order to appreciate why it is important to examine these claims we need to consider the status of the rule scheme as a representation for knowledge in expert systems, the incremental nature of expert system development and the technological aspect of expert systems. In the following paragraphs we discuss each of these factors in turn.

The rule scheme occupies a predominant place in the representation of knowledge in expert systems.

> Rule-based systems (RBSs) constitute the best currently available means for codifying the problem-solving know-how of human experts. Experts tend to express most of their problem-solving techniques in terms of a set of situation-action rules, and this suggests that RBSs should be the method of choice for building knowledge-intensive expert systems.
>
> Hayes-Roth, 1985, p921.

Many of the most influential expert systems have rule representations of knowledge. The DENDRAL system (Buchanan and Feigenbaum, 1978) is designed to provide assistance to chemists with structure elucidation problems in organic chemistry. DENDRAL contains rules encoding relationships between physical, chemical and spectroscopic data and the plausible structure of unknown compounds. The MYCIN system (Shortliffe, 1976) helps a physician diagnose the cause of a patient's infection and recommend a suitable therapy for that patient. The knowledge-base contains about five hundred rules, mostly about

meningitis infections. The PROSPECTOR system (Duda et al, 1979) is intended to help a geologist in evaluating the likelihood of the presence of particular ore deposits at a site. The expert's geologists knowledge is represented as a set of rules linked together to form an inference network. The R1 system (McDermott, 1982) is able to configure the numerous components that are used to construct a Digital Equipment Corporation VAX[1] computer. The knowledge used by the system consists of a large set of rules about the way in which components should be assembled, together with a data-base of facts about each component.

The authors of all these systems have claimed[2] that the success of their projects relied largely on the use of artificial intelligence techniques and that the use of the rule representation scheme has been a key factor.

THE IMPORTANCE OF MODIFIABILITY

The power of an expert system to solve problems arises from the use of domain specific[3] expert knowledge.

> ... to enhance the performance of AI's programs, knowledge is power. The power does not reside in the inference procedure. The power resides in the specific knowledge of the problem domain.
>
> Feigenbaum, 1983, p2.

Consequently, the construction of an expert system is largely the construction of a representation of expert knowledge and this is said to resist the application of conventional system development methods

---

[1] VAX is a trademark of the Digital Equipment Corporation.

[2] These claims are detailed in chapter three.

[3] This is in contrast to earlier work on problem solving in AI that concentrated on general-purpose problem solving methods, devoid of specific problem domain knowledge. It was later found that such general-purpose methods were too weak to solve complex problems.

in which detailed specifications are produced before implementation is begun. For knowledge-based systems, the development method centres on the construction of a working prototype early on in the project and then successive refinements are made until the required performance is achieved (see inner loop of figure 1.1).

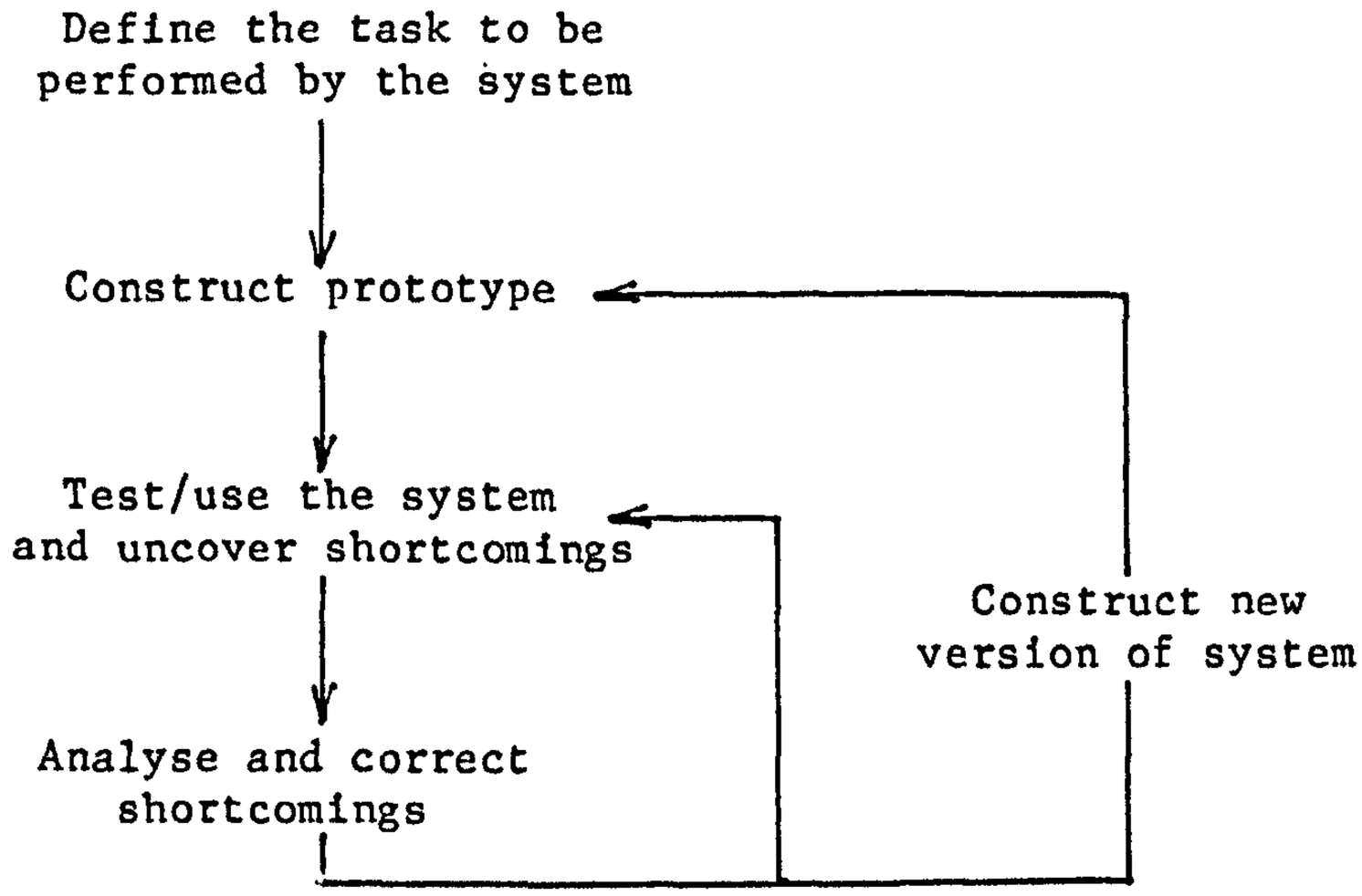


fig 1.1

The "incremental" expert system development method

> With open-ended problems and ill-defined bodies of knowledge, it was obvious that building a knowledge base was more a matter of iteration and refinement than bulk transfer of facts.
>
> Buchanan, 1980, p270.

> Because it takes experimentation to achieve high performance, an expert system evolves gradually. This evolutionary or incremental development technique has emerged as the dominant methodology in the expert systems area. The procedure of extracting knowledge from an expert and encoding it in program form is called knowledge acquisition. This transfer and transformation of problem-solving expertise from a knowledge source to a program is the heart of the expert-system development process.
>
> Hayes-Roth, 1983, p23.

The need to continually experiment and refine the representation of knowledge is said to be due to the nature of the expertise.

> The knowledge of an area of expertise -- of a field of practice -- is generally of two types: a) Facts of the domain -- the widely shared knowledge that is written in textbooks, and in journals of a field; that constitutes the kind of material that a professor would lecture about in a class. b) Equally as important to the practice of a field is heuristic knowledge -- knowledge which constitutes the rules of expertise, the rules of good practice, the judgmental rules of the field, the rules of plausible reasoning. ... The programs [expert systems] I will describe require, for expert performance on problems, heuristic knowledge to be combined with the facts of the discipline.
>
> Feigenbaum, 1983, p1.

> When that knowledge is firm, fixed, and formalized, algorithmic computer programs that solve problems in the domain are more appropriate than heuristic ones. However, when the knowledge is subjective, ill-codified and partly judgmental, expert systems embodying a heuristic approach are more appropriate. This type of knowledge is rarely formulated in a fashion that permits simple translation into a program.
>
> Buchanan et al in Hayes-Roth (ed), 1983, p127,128.

Thus we see the importance of the incremental development method for expert systems.

The use of the incremental development method implies that the representations of expert knowledge are subject to continual modification therefore the successful development of an expert system depends crucially on the use of modifiable representations of knowledge.

Knowledge represented within the rule scheme is said to be easily modifiable and therefore if these claims are well founded the use of the rule scheme solves the problem of constructing representations of knowledge that are suitable for systems developed incrementally.

In summary, we can say that the claims about the modifiability of rule representations of knowledge deserve close examination because:

a) there is a predominance of the scheme for the representation of knowledge,

b) the incremental nature of expert system development requires that the knowledge-base be continually modified and hence constructed from modifiable representations of knowledge.

RELATED WORK

In order to locate this thesis in a wider context we now discuss its relation to other work in the expert systems field. There are two ways in which this thesis can be compared to related work. Firstly, a comparison can be made in terms of research method and secondly in terms of research results. We start by discussing the research method.

The research method adopted in this thesis has much in sympathy with that adopted by Clancey (1983) in his discussion of MYCIN and explanation of reasoning within expert systems. Within this method, the research is essentially done "bottom up"; moving from the investigation of practical, system specific, problems towards general conclusions. For example, Clancey (ibid) uses specific problems with the explanatory abilities of MYCIN as a basis for his epistemological framework for explanation. This framework arose from the need to recognise general domain independent distinctions between various kinds of knowledge including structural, strategic and support knowledge, in order to provide adequate explanations.

> The demands of tutoring provide a "forcing function" for articulating the structure of the rule-base and the limitations of the program's explanation behavior. These insights have implications for generating and explaining consultative advice, and modifying the system.
>
> Clancey, ibid, p216

Our approach in this thesis is also "bottom up" in that practical, system specific, modification problems are taken as the starting point. The solution to these specific modification problems

leads us to focus on more general epistemological, representation and engineering issues. Although, the research method is fairly typical of the AI field in that it relies on the testing and subsequent rational analysis of programs, it is novel in that it is not the performance of the program that is being tested but its modifiability. This means that we are concerned with the structure and content of a program as well as its behaviour.

The second aspect of comparison with related work concerns the research results. In this thesis we argue that the use the rule representation scheme does not solve the modifiability problem for expert systems and that the modifiability of a system is a function of its organisation of knowledge. It has also been suggested (Shortliffe, 1976) that representing knowledge as rules renders it available for explanations to the user. Clancey (1983) has criticised this view and concluded that providing explanations requires the presence of a suitable organisation of knowledge (in accordance with the epistemological framework mentioned above).

More generally, Johnson (1985) has argued that problems such as the modifiability of the knowledge-base, capacity for explanation of reasoning, natural dialogue structure for the consultation, etc. are not independent problems but separate manifestations of the underlying problem of capturing a suitable organisation of knowledge, a competence model.

> All the above problems [modifiability, explanations, dialogue structure and teaching] have been repeatedly identified in the literature on expert systems. However, they are treated as independent problems rather than as the side effects of the (global) problem of how to elicit and represent the model of competence underlying the expertise displayed in a given domain.
>
> ibid, p34.

Looking even further afield at related work; An important theme in the work of Johnson (1985), Newell (1982), Clancey (ibid, 1984) and Sticklen et al (1985) is the distinction between knowledge and its representation in some scheme.

Johnson (ibid) distinguishes between an expert system's model of expert competence and the representation of that model in some scheme.

> The more accurately we can capture the model of competence in a knowledge-base representation scheme or a hybrid of schemes the better the cognitive coupling between the computer and the user and the closer the recommendations and/or decisions will be to the human experts'.
>
> In short, having a competence model is a necessary standard for determining what constitutes a good knowledge representation.
>
> Johnson, ibid, p29

Newell (1982, p99) distinguishes between the program or representation that lies at the symbol level and the knowledge represented in the program that lies at the knowledge level.

> As is true of any level, although the knowledge level can be constructed from the level below (i.e. the symbol level), it also has an autonomous formulation as an independent level.

The distinction between knowledge and its representation is also present throughout Clancey's work. The epistemological framework for explanation (Clancey, 1983a) is independent of any representation scheme. Clancey's (1985) formulation of the classification problem solving method is done entirely at the knowledge level.

> ... the classification model provides a knowledge level analysis of programs, as defined by Newell (Newell, 1982). It "serves as a specification of what a reasoning system should be able to do." Like a specification of a conventional program, this description is distinct from the representational technology used to implement the reasoning system.
>
> Clancey, ibid, p53.

This general point is echoed by Sticklen et al (1985, p300).

There is also an increasing awareness that the problem solving behaviour of knowledge-based reseasoning systems is best understood at what Marr [1976] has called the information processing level, or the knowledge-base level as it has recently been called by Newell [1982]. E.g., at the implementation language level MYCIN's diagnostic action can be thought of as backward-chaining, while at the information processing level its activity is best understood as a form of classification.

The distinction between knowledge and its representation is also an important theme in this thesis. However, rather than stressing the epistemological implications of the distinction we concentrate on the technological and engineering implications. Rather like in conventional programming where the choice of language to implement a specification has important engineering implications, the choice of representation scheme for some given expertise has analogous implications for building expert systems. A representation scheme that obscures the structure of the problem solving method, ignoring knowledge level distinctions, for example, will not make a suitable "implementation language".

OVERVIEW OF THESIS

The time consuming nature of the expert system development process has led some authors (for example, Feigenbaum, 1983) to suggest that the knowledge used in such systems should be acquired automatically. The prospect of automatic knowledge-base construction suggests that it may be possible to neatly "side step" the problem of knowledge-base modifiability. In the second chapter we consider the feasibility of this approach to expert system development and conclude that although there exist knowledge acquisition tools that can automatically construct knowledge-bases for use in certain simple and well defined domains, they do not constitute a general purpose method for knowledge-base construction.

The investigation proper begins by collecting together a representative sample of claims made about the modifiability of rule representations of knowledge. These claims are not as straightforward as they might first appear since the concepts of a representation scheme and modifiability are complex. The investigation into these claims therefore begins by clarifying these concepts.

We note that in practice knowledge is represented in a representation language that is an instance of one or more schemes. This complicates matters since two different representation languages may both be instances of the same scheme and yet possess different features.

The clarification of the concept of modifiability leads to two distinct notions of that concept. The first notion is the least interesting of the two and is analogous to the sort of modifiability that results from the use of a high level, or problem oriented, language as opposed to a low level assembly language. The second notion of modifiability is best thought of as the ability to extend the knowledge-base without "disrupting" any of the knowledge already represented there.

We argue that this latter notion of a convenient modification is the sort of modifiability that is most valuable for systems constructed incrementally. The reason for this is that the efficient incremental construction of any object depends on the ability to make changes to it without destroying previously constructed parts of the object. To take a mundane but familiar example, building a house of cards is difficult precisely because adding a new card often topples a large number of those previously arranged. Similarly, large badly structured programs are difficult to extend.

Armed with the notion of a knowledge-base extension, we consider the question of whether rule representations of knowledge allow extensions or not. In answering this question we show that, depending on the particular modification being considered, some rule representations can be extended although others cannot. We go on to argue that the suitability of a particular rule representation for various extensions is determined, not by the form of the representation, but by the organisation of the knowledge represented.

Organisations of knowledge exist at the knowledge level and are independent of their representation. Roughly, the distinction between an organisation of knowledge and its representation, is analogous to the distinction between a language independent description of an algorithm (inclusive of data structures) and an implementation of that algorithm in a particular programming language.

Given that the extensibility of rule representations is determined by the organisation of knowledge, the question arises as to whether the same is true of non-rule representations of knowledge. We show that the extensibility of a non-rule representations of knowledge is, again, determined by the organisation of the knowledge. Consequently, we argue that extensions to rule representations are no more or less convenient than extensions to non-rule representations.

A justification often put forward for the modifiability of rule representations is their so called "modularity". A system is modular if it is composed of modules. However, this is not the only meaning of the term 'modularity' that is attributed to the rule scheme. The additional meaning is 'modularity' in the sense of having modules suitable for various modifications. In the case of the rule scheme, modules representing "small independent chunks" of knowledge. We show that Davis and King (1977) confuse these two meanings and consequently

mistakenly argue that the first kind of "modularity" leads to the second. We go on to show that "modularity", in the sense of having modules suitable for modification purposes, instead arises from the particular organisation of knowledge.

It should be noted that some authors have expressed dissatisfaction with the rule representation scheme. Aikins (1983) and Clancey (1983) have both argued that representing knowledge as rules can sometimes lead, amongst other things, to obscure representations that hinder modifications to the knowledge. Each of these authors has constructed a "second-generation" version of a rule-based system and both of them have attempted to overcome the shortcomings, as they see them, of the rule representation scheme, by using non-rule representations of knowledge.

In each of two case studies we compare the modifiability of the original rule-based system with its "second generation" counterpart. The first study is a comparison of PUFF, a rule-based system for interpreting pulmonary function test results (Kunz, et al, 1978), with CENTAUR (Aikins, 1983), a system which uses frames and rules for performing essentially the same task. The second study is a comparison of MYCIN with NEOMYCIN (Clancey and Letsinger, 1981). NEOMYCIN also uses frames and rules for performing essentially the same task as MYCIN.

In each study we show that the use of a suitable representation scheme can be useful in clearly displaying the organisation of knowledge used by the system and since in practice understanding an organisation of knowledge is a prerequisite for modifying it, a suitably chosen scheme or schemes can improve the modifiability of a system. However, merely representing the same organisation of knowledge in a different scheme does not affect the extensibility of

that knowledge.

The thesis concludes that, in terms of knowledge-base extensibility, rule representations of knowledge are not inherently any more conveniently modifiable than representations in a non-rule scheme. Rather the extensibility of a representation of knowledge does not depend on the scheme in which the knowledge is represented but rather on the organisation of that knowledge. Furthermore, given a particular organisation of knowledge, certain representation schemes will be better suited for representing that knowledge clearly than other schemes. Even within a particular scheme the choice of representation language is important for the clarity of the representation. Therefore, to the extent that a clear and perspicuous representation of knowledge is helpful in the modification of that knowledge, the choice of representation scheme and language does affect the modifiability of the system. However, the choice of scheme or language for a perspicuous representation depends on the organisation of knowledge in the domain and cannot be determined a priori. There are important implications here for the process of knowledge elicitation. Knowledge elicitation should not begin with a commitment to a particular representation scheme instead it should be concerned with identifying and characterising the organisation of the expert's problem solving knowledge.

CHAPTER TWO

AUTOMATIC KNOWLEDGE-BASE CONSTRUCTION

INTRODUCTION

The development of systems that achieve expert level performance is a difficult and time consuming job. Such systems must be developed incrementally and if this kind of development method is to be at all efficient the representations of knowledge must be easily modifiable. It is for this reason that claims for the modifiability of knowledge-bases constructed with the rule scheme are important. However, given the time consuming nature of knowledge-base construction it has been suggested that knowledge-bases might be constructed automatically. If the incremental development process can be "bypassed" in this way the the issue of the modifiability of rule representations is no longer of such major importance.

In this chapter we argue that the currently available methods of automatic knowledge acquisition are not adequate to replace the "manual" iterative method and consequently the modifiability of the rule scheme remains an important issue. We begin by discussing some examples of the application of automatic knowledge acquisition techniques to the problem of constructing a knowledge-base automatically. On the basis of these examples we consider the prospects for the use of these techniques as a general method for knowledge-base construction. We argue that there are important aspects of the problem of building expert systems that automatic knowledge acquisition techniques do not address. As a consequence, machine induced knowledge-bases typically suffer shortcomings in aspects of human engineering, specifically explanation and dialogue structure.

EXPERIMENTS IN AUTOMATIC KNOWLEDGE ACQUISITION

The META-DENDRAL project (Buchanan et al, 1969 and Buchanan et al, 1976) is a well known example of the application of automatic knowledge acquisition techniques to the problem of constructing a knowledge-base. META-DENDRAL induces some of the rules that are used in DENDRAL (Buchanan and Feigenbaum, 1978), an expert system for molecular structure elucidation.

DENDRAL's domain is the interpretation of data produced by a mass spectrometer. In a mass spectrometer, a compound is bombarded by high energy electrons, causing fragmentations. The charged fragments of various mass are dispersed by a field into a spectrum. DENDRAL's job is to infer the molecular structure of the compound from the mass spectrum. To do this job, DENDRAL makes use of, amongst other forms of knowledge, knowledge of the way in which various structures fragment. META-DENDRAL induces rules of fragmentation.

The method used by META-DENDRAL consists of three stages, plan, generate and test. In the planning stage, distinctive peaks in the spectrum of a compound trigger the application of rules that specify likely fragmentations in the compound. The fragmentations are only grossly specified at this stage nevertheless classes of possible substructures will have been identified. In the generation stage, all the possible rules that could account for the "so far described" fragmentations are generated. This program limits its potentially enormous output by taking into account the likely substructures deduced from the planning stage and whatever constraints the user can supply. The planning stage is therefore essential in the operation of META-DENDRAL. In the test stage, the system tests each candidate rule against the spectra of samples with known structure. Many rules are deleted but others are refined or merged.

In considering the prospects for automating the job of constructing a knowledge-base, there are a number of important points to consider here. Firstly, META-DENDRAL makes use of some very specific chemistry knowledge and therefore the program cannot be used to construct knowledge-bases in any domain other than the mass spectrometry of a particular class of compounds. Feigenbaum (1977, p1021) argues that it is the approach used in META-DENDRAL that should be exported to other domains rather than the software.

> In a test of the generality of the approach, a version of the META-DENDRAL program is currently being applied to the discovery of the rules for the analysis of nuclear magnetic resonance data.

However, the nuclear magnetic resonance data referred to above is the output of a magnetic nuclear resonance spectrometer, a domain very similar to the interpretation of mass spectrometry data.

Secondly, META-DENDRAL can only induce fragmentation rules; DENDRAL however, makes use of other important sources of knowledge such as rules for interpreting data points of the spectrum, knowledge of stable and unstable configurations of atoms and an algorithm for generating all the possible "legal" molecular structures subject to various constraints. Consequently the META-DENDRAL "approach" cannot be used to entirely automate the construction of even "DENDRAL-like" knowledge-bases.

We must, if we aim to eliminate the need to modify a knowledge-base, consider the prospects for completely automating knowledge-base construction. In an important experiment, Michalski and Chilauski (1979) constructed an entire knowledge-base for an expert system for diagnosing soybean diseases. In this system, the AQ11 program (Michalski and Larson, 1978) was used to induce the rules for diagnosing soybean diseases from examples of diagnoses provided by

experts. The machine induced knowledge-base performed better than the rules derived from experts. Michalski and Chilauski (ibid, p263) conclude

> The comparison of 2 knowledge acquisition techniques indicates that decision rules derived inductively performed somewhat better than the rules derived by representing the knowledge of experts (in the specific context of soybean disease diagnosis). ...
>
> The major conclusion of this experiment is that the current computer induction techniques can already offer a viable knowledge acquisition method if the problem domain is sufficiently simple and well defined.

Notice the caution with which Michalski and Chilauski draw their conclusions. Computer induced rules were able to out perform the expert's 'in the specific context of soybean diagnosis'. Computer induction is a viable knowledge acquisition method 'if the problem domain is sufficiently simple and well defined'. Michalski and Chilauski are not suggesting that they have a technique which can entirely replace the "manual" development method for expert systems.

A popular application domain for the use of automatic knowledge acquisition techniques has been the game of chess. Quinlan (1979), for example, applies a rule induction program, (ID3), to a chess endgame problem. In this problem, there are four pieces left on the board, the two kings, a black knight and a white rook. The problem is to determine for any given board position whether the black knight's side is lost within two moves[4], given that it is black's move.

The method used by Quinlan involved chosing a number of attributes with which to describe various board positions; attributes such as 'distance from black king to the black knight', 'rook threatens knight' and so on. The class of board positions that are equivalent when described in terms of attribute values, is called a

---

[4] Two moves is one move by black and one move by white.

configuration. For the attributes to be adequate for the problem it is necessary that all the positions in one configuration are either all lost or all safe. There are eleven million or so positions divided amongst twenty nine thousand and two hundred and thirty six configurations. However, given this considerable reduction in the size of the problem, there are still too many configurations to present to the induction program.

Quinlan's way around this problem was to formulate, from a small training set of board configurations (four hundred of them), a rule (decision tree) for determining if black is lost or safe in two moves. This rule is then tested on every other possible board configuration; the answer (lost or safe) is known for each one. Whenever a rule fails, the training set is suitably modified and the procedure repeated. If such a procedure terminates it will of course have produced a correct rule.

The advantage of this method is that it is not necessary to "train on" all the twenty nine thousand or so configurations in order to achieve a correct rule, Quinlan (ibid, p178).

> We assume that the training set available to the experimenter is substantial -- it may even contain all possible instances. Such a training set will usually be too large to present to the induction algorithm, requiring either too much memory or excessive computation.

Quinlan (ibid, p185) speculates that his iterative method consumes resources at a linear, rather than exponential, rate.

> When the numbers of cpu seconds per million difficulty units required to solve these four problems are computed, the figures are 1.5, 1.8, 1.4, 1.4 respectively, which are remarkably consistent given the above range of difficulties. Consequently it would seem that no combinatorial explosion should prevent these same techniques being applied to larger problems.

Quinlan concludes, (ibid, p185)

> ... the main point of this article is the demonstration that techniques exist for discovering complex regularities in large collections of data. The 'large' can be justified by quoting the number of instances and attributes, but the 'complex' is less easy to pin down.

Indeed there is evidence to suggest that complex problems are not easily handled by these techniques. For example, one of the problems with the sort of induction program used by Quinlan and Michalski is that the machine induced decision rules are "flat". To elaborate, consider that the rules constitute a mapping from logical combinations (conjunctions and disjunctions) of attribute values to values of the single goal. For example, in the chess problem studied by Quinlan, the induction program produced the following two rules.

combination of attribute values	goal values
black knight can capture rook	black safe
black knight cannot capture rook and black knight can move to a square where it is not threatened and checks the white king	black safe

All the rules produced by the induction program conclude on a single goal, the safety or otherwise of black's position. Consequently, the induction program cannot induce relationships between attributes. For example, the following obvious relationship between attributes

```
if  knight is next to rook
then
    knight cannot capture rook
```

could not be induced. In looking for rules between values of attributes and values of the goal, the induction program ignores dependencies between attributes. This is not to say that the above rule could not be found by an induction program, it could, but it could not be found unless 'knight can or cannot capture rook' is the

specific goal under investigation in the application of the induction program. In other words, the problem structuring must be done by the knowledge engineer and cannot be induced by the induction program.

Given this inability of the sort of induction program we have been considering, it is not entirely clear what the practical consequences are in terms of the ability of the program to induce rules in complex domains. The notion of a 'complex regularity' in data, is to a large extent, an intuitive one. However, it ought, we claim, include the notion of intermediate relationships between primitive and non-primitive attributes. These latter attributes have values that depend on the values of other attributes. Furthermore, a well known strategy for attacking complex problems is to decompose the problem into judiciously chosen subproblems. The inability of the sort of induction programs used by Michalski et al and Quinlan to discover relationships between attributes means that they cannot discover suitable subproblems for a given problem. However, this simply means that the strategy of problem decomposition will not figure in machine induced rule-bases, and such knowledge-bases are disadvantaged to the extent that the problem decomposition strategy is essential in tackling complex tasks.

It should not be overlooked that the use of automatic knowledge acquisition technique require large amounts of good quality examples from which the required rules can be induced. In each case described above a large amount of good quality data was available. Michalski et al used two hundred and ninety cases in order to formulate rules to conclude on fifteen diseases. Quinlan had access to the entire set of possible configurations if need be. Each instance of DENDRAL's use potentially provides hundreds of data points that can be used by META-DENDRAL.

In many domains, it is very difficult to obtain such data and in some it may be impossible. Shortliffe (1976, p162,3) quotes Edwards (1972) in a discussion of the lack of suitable medical data for statistical analysis.

> ... My friends who are expert about medical records tell me that to attempt to dig out from even the most sophisticated hospital's records the frequency of association between any particular symptom and any particular diagnosis is next to impossible--and when I raise the question of complexes of symptoms, they stop speaking to me. For another thing, doctors keep telling me that diseases change, that this year's flu is different from last year's flu, so that symptom-disease records extending far back in time are of very limited usefulness. Moreover, the observation of symptoms is well-supplied with error, and the diagnosis of diseases is even more so; both kinds of errors will ordinarily be frozen permanently into symptom-disease statistics. Finally, even if diseases didn't change, doctors would. The usefulness of disease categories is so much a function of available treatments that these categories themselves change as treatments change--a fact hard to incorporate into symptom-disease statistics.

However, although induction programs do not place such stringent requirements on the quality of the data, induction programs nevertheless require large amounts of reasonably accurate data.

The expert could of course labouriously construct the data required by the induction program but in this case the "knowledge acquisition bottleneck" risks replacement by the "data construction bottleneck". Hence the requirement for good data, is another factor that argues against the complete subsumption of the "manual" method for knowledge-base construction.

TOOLS FOR AUTOMATIC KNOWLEDGE-BASE CONSTRUCTION

There are a number of knowledge acquisition tools that can be used to partly automate the construction of a knowledge-base. In practice, the user of such tools is not necessarily freed for the

problem of modifying the knowledge-base. TIMM[5], for example, is an expert system building tool that is able to suggest new rules from the rules provided by the expert. These rules are presented to the expert before insertion into the knowledge-base because they may need modification. In some cases, the rules already present in the rule-base may require modification before the new rule can be added. Consequently TIMM provides facilities to allow the knowledge engineer to make these modifications.

Ex-Tran 77[6] is another expert system building tool that is able to induce rules from examples. However, there are facilities with this tool for the knowledge engineer to modify the rules produced by the induction program. There is also a facility for the knowledge engineer to decompose a knowledge-base into modules. This facility is directed at overcoming the inability of induction programs (as discussed above) to automatically induce a decomposition of a problem.

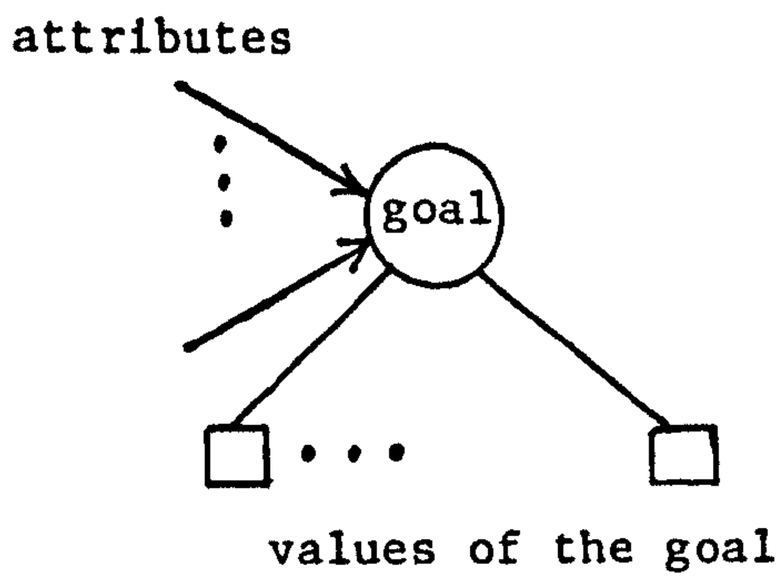
The simplest kind of module allowed in Ex-Tran 77 consists of a single goal with rules relating the values of various attributes to particular values of the goal. This is shown diagrammatically in fig 2.1.[7]

---

[5] TIMM is a product of the General Research Corporation.

[6] Ex-Tran 77 is a product of Intelligent Terminals Ltd.

[7] This and the next two figures are adapted from 'Ex-Tran 77, A technical Overview', 1984, a publication of Intelligent Terminals Ltd.



key: A circle indicates a goal or subgoal
A square indicates a final value of a goal or subgoal
'...' should be read as 'some number of'

fig 2.1

Primitive attributes conclude on a single goal

One way in which a problem can be structured is to use rules that relate the values of primitive attributes to the values of non-primitive attributes and rules relating values of non-primitive attributes to values of the goal. This is shown diagrammatically in

fig 2.2.

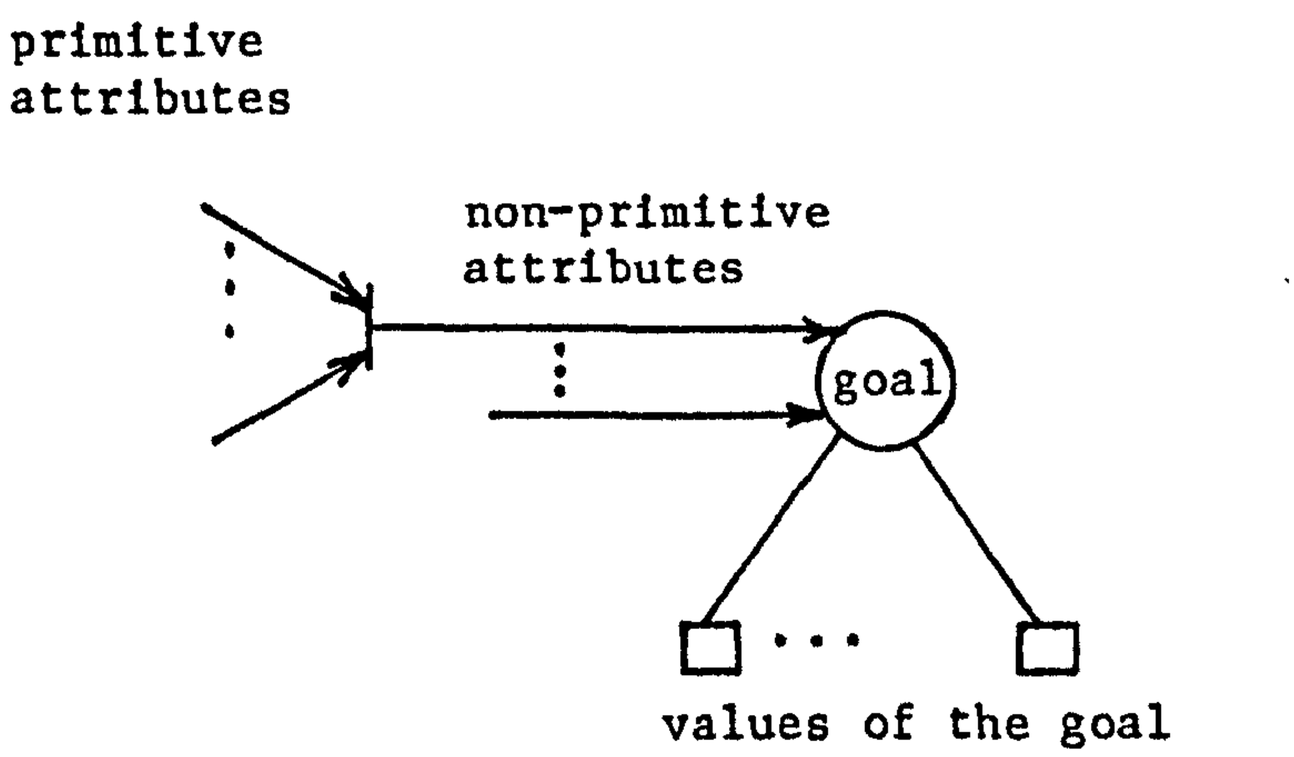


fig 2.2

Primitive attributes conclude on non-primitive attributes which conclude on a single goal

In addition, goals can be decomposed into subgoals as typically occurs in multi level decision trees (shown in fig 2.3).

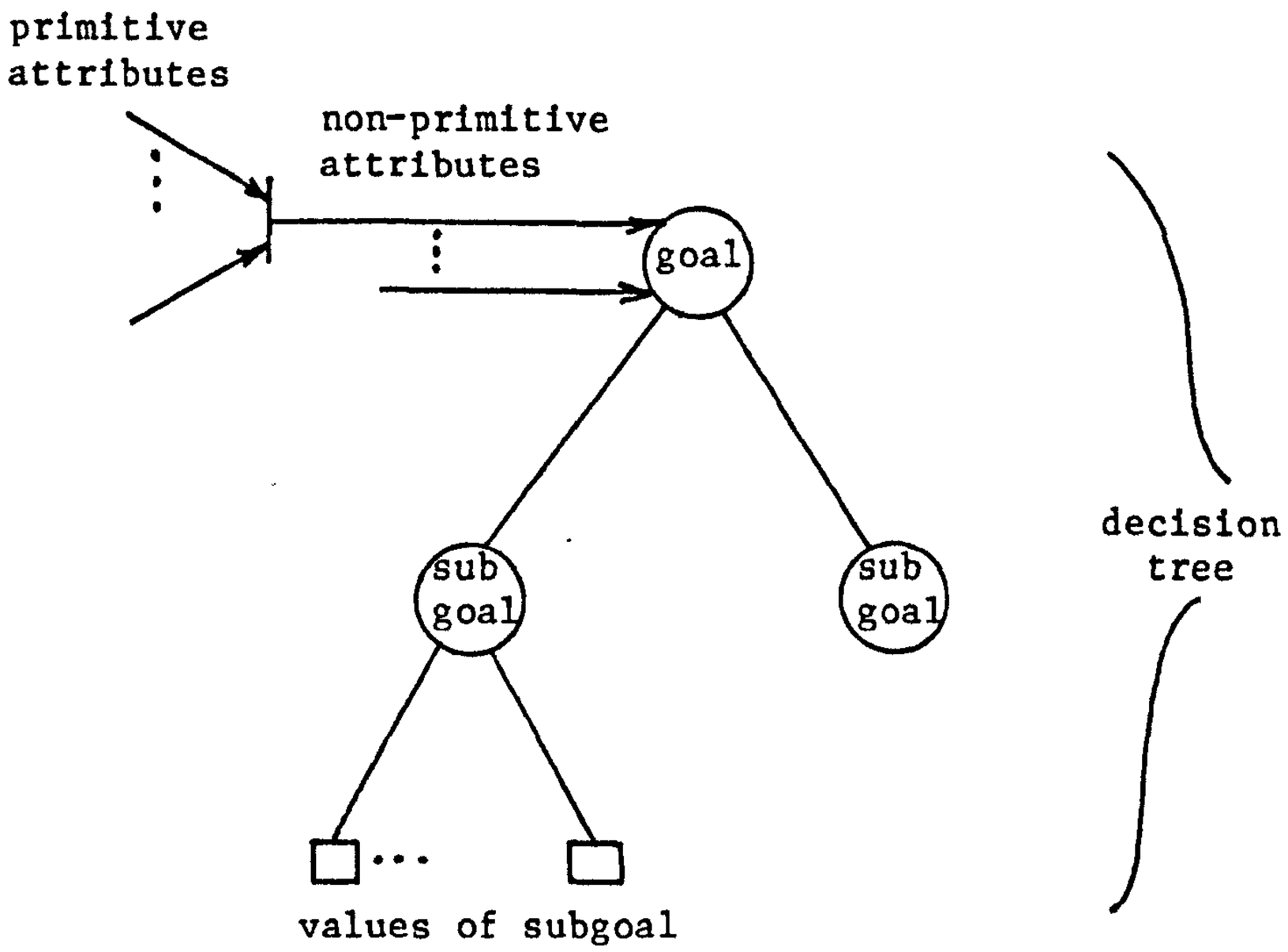


fig 2.3

Non-primitive attributes conclude on subgoals

ID3, the algorithm used by Quinlan, is the basis of the expert system building tool RuleMaster[8] (Michie et al, 1985). In this tool also allowance is made for the knowledge engineer to modify the rules produced by the tool. Once again, a facility exists for the knowledge engineer to manually decompose a problem into modules.

Since expert system building tools discussed above make allowance for the knowledge engineer to modify and structure the induced rules this suggests that such modification may sometimes be necessary.

---

[8] RuleMaster is a product of Intelligent Terminals Ltd. and Radian Corporation.

HUMAN ENGINEERING ASPECTS

Explanation of reasoning

It should not be forgotten that the machine induced knowledge is not human knowledge. Although this point may seem obvious enough, it has important implications. The human method of solving a problem has, over a period of time, evolved so that it satisfies various kinds of constraints. Typically, a problem solving method used by an expert must satisfy cognitive, social and economic constraints. In an expert system two important ways in which the need to obey these constraints manifests itself is in the need for explanations and an appropriate dialogue structure.

The need for explanations is recognised by those that advocate the automation of knowledge-base construction.

> Our focus on end-users in applications domains has forced attention to human engineering issues, in particular making the need for the explanation capability imperative.
>
> Feigenbaum (1977, p1027)

Michie's (1981) concept of the "human window" is particularly pertinent here.

> The mismatch between a technological system and the humans who operate it can be either at a "syntactic" or "semantic" level. A knowledge representation can be syntactically correctable. But if its semantic structure is wrong, then no amount of human technology can correct it.
>
> ibid, p1.

> Certification should only be granted to systems which demonstrably augment the user's understanding of his task environment. A clear distinction between "surface" (cosmetic) and "structural" (conceptual) causes of misunderstanding in an information system is needed. ... An analogy exhibits this last point: If a patient were to enter a doctor's office complaining of a boil on the thigh, lancing could be the indicated treatment. If the true problem were dislocation of the hip, surface treatment of any kind would be ineffective.
>
> ibid, p11.

Rule-based expert systems explain their reasoning by "replaying" the sequence of rules used in a line of reasoning. These rules offer an explanation only to the extent that they show relations between concepts that the user can understand. Furthermore, these relations must make "contact" with the user's present understanding of the problem solving methods of that domain if they are to be useful as explanations.

To understand why machine induced rules may be of little use for explanation it is necessary to understand something of how the rules are induced. The induction program finds a mapping between values of attributes and values of the goal. Typically, many mappings are possible. For example, figure 2.4 shows descriptions of three objects in terms of the attributes, colour, shape, size and texture.

attributes				goal
colour	shape	size	texture	material
blue	square	small	smooth	plastic
white	oval	medium	rough	wood
blue	octagon	large	smooth	plastic

fig 2.4

Attributes of plastic and wooden objects

An induction program that is to formulate a rule for discriminating between plastic and wooden objects may construct either the rule

```
if  the texture is smooth
then
    the object is made of plastic
else
if  the texture is rough
then
    the object is made of wood
```

or the rule

```
if  the colour is blue
then
   the object is made of plastic
else
if  the colour is white
then
    the object is made of wood
```

Given that in the general case, attribute values can be combined in many ways to achieve a correct rule, the induction program may well produce rules that are not suitable for use in explanations.

For instance, in the case where the rule used in a line of reasoning is a specific application of a more general but understood relation then the rule itself can serve as an explanation of that line of reasoning. Returning to the above example, if the user already understands that different materials have different textures then the rule that exploits the specific textures of plastic and wooden objects can be understood as an instance of the more general relation. The general relation is the point of "contact" between the users understanding and the reasoning that the system is seeking to explain. However, in the case where the underlying basis of the rule is not understood by the user the rule by itself explains nothing. Continuing the above example, if the user does not understand why the colour of an object should bear any relationship to its material then no explanation is provided by showing the user the rule that describes plastic objects as those that are blue and wooden ones as those that are white. In this latter case, additional knowledge is needed to to justify and make clear the underlying basis of the rule. For example, the rule might be justified by explaining that all objects are painted for the purpose of discrimination into types of material. This additional knowledge, necessary to justify a rule, cannot be induced by the induction program from examples of correct solutions to the

discrimination problem. The knowledge that justifies the rule does not bear on the discrimination problem and therefore lies outside the scope of the induction program. An expert however, may be able to justify any rules used and hence provide additional knowledge that could not be made available by the induction program.

Clancey (1983a) has described how in medical diagnosis problem solving, explanations depend on the presence of structural and strategic knowledge. Structural knowledge, such as the taxonomy of diseases and network of causal associations, allows the use of efficient strategies for diagnosis. Clancey (ibid) describes how in order to explain a method of reasoning it is necessary to show how it is an instance of a more general strategy. Rules produced by induction need not capture the way in which the facts and heuristics in a domain are structured to make problem solving efficient for humans. Consequently the absence of such knowledge in an induced rule-base would render the rule-base unsuitable for explanation.

Induction programs of the sort used by Michalski and Chilauski and Quinlan are not provided with any domain specific knowledge about understandable ways in which attribute values may be combined. Consequently the rules generated by these programs are arbitrary from the point of view of explanations.

Dialogue Structure

Much of what has been said about the quality of machine induced rules for explanation also applies to dialogue structure. The importance of the dialogue structure in expert systems is emphasised by Johnson (1985, p23).

> The central tenet of our research approach is that a human-computer system needs to be conceived, designed, analysed and evaluated in terms of a cognitive conversational context. The "object" to be designed is an interaction.

We have seen that the induction program produces a single rule out of the many that could typically be produced. One way in which the induction program can be "steered" towards particular rules is to assign costs to the use of various attributes in a rule. For example, if the cost of finding the value of the texture attribute is higher than that of finding the value of the colour attribute (because it is easier to look than to feel, say) then the program produces a rule relating colour to material. In a more sophisticated approach, it is possible to attach individual costs to the measurement of each attribute and to the misclassification of a particular class as some other particular class. For example, the cost of misclassifying wood as plastic may be higher than the cost of misclassifying plastic as wood. The decision tree can then be constructed so that the overall cost is minimised.

However, there is no a priori reason why such cost mechanisms should accurately model any of the underlying mechanisms responsible for the dialogue structure. For example, in the NEOMYCIN medical expert system (Clancey and Letsinger, 1981), the lower cost of collecting general information about the patient, age, sex, history etc., as opposed to the higher cost of performing laboratory tests means that the former is done before the latter. However, within these two broad categories, there may not be any important variation in "attribute evaluation cost" and so the ordering of questions must be done according to some other criteria. One such criteria used in NEOMYCIN is to group together all the questions about a single topic. It is perhaps a psychological/sociological fact that doctors prefer to provide data in this way. Our argument is that to simply assign costs to the evaluation of attributes and various misclassifications is to make a commitment to a particular model of the way the problem should

be solved. It may turn out that the model is inappropriate. In any case there can be no guarantee that the important factors underlying the dialogue structure have been taken into account without eliciting domain specific knowledge.

Johnson (ibid, p23) stresses the need to elicit the knowledge that determines a suitable dialogue structure.

> In the context of expert system this implies that a broad based knowledge elicitation which captures relevant aspects of the "user's conversational grammar" is an essential prerequisite to the design process (the notion of a grammar here is not literally meant but is meant to refer to deep structural characteristics of the conversational context).
>
> ... the conversational structure can only be accurately captured provided that the model of competence is accurately captured; the order in which items of information are being elicited from the non-expert by the expert as well as the order in which hypotheses are being pursued by the expert are inherent in the competence model.
>
> (ibid, p34)

It is our conclusion that although automatic knowledge acquisition programs can be a useful tool for the knowledge engineer, they cannot at present take over the entire job of knowledge-base construction. Consequently the knowledge engineer must to some extent iteratively develop and modify the knowledge-base. For this reason, the claimed advantages of the rule scheme for expert system development deserve serious investigation.

CHAPTER THREE

THE CLAIM THAT RULE REPRESENTATIONS OF KNOWLEDGE ARE MODIFIABLE

THE CLAIMS

Claims about the modifiability of rule representations of knowledge are not as straightforward as they might on first sight appear. We begin this chapter by listing a number of such claims, many of which assert that rule representations are easily modifiable but some assert the opposite! One of the difficulties found in examining these claims is that there are at least two distinct senses in which a scheme can be said to be modifiable. These two kinds of modifiability are distinguished. First however, we consider the claims themselves.

The DENDRAL system (Buchanan and Feigenbaum, 1978), is designed to provide assistance to chemists with structure elucidation problems in organic chemistry. As the first, pioneering, expert system project, the authors of the system are highly regarded by other workers in the field. Buchanan (1982, p278,279) describes the way in which the rule representation of knowledge was found to be useful in the construction of DENDRAL.

> Knowledge acquisition has become recognised as an issue with expert systems because it has turned out to be difficult and time consuming, DENDRAL, for example, was originally 'custom-crafted' over many years. ... We rewrote large parts of the systems as the knowledge base changed. After doing this a few times we began looking for ways to increase the rate of transfer of chemistry expertise from chemists into the program. Making procedures highly styalised and dependent on global parameters was a first step, but still required programmers to write new procedures. DENDRAL's knowledge of mass spectroscopy was finally codified in production rules.
>
> Once the vocabulary and syntax for the knowledge base are fixed, the process of knowledge acquisition can be speeded considerably by fitting (sometimes forcing) new knowledge into the framework.

The R1 system (McDermott, 1982) is able to configure the numerous components that are used to construct a Digital Equipment Corporation VAX computer. McDermott (ibid, p40) in describing R1, states

> I have tried to avoid letting the details of R1's inner working overshadow the domain independent lessons that have emerged from this research. ...
> When an expert system is implemented as a production system, the job of refining and extending the knowledge-base is quite easy.

The PUFF system (Kunz, et al, 1978) is designed to interpret pulmonary function test results. The system's knowledge consists of rules that relate pulmonary function test results to diagnoses of pulmonary disorders. The authors of the system report, (ibid, p13),

> Our experience was that the production rule formalism provided an extremely powerful technique for developing the pulmonary function interpretation system. ... The single-rule specification allowed for far more rapid and amicable system development than any that the participants had previously experienced.

The PROSPECTOR system (Duda et al, 1979) is intended to help a geologist in evaluating the likelihood of the presence of particular ore deposits at a site. In discussing the production rule representation scheme of the PROSPECTOR expert system Duda (Duda et al, 1978, p203,204) report

> The advantages of this approach stem from the fact that the representation is modular and declarative. This ... encourages incremental development, ...

The MYCIN system (Shortliffe, 1976) helps a physician diagnose the cause of a patient's infection and recommend a suitable therapy for that patient. MYCIN, another of the early pioneering expert system projects, has been particularly influential in subsequent work in this area. One of the reasons for MYCIN's widespread influence is Shortliffe's detailed and commendable description of the system (ibid). MYCIN is used as an example of an expert system in many

descriptions of the expert systems technology, so much so that it has almost become the archetypal expert system.

Shortliffe describes the modifiability advantages of the scheme in the MYCIN system (Shortliffe, 1976, p70) as follows:

> ... an inference model that depends on a complex decision tree is apt to be difficult to augment without a complete diagram of the tree so that all implications of additions can be observed. A modular system, on the other hand, permits knowledge to be acquired as isolated facts and allows the consultation program itself to decide under what conditions the new information is relevant.
>
> We accomplish modularity of system knowledge by storing all information in decision rules.
>
> ibid, p71.

> ... one of the major design considerations during the development of MYCIN has been the isolation of pieces of knowledge as discrete facts. MYCIN's decision rules achieve this goal. Since each rule represents a discrete packet of knowledge, the integration of new information into the system is simplified.
>
> ibid, p155.

One of the ways in which MYCIN has been most influential is in the development of the the EMYCIN "shell" (essentially MYCIN's rule interpreter) by van-Melle (1979). This "shell" has been used as the basis of several expert systems. van-Melle (1978), commenting on MYCIN's source of power, states (ibid, p313)

> Much of MYCIN's power derives from the modular, highly styalised nature of these decision rules, enabling the system to dissect its own reasoning and allowing easy modification to the knowledge-base.
>
> The modularity of the rules simplifies the task of updating the knowledge-base. Individual rules can be added, deleted or modified without drastically affecting the overall performance of the system.
>
> ibid, p317.

Davis and King (1977) discuss the use of the production rule representation scheme in the construction of expert systems.

> For the designer of knowledge-based systems, production rules offer a representation of knowledge that is relatively easily

accessed and modified, making it quite useful for systems designed for incremental approaches to competence.

This inherent modularity of pure production systems eases the task of programming in them. Given some primitive action that the system fails to perform, it becomes a matter of writing a rule whose LHS matches the relevant indicators in the data base, and whose RHS performs the action.

ibid, p306.

In an article on the general applicability of the expert system technology in commerce, Michaelson and Michie (1983, p240) state

> This type of programming has two major advantages over that used for DSSs [decision support systems]. In the first place, since the rules are independent of each other, program revision is much easier than in procedural FORTRAN or COBOL.

Rychener and Newell (1978, p137) describe the attractions of the OPS production system architecture.

> In practice, productions tend to be small (only a few conditions and actions) and relatively independent of each other. Thus they are attractive where structure is to be added gradually and incrementally.

The RITA (Anderson and Gillogly, 1976) rule-based expert system provides assistance to users in accessing large computer networks from their terminals. Waterman (1978, p278,9), describes the role of RITA's production rule encoding of knowledge in relation to an additional learning system. This system synthesizes RITA agents from traces of the network access activity that the new agent is to perform.

> The modularity of production systems is viewed as the critical factor underlying the successful approach taken to program creation. That is, the success obtained with RITA agents creating other RITA agents is attributable to the fact the programs being created, the new agents, are all based on the RITA production system architecture.

In discussing knowledge-based programs for signal understanding, Nii and Feigenbaum (1978, p501) state,

> The use of production rules to represent control/strategy

> knowledge offers the advantages of uniformity of representation and accessibility of knowledge for purposes of augmentation and modification of the knowledge base.

In a general article on rule-based systems Hayes-Roth (1985, p922) states

> Because each rule in a RBS[rule-based system] approximates an independent nugget of know-how, these systems have two characteristic features: First, existing knowledge can be refined, and new knowledge added for incremental increases in system performance.

Not all claims about the rule representation scheme have asserted that it is convenient to modify. Those involved in the DENDRAL and MYCIN projects also have the following to say about the modifiability of the rule scheme. Davis, Buchannan and Shortliffe (1977, p33) state

> Styalization and modularity also result in certain shortcomings, however. It is, of course somewhat harder to express a given piece of knowledge if it must be put into a predetermined format. ... It is not always easy to map a sequence of desired actions or tests into a set of production rules whose goal-directed invocation will provide that sequence.

More recently, Aikins (1983, p168,169) describes the shortcomings of the production rule scheme as follows.

> In rule-based systems the modularity of the rules prevents organization of the knowledge-base in a way that would identify groupings of similar rules and would be useful in making modifications to sets of rules or in identitying interactions between rules. Adding or modifying rules may have indirect effects on other rules that are difficult to predict without these explicit groupings.

The problems described above led Aikins to mix rule and frame representations of knowledge in building CENTAUR, a "second generation" system for the pulmonary function interpretation task preformed by PUFF.

Clancey (1983, p215) in discussing the representation of knowledge within MYCIN states

> ... people other than the original rule authors find it difficult to modify the rule set, ...

To overcome problems with MYCIN's rule representation of knowledge Clancey also uses a variety of representation schemes in building NEOMYCIN (Clancey and Letsinger, 1981) a "second generation" system for MYCIN's task domain.

WHAT EXACTLY IS BEING CLAIMED?

Scope of the claims

To examine these claims we need to be clear about what it is that is being asserted. Our first problem is in circumscribing the scope of the claims, i.e. in deciding what should qualify as a rule representation as opposed to some other kind of representation. It must not be assumed that there is a clear formally defined notion of what constitutes a rule representation of knowledge. Many descriptions of rule representations describe a rule as consisting of two parts, an "if" part that specifies a condition and a "then" part that specifies a conclusion or action to be performed. Without further constraints on what the "if" and "then" parts may consist of, virtually any representation can be construed as a rule representation. In practice, rule-base authors tend to adhere to the "spirit" of the notion of a rule and do not use overly involved constructions for the "if" and "then" parts of a rule.[9]

The rules of a particular rule-based expert system however, will, in general, conform to a well defined "rule language". This "rule language" is designed by the authors of that particular system. For

---

[9] Lenat (1978), however, argues for a number of departures from this notion of a rule for systems that do automatic scientific discovery. In particular he advocates the encoding of "complete" operations as functions and placing them in the "if" and "then" parts of rules as required.

example, Shortliffe (1976, p86,87) provides a BNF grammar for MYCIN's rule language, shown in fig 3.1.

```
<rule>::=<premise><action>|<premise><action><else>
<premise>::=($AND<condition>...<condition>)
<condition>::=(<funcl><context><parameter>)|
              (<func2><context><parameter><value>)|
              (<special-func><arguments>)|
              ($OR<condition>...<condition>)
<action>::=<concpart>
<else>::=<concpart>
<concpart>::=<conclusion>|<actfunc>|
             (DO-ALL<conclusion>...<conclusion>)|
             (DO-ALL<actfunc>...<actfunc>)
```

fig 3.1

The grammar for MYCIN's "rule language"

The "rule language" can be thought of as a special purpose programming language. In MYCIN's case, the knowledge engineer constructs a "program" (rule-base) by choosing appropriate contexts, parameters and values and combining them to form rules.

Although the rules of a single system will generally belong to a single "rule language", the "rule language" of one system may be quite different from the "rule language" of another. MYCIN's "rule language" for example, is very different from the that used in PROSPECTOR. PROSPECTOR's rule interpreter, for instance, allows forward chaining of rules as well as backward chaining (to which MYCIN is restricted). In practice this has a marked effect on the style of the "rule programming". PROSPECTOR's facility for taxonomic links between the components of its rules is another distinctive feature. Subclass relations are represented by these links rather than as rules. However, despite differences such as these, "rule languages" may, nonetheless, all be thought of as instances of the informal and more general concept of a rule representation scheme.

The concept of modifiability

In addition to the problem of circumscribing the scope of the claims, the concept of 'modifiability' provides us with a difficulty. There is an uninteresting sense in which all systems can be said to be modifiable in that, given sufficient effort, any modification can be performed to any system. In the extreme case the system may need to be entirely rebuilt! In order therefore to say something interesting about the modifiability of representations of knowledge it is necessary to distinguish between different degrees, or perhaps kinds, of modifiability.

It is a well known principle of computer science that high-level (problem oriented) languages are more convenient to use than low-level machine oriented languages. The reason for this is that high-level languages contain the sort of constructs that application programmers require.

Given that a "rule-language" can be regarded as a special purpose programming language, it is possible that "rule-languages" can act as high-level languages for the representation of expert knowledge. If this is so then the use of the rule representation scheme would reduce the effort required to build a knowledge-base. The claims made by some of the authors quoted above, i.e. (Buchanan and Feigenbaum, ibid), (McDermott, ibid) and (Kunz, et al, ibid), are consistent with this view of the rule representation scheme. Kunz et al (ibid) for example, simply claims that the rule scheme provides

'... far more rapid and amicable system development ...'

Hence, it is possible that these authors are asserting that the rule scheme provides expert system builders with the sort of advantages that programmers enjoy from the use of high-level programming

languages.

However, not all the claims listed above can be interpreted in this way. For example, consider how Shortliffe (ibid, p70) describes the way in which some reasoning programs are difficult to modify,

> ... an inference model that depends on a complex decision tree is apt to be difficult to augment without a complete diagram of the tree so that all implications of additions can be observed.

A program that does not suffer from this problem is one that (ibid, p70)

> ... permits knowledge to be acquired as isolated facts and allows the consultation program itself to decide under what conditions the new information is relevant.

A number of the other claims shown above (i.e. Michaelson and Michie, Rychener and Newell and Hayes-Roth) refer to the so called "independence" of the rules in a knowledge-base as a justification for the modifiability of the knowledge-base.

The kind of modifiability that is being described in this last quote lies at the heart of our argument in this thesis and is worth making clear with an example.

The example representation of knowledge is a production system program which is to be regarded as an expert system in the domain of motoring navigation. More specifically, the system solves a "toy" problem of choosing the shortest route between two towns. Anyone with access to and the ability to read a map will, of course, be expert in this domain. But, for the purpose of this example, assume that only an expert is able to provide routes between towns and that anyone who makes a new journey must consult an expert for a route.

A knowledge engineer has been given the job of constructing an expert system to provide this navigational service. The knowledge

engineer starts by representing the expert's knowledge about particular routes. Each route is represented as a rule such as

AD ---> AB BC CD

which represents the fact that to travel from town A to town D one should travel from towns A to B to C and then to D[10]. The application of the rules is controlled by a forward chaining interpreter. This interpreter matches the starting town and destination against the premises of each rule until a match is obtained. The conclusion of the matched rule is the required route. Clearly, in this way knowledge about any number of routes can be represented within the rule-base.[11]

Here we have a representation of knowledge that allows knowledge of new routes to be added to the rule-base without the need to examine and perhaps modify the rules that are already present. If we wish to add the knowledge that the route from A to F passes through B then it is simply a matter of adding the rule

AF ---> AB BF

to the rule-base. There is no need to consider any other rules that might be present in the rule-base.[12] Similarly routes can be independently deleted or modified.

---

[10] There is no assumption that a road can be traversed in both directions.

[11] This production system, which we call NAVIGATE-1, requires a large amount of memory but does little calculation. We use a number of "oddly" designed programs as examples in this thesis in order to make specific points. There is no suggestion that any of the problems used as examples in thesis should be tackled seriously in the way shown.

[12] The assumption is that the expert does not provide inconsistent knowledge.

The special kind of modifiability shown in this example we will call extensibility. Hence, we would say that the addition of a new route is an extension to the knowledge-base. The essential property of an extension is that the new knowledge can be incorporated directly into the knowledge-base without eliciting any other knowledge from the expert. An extension can be thought of as a modification that simply "slots into place" in the knowledge-base and is ready to use. In the following chapters we give examples of modifications that are not extensions and so the notion of an extension will be made clearer.

We have distinguished two ways in which claims about the modifiability of the rule scheme can be understood. In the first, claims about the modifiability of the rule scheme can be understood as asserting that the rule scheme is able to act as a convenient "high level language" for the representation of knowledge. In the second, the claims can be understood as asserting that rule representations of knowledge can be extended in the sense just described.

The question of whether the rule scheme is indeed able to act as a "high-level language" for the representation of knowledge is largely an empirical question. Particular rule representations could be examined to assess how easily and naturally the knowledge is represented as rules in comparison to some other representation scheme. Two rule representations, in PUFF and MYCIN, have been examined in this way and the results are reported in chapters seven and eight. To summarise, it was found that the rule scheme did not provide a natural and explicit representation of the knowledge in each system.

The second sense in which the claims can be understood is that the rule representation allows knowledge to be acquired as 'isolated facts' and added to the knowledge-base as extensions. In this thesis

we concentrate on investigating this sense of the claim because it is the stronger and therefore more interesting sense. To appreciate the strength of this claim notice that a knowledge-base that can be extended is particularly convenient to develop incrementally since knowledge elicited from the expert can be simply "thrown into the pot". The knowledge engineer does not need to make any (possibly time consuming) changes to the existing representation in order to represent the new knowledge.

In comparison with the case where the modification cannot be carried out as an extension, there is no guarantee that the inclusion of the knowledge is trivial. Indeed, as is shown in the next chapter, it can be arbitrarily difficult to add the knowledge elicited from the expert. The difficulty is present because the knowledge cannot be included in the knowledge-base without eliciting further knowledge from the expert.

CHAPTER FOUR

THE EXTENSIBILITY OF RULE REPRESENTATIONS

INTRODUCTION

Given the concept of an extension to a representation of knowledge, which appears to be the kind of modifiability ideally required for the incremental development of a knowledge-base, we now investigate whether rule representations of knowledge are modifiable in this way or not. We show that a rule representation of knowledge is not necessarily extensible. We go on to show that those rule representations that are extensible are so because they embody a particular organisation of knowledge.

We convey our argument by discussing two extended examples. However, these examples are only used to convey general points; points not dependent on any special characteristics of the chosen examples. For each example, the discussion proceeds according to the following plan. Given a rule representation of knowledge we choose a particular modification to that representation and demonstrate it cannot be performed as an extension. We then change the organisation of knowledge to produce a new rule representation and demonstrate that in this representation the modification can be performed as an extension. Hence we conclude that the extensibility of the representation is due to its organisation of knowledge rather than to its rule form.

THE FIRST EXAMPLE: NAVIGATE-1 AND NAVIGATE-2

Modifications that are not extensions in NAVIGATE-1

Returning to the example of a navigational expert system; since we showed its rule-base to be extensible it would appear that NAVIGATE-1 is evidence in favour of the claim that rule

representations of knowledge are idealy suited for incremental development. However, it should not be forgotten that we have only demonstrated the extensibility of NAVIGATE-1 in terms of knowledge of new routes. If we wish to add navigational knowledge that does not consist of a route between two towns then it may not be possible to do this as an extension.

For example, suppose that after a period of development a large number of rules have been accumulated in NAVIGATE-1 and the system is performing well. Now suppose that a new road is built between two towns where previously there had not been a road. Expert navigators are quickly able to incorporate this fact and to continue to provide the shortest routes. The knowledge engineer, however, has a problem in incorporating this new fact into NAVIGATE-1 since many routes and hence many rules may be affected.

To appreciate this problem consider the arrangement of towns and roads shown in fig 4.1.

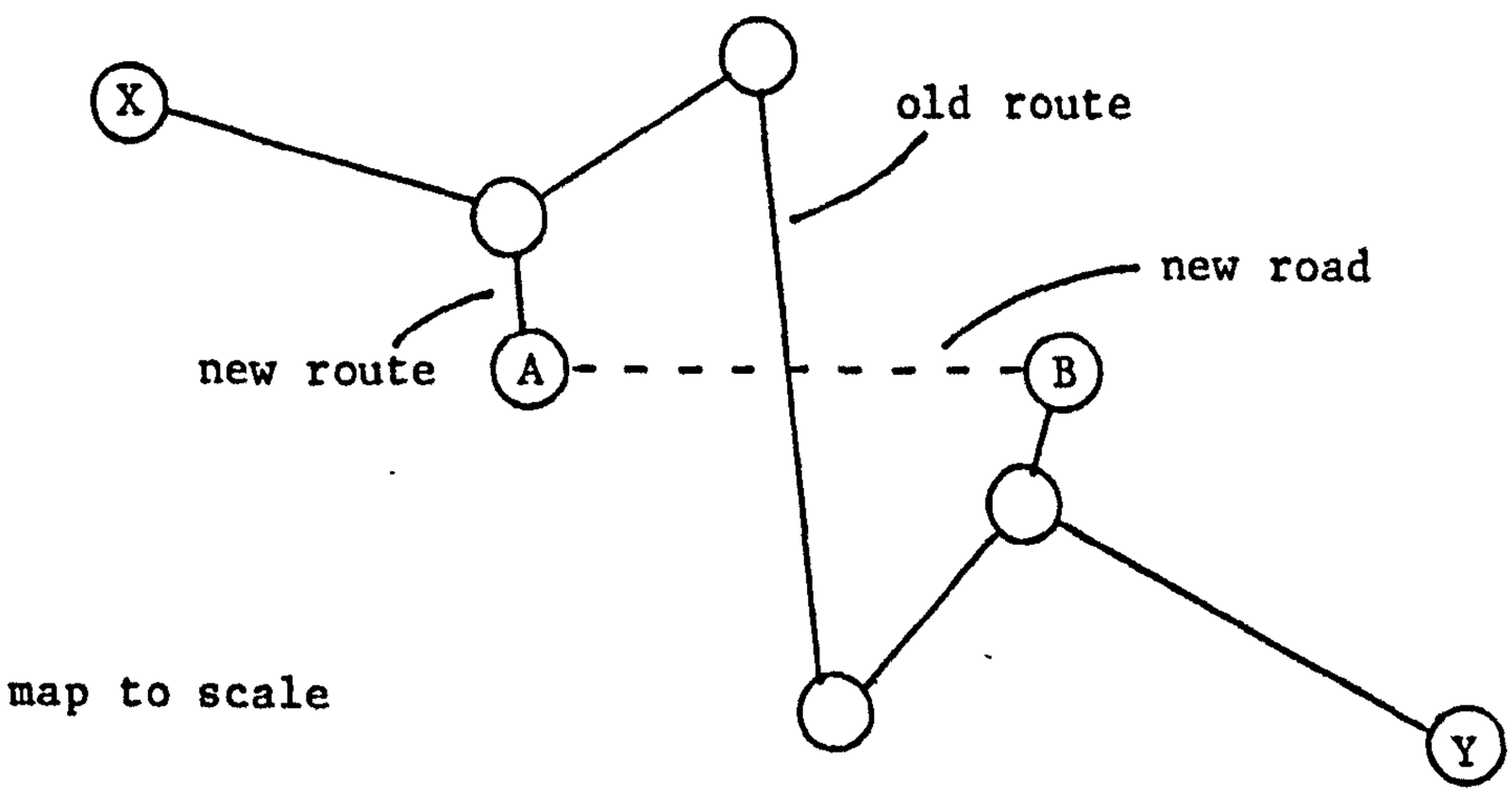


fig 4.1

An inconvenient modification to NAVIGATE-1

At present the route from X to Y passes neither through A nor through B. Nevertheless the new road from A to B will shorten the present route from X to Y and so the rule that encodes the route

between these two towns requires modification. However the knowledge engineer is only provided with the fact that a new road connects A and B, and since neither of the two towns A or B lies on the present route from X to Y, the knowledge engineer has no way of telling that the route from X to Y requires modification.

The knowledge engineer has been supplied with a new fact in the navigational domain but the knowledge engineer cannot incorporate this fact in the present representation. The new fact cannot be incorporated without asking for more information from the expert. In particular the knowledge engineer must elicit additional knowledge about the routes that are affected by the new road. It is this additional work of eliciting and representing knowledge about the affected routes that would have been avoided if the knowledge of a new road could have been added as an extension. This example illustrates how an extension is a very convenient modification and why extensible knowledge-bases are idealy suited for incremental development.

NAVIGATE-2: a more modifiable system

We may regard NAVIGATE-1 as a prototype that was constructed in order to learn something about the domain of navigation. The representation of navigational knowledge in NAVIGATE-2 is more complex than that of NAVIGATE-1 and is therefore perhaps best approached by considering how a knowledge engineer might go about discovering it. Recall that as yet there is no computational theory of navigation, it is the knowledge engineer's job to elicit such a theory.

Suppose the problem of modifying NAVIGATE-1 to take account of the new road leads the knowledge engineer to notice certain regularities in the rules of the system. For example the knowledge engineer notices that in the RHS (right hand side) of a rule the names

of towns occur in pairs. Within any pair the first town of the pair is invariably followed by one of a small set of towns. In the rules shown below, B is followed by A, C or F but never any other town.

AD ---> AB BC CD

AE ---> AB BC CE

AF ---> AB BF

DA ---> DC CB BA

The regularity seems important to the knowledge engineer because the building of a new road from B to E requires that this regularity be violated since the rule-base should at least contain the rule

BE ---> BE

and now B may be followed by E, in addition to A, C or F. The knowledge engineer consults with the expert and discovers that each town has a particular set of roads that lead away from it. In other words, including the new road, there are now four roads that join B to the towns A, C, E and F. The knowledge engineer constructs a diagrammatic representation of this concept, shown below in fig 4.2.

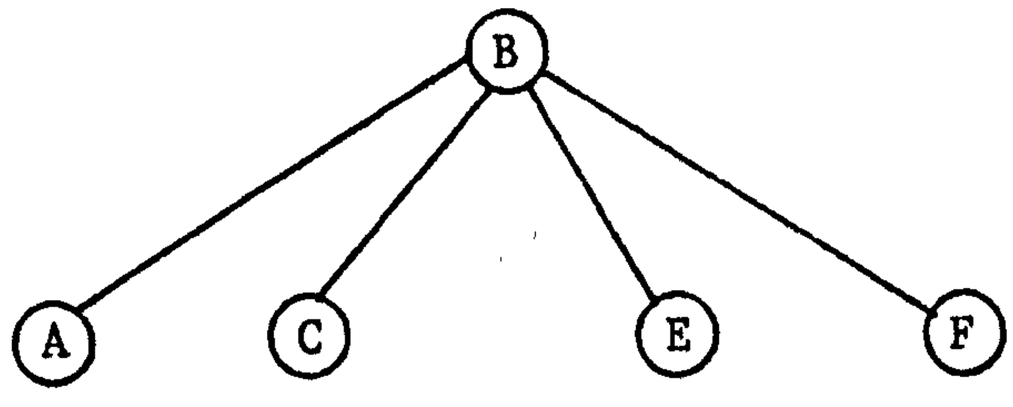


fig 4.2

The towns joined directly to B

In addition to discovering the group of towns that are immediately joined to a given town, the knowledge engineer determines that, in any route, it is always possible to determine which town should actually follow a given town. Of all the towns that are joined to a particular town, the selection of the town that is to be part of

the route is determined by the destination. For example, in the rules shown above, B is followed by C whenever the destination town is D or E, followed by F only when F is the destination town and followed by A when A is the destination town. The expert explains to the knowledge engineer that the choice of road by which to leave a given town is determined by the direction of the road and the overall direction of travel from the given town to the destination town. The knowledge engineer represents this selection operation diagrammatically as shown below.

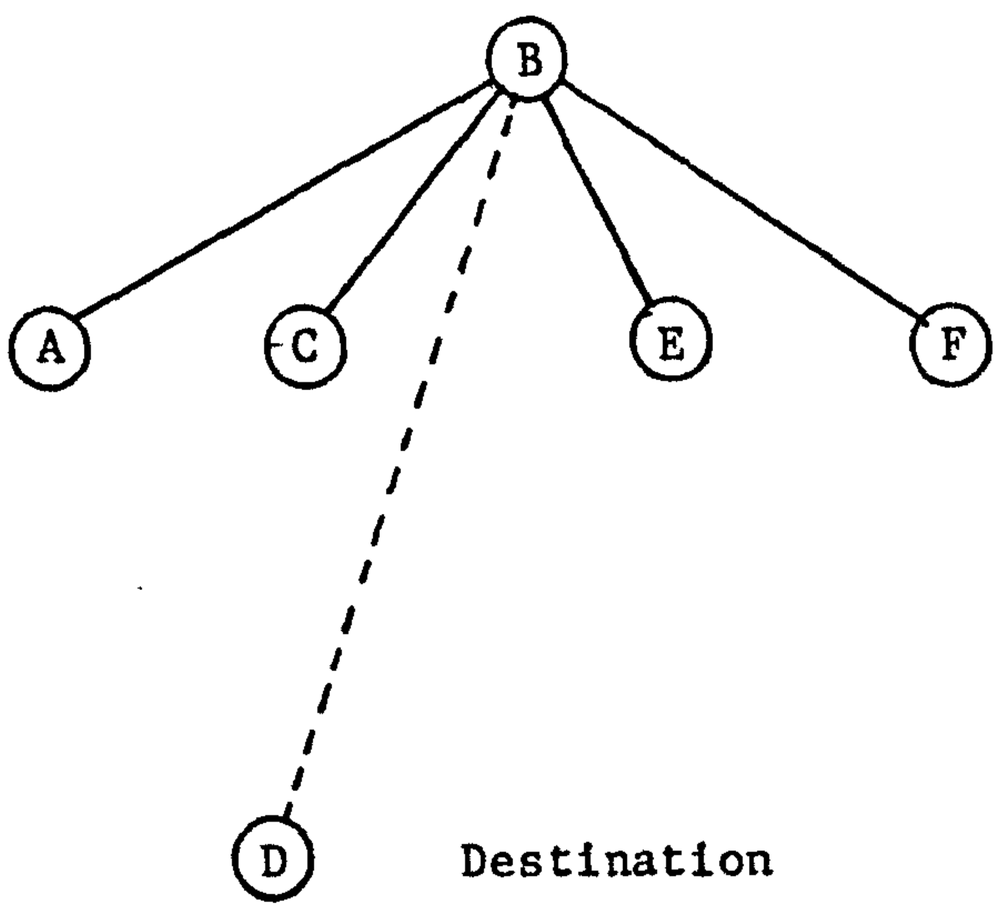


fig 4.3

BC is selected for the destination D
because the direction of BC is closest to that of BD

For any pair of towns the expert can provide the direction from one town to the other.

By this process of analysis and knowledge elicitation, the knowledge engineer conjectures that NAVIGATE-2 should contain rules that represent knowledge about which towns are immediately connected to other towns. Rules are also required that represent the direction between towns and a rule for selecting among the roads leading out of a town on the basis of their directions relative to the direction of

the destination. We now describe NAVIGATE-2 in detail.

NAVIGATE-2: the details

The majority of the knowledge represented in NAVIGATE-2 is represented in two kinds of rule. The first kind of rule is called a 'find-roads' rule. For each town there corresponds a 'find-roads' rule. The 'find-roads' rule for a particular town encodes the roads[13] that are joined to that town. For example the 'find-roads' rule for the town B is

```
R1 if  the context is find-roads
   and the current-town is B
   then
        the following roads, BA BC BF are connected
        to the current-town and the context is
        find-direction-roads.
```

because to leave the town B one must take one of the roads BA, BC or BF.[14] Notice that the above rule may only be applied if the value of the 'context' variable is 'find-roads' and that when the rule is applied the value of the 'context' variable is changed to 'find-direction-roads'.[15]

The second kind of rule is a 'direction' rule. For each pair of towns the system contains a 'direction' rule. The 'direction' rule corresponding to BC is

```
R2 if  the context is find-direction
   and the first town is B
   and the second town is C
   then
        the direction from the first to the second town
        is 60 degrees and the context is found-direction.
```

---

[13] More accurately the rights of way, since we assume roads to be "one way" unless otherwise stated.

[14] Roads are represented by the towns at either end.

[15] Readers familiar with the R1 expert system will notice a similarity between the use of context variables in R1 and the use of the context variable in NAVIGATE-2.

Before examining the rest of the rules in NAVIGATE-2 it will be helpful in understanding them if we overview their operation. NAVIGATE-2 generates a route by performing a number of iterations (one for each stage of the route) of the following procedure.

1) Use the 'find-roads' rules to find all the roads that are joined to the current town.

2) Use the 'find-direction' rules to find the direction of each of the roads found in 1) above.

3) By comparing the direction of the roads joined to the current town to the overall direction to the destination, choose the best road.[16]

4) The town at the far end of the best road becomes the new current town and the procedure returns to stage 1) above.

In addition to the 'find-roads' and direction rules there are also a number of rules representing knowledge of how the 'find-roads' and direction rules should be used to find a route. For example, the following rule is used to find the directions of the roads connected to the current town.

```
R3 if  the context is find-direction-roads
   and there is a road connected to the
       current-town with no associated direction
   then
       let the first-town be the current-town
       let the second-town be the town at the far end
       of the road and context is find-direction.
```

Once the direction between two towns has been found, (using a direction rule) three possibilities present themselves. Firstly, there may be roads outstanding with no associated direction, hence the rule

---

[16] We assume that the geography of the roads is such that this simple procedure is not mislead.

```
R4-1 if  the context is found-direction
     and there is a road connected to the
         current-town with no associated direction
     then
         the context is find-direction-roads.
```

Secondly, all roads may be associated with a direction but the direction of the current town to the destination has yet to be found, hence the rule

```
R4-2 if  the context is found-direction
     and all roads connected to the current-town
         have an associated direction
     then
         the context is find-destination-direction.
```

Thirdly, once the direction to the destination has been found it is time to select the best road for the route, hence the rule

```
R4-3 if  the context is found-direction
     and the direction to the destination is known
     then
         the context is find-best-road.
```

The following rule allows the current direction of travel to be found, i.e. the direction from the current town to the destination.

```
R5 if  the context is find-destination-direction
   then
        let the first-town be the current-town
        let the second-town be the destination-town
        and context is find-direction.
```

The following rule compares the current direction of travel with the directions of a number of roads and selects the most appropriate road.

```
R6 if  the context is find-best-road
   and there is a road connected to the current-town
   and of all the roads connected to the current-town
       the direction of this road is closest to the
       direction to the destination
   then
       add this road to the route
       let the current-town be the town at the far end
       of the selected road and context is check-route-complete.
```

The route is complete when the current-town is the destination town, hence the rules

```
R7 if  the context is check-route-complete
   and the current-town is the destination-town
   then
       output route and halt.


R8 if  the context is check-route-complete
   and the current-town is not the destination-town
   then
       context is find-roads.
```

The rule interpreter forward chains these rules, in other words, repeatedly attempts to match the LHS of rules with the contents of working memory or data base. The use of contexts ensures that only the appropriate rules are successfully matched in any one cycle of the control strategy. NAVIGATE-2 has been implemented as a program in the YAPS[17] production rule language and appears as appendix C.

The operation of NAVIGATE-2 is best described by an example. The reader may use the following map of roads and towns as an aid in following the operation of NAVIGATE-2.

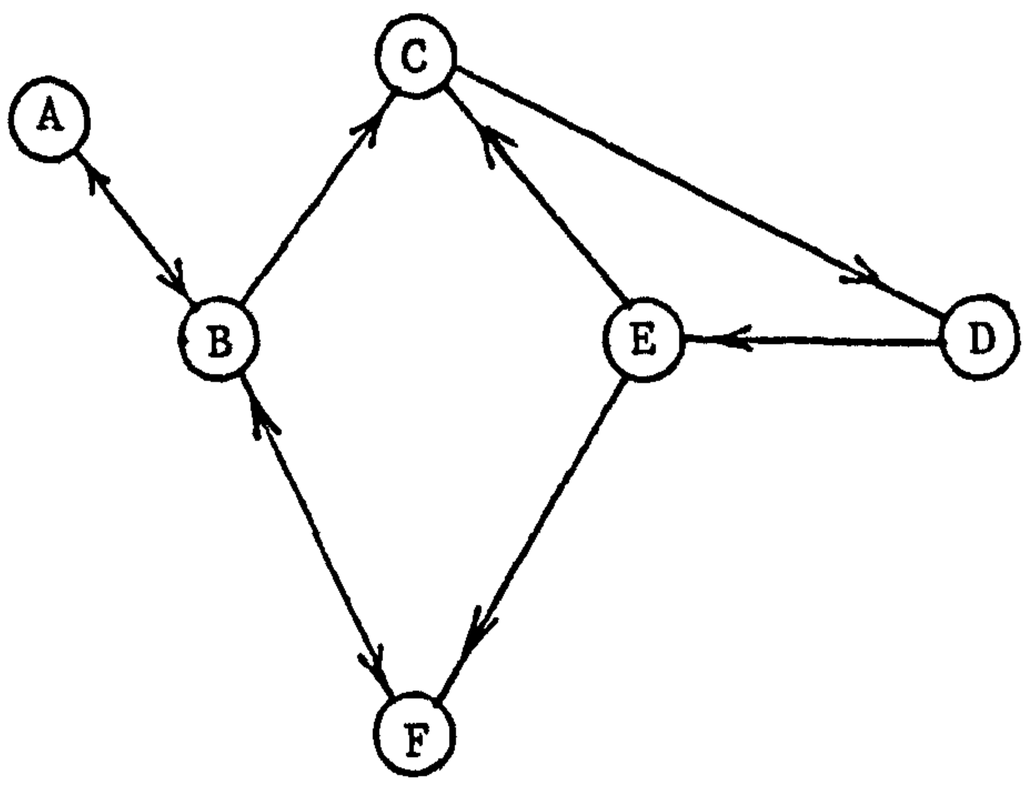


fig 4.4

The roads and towns in NAVIGATE-2's domain

The following figure 4.5 shows the general flow of control from 'context' to 'context'. Initially the system is in the 'find-roads'

---

[17] Very similar to the OPS-5 production rule language.

'context'.

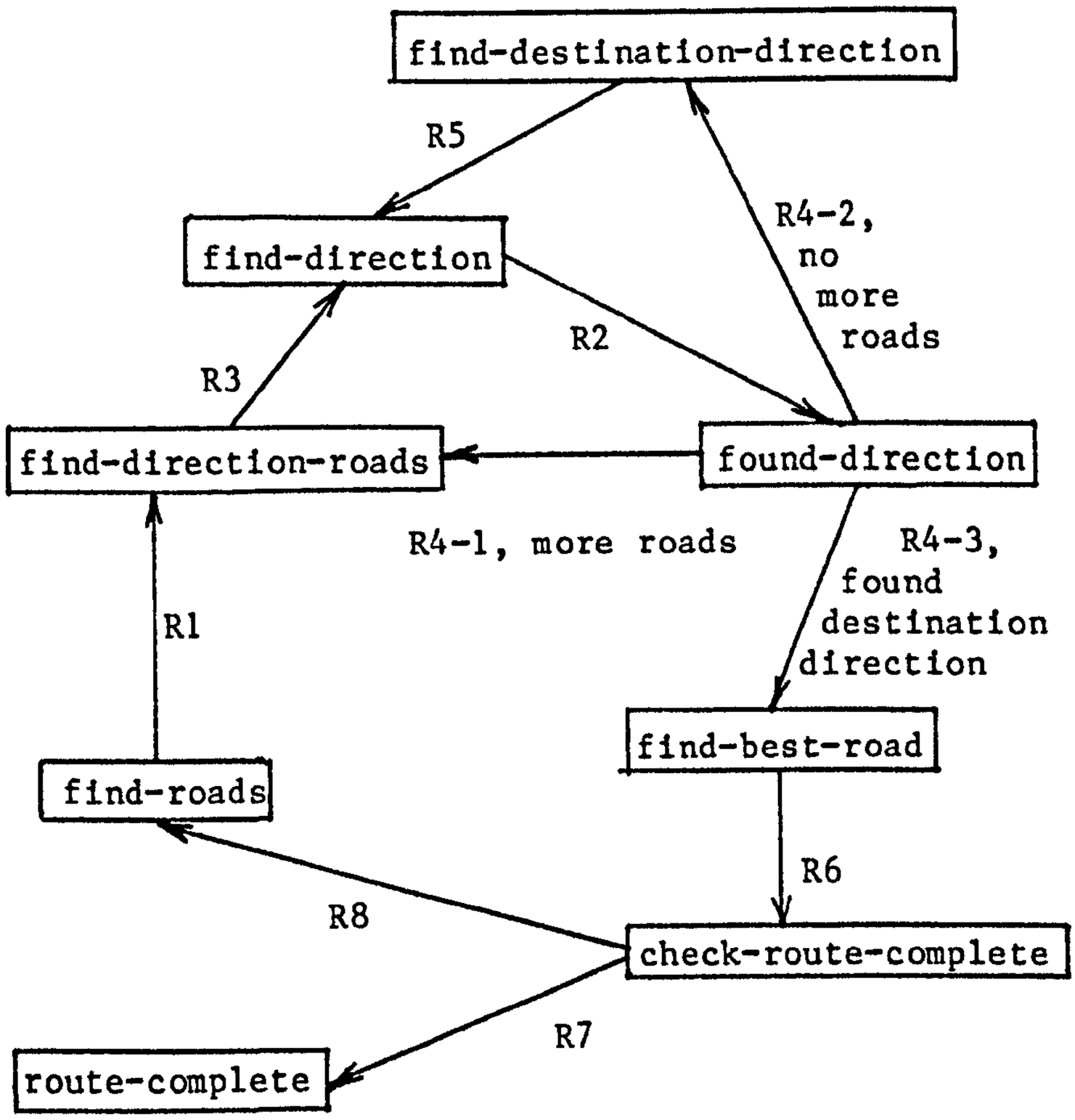


fig 4.5

The flow of control in NAVIGATE-2

The following figure 4.6 shows an extract of the operation of NAVIGATE-2 as it finds a route from A to D, the 'current-town' at the start of the extract is B and the route calculated so far is AB.

Context	Data Base Contents
find-roads	current-town is B destination is D
R1 find-direction-roads	" current-town-joined to A current-town-joined to C current-town-joined to F
R3 find-direction	" first-town is B second-town is C
R2 found-direction	" direction-to-C is 60
R4-1 • • R4-2 find-destination-direction	
R5 find-direction	" first-town is B second-town is D
R2 • • R4-3 find-best-road	
R6 check-route-complete	" road-on-route is BC current-town is C
R8 find-roads	

fig 4.6

An extract of the operation of NAVIGATE-2

NAVIGATE-2 can provide any route that NAVIGATE-1 can provide. However, the development of NAVIGATE-2 requires the acquisition of a different kind of knowledge to that required in the development of NAVIGATE-1. Instead of eliciting particular routes, the knowledge engineer asks the expert about the towns that are immediately connected by road to a given town, and thereby elicits a 'find-roads' rule. This 'find-roads' rule can be added to the rule-base without having to examine, and possible modify, any of the other rules that are present there, In other words, knowledge about the towns

immediately connected to a given town can be added as an extension to the rule-base[18]. In a similar way new 'find-direction' rules can be freely added to the rule-base as extensions.

NAVIGATE-2: new road modification

Now that NAVIGATE-2 has been described in detail we can return to the problem that motivated the construction of the new system, namely the problem of adding knowledge of a new road. Recall that knowledge of a new road could not be added as an extension to NAVIGATE-1 since many of the rules already present in the rule-base would require modification. In NAVIGATE-2, however, this knowledge can be added as an extension. As an example consider the modification required to NAVIGATE-2 to take account of a new road from B to E. The new road joins the town B to E; therefore, the 'find-roads' rule corresponding to the town B must be modified to include the new road. The necessary modification is shown below.

```
R1 if  the context is find-roads
   and the current-town is B
   then
       the following roads, BA BC BF are connected to the
       current-town and the context is find-direction-roads.
```

becomes

```
R1' if  the context is find-roads
    and the current-town is B
    then
        the following roads, BA BC BF BE are connected to the
        current-town and the context is find-direction-roads.
```

The 'find-roads' rule modification shown above can be carried out without asking the expert for any knowledge in addition to the towns at each end of the new road, hence the modification is an extension.

---

[18] By adding rules in this way the knowledge engineer incurs the risk that the rule-base may contain inconsistent rules but this is a different issue from the modifiability of the rule-base. Assume therefore that the expert never provides inconsistent rules.

Both NAVIGATE-1 and NAVIGATE-2 use rule representations of knowledge and yet as far as knowledge of new roads is concerned only NAVIGATE-2 is extensible. The rule representations of knowledge in NAVIGATE-1 and NAVIGATE-2 however, differ in content. The knowledge used by NAVIGATE-1 to find routes is not the same as the knowledge used by NAVIGATE-2, and it is to this difference that we look to in explaining the difference in modifiability between the two representations. Before proceeding with this explanation we point out that in thesis we use a rather particular conception of knowledge, distinguishing it from other conceptions by calling it an organisation of knowledge.

An organisation of knowledge is a collection of domain facts including relations between facts and a method for using these facts to solve particular problems. We call the knowledge an organisation of knowledge because it is knowledge organised for a purpose, i.e for use in an expert system to perform a particular reasoning task.

Given a particular reasoning task, it is usually possible to have a number of different organisations of knowledge all of them capable (if we neglect explanations and modifiability requirements) of being used in an expert system to perform that task. More generally, the distinction between "compiled" or "surface" knowledge approaches and "deep" knowledge approaches (Chandrasekaran and Mittal, 1982) (Hart, 1982) (Michie, 1982) is a distinction between two general categories of organisation of knowledge.

Given the concept of an organisation of knowledge we now describe the organisations of knowledge in NAVIGATE-1 and NAVIGATE-2 and show how the differences in modifiability between the two systems are attributable to differences between these two organisations of knowledge.

The navigational knowledge in NAVIGATE-1 is organised as an unstructured collection of routes. The collection is unstructured in that there are no constraints, save consistency which is assumed, among individual routes. In other words any subset of all the possible motoring routes will constitute a valid collection of routes for NAVIGATE-1's rule-base. The sort of independence between routes being described here is essentially the sort that exists among the words of an English language dictionary. Individual entries may be added or deleted from a dictionary without the need to amend other entries. For the same reason knowledge of routes can be added as extensions to NAVIGATE-1, i.e. in the system's organisation of knowledge, routes are (in the sense described above) independent of each other.

Consider now the organisation of knowledge in NAVIGATE-2 and the way it allows new knowledge about roads to be conveniently added to it. In this organisation of knowledge, it is the individual collections of roads that leave particular towns that are independent of each other. From the road map shown below it can be seen that only the collection of roads that leave B is changed by the addition of the new road from B to E.

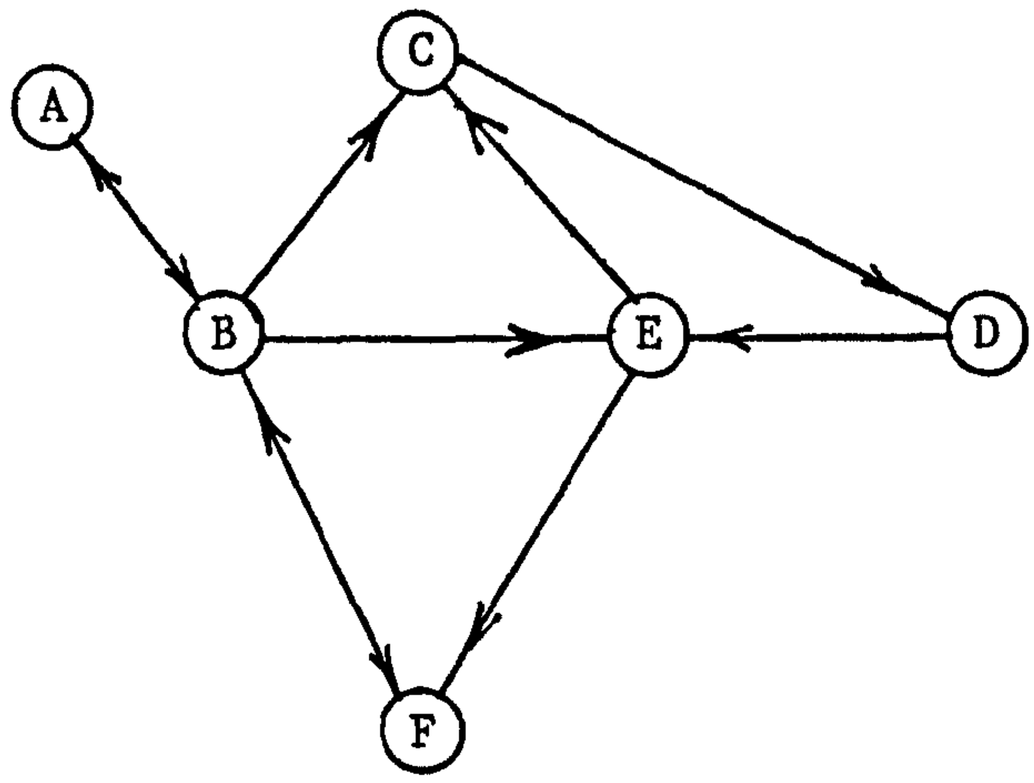


fig 4.7

The new road from B to E

Since the collections of roads that leave every town other than B are unaffected by the new road from B to E there is no need to amend any of the `find-roads' rules that represent this knowledge. Consequently knowledge of a new road can be added as an extension to NAVIGATE-2

It is important to notice that in each of the two previous paragraphs in which we explained the extensibility of NAVIGATE-1 and NAVIGATE-2 there is no mention of any properties of the rule representation scheme. The independence of routes in NAVIGATE-1, is not explained by their representation as individual rules. Similarly, in NAVIGATE-2 the independence among collections of roads leaving a town is not explained by their representation as individual rules. We account for the extensibility of NAVIGATE-1 and NAVIGATE-2 entirely in terms of their organisations of knowledge.

THE SECOND EXAMPLE: MICRO-ORGANISMS-1 AND MICRO-ORGANISMS-2

The argument in the first section of this chapter was illustrated by two simple production system examples in which production rules are forward chained. The simple "micro-world" problem of finding routes, although not a realistic problem for an expert system, allowed us to

describe representations of knowledge in detail without the descriptions overshadowing and obscuring the point of the examples. However, it should not be thought that the argument of the first section of this chapter applies only to production system representations of knowledge from "micro-world" domains.

We now consider another example of how the extensibility of a rule representation is determined by the organisation of the knowledge represented. This example is from the medical domain of infectious disease diagnosis and the knowledge is represented in "MYCIN-like" rules which are backward chained. Again we present two rule representations of knowledge, only one of which allows knowledge of a certain kind to be added as an extension. As before, we argue that the difference in modifiability between the two representations is attributable to differences in the organisation of knowledge in each representation.

MICRO-ORGANISMS-1

The organisation of knowledge in this rule-base consists essentially of two sorts of "rule-like" associations between three types of domain knowledge as shown in fig 4.8.

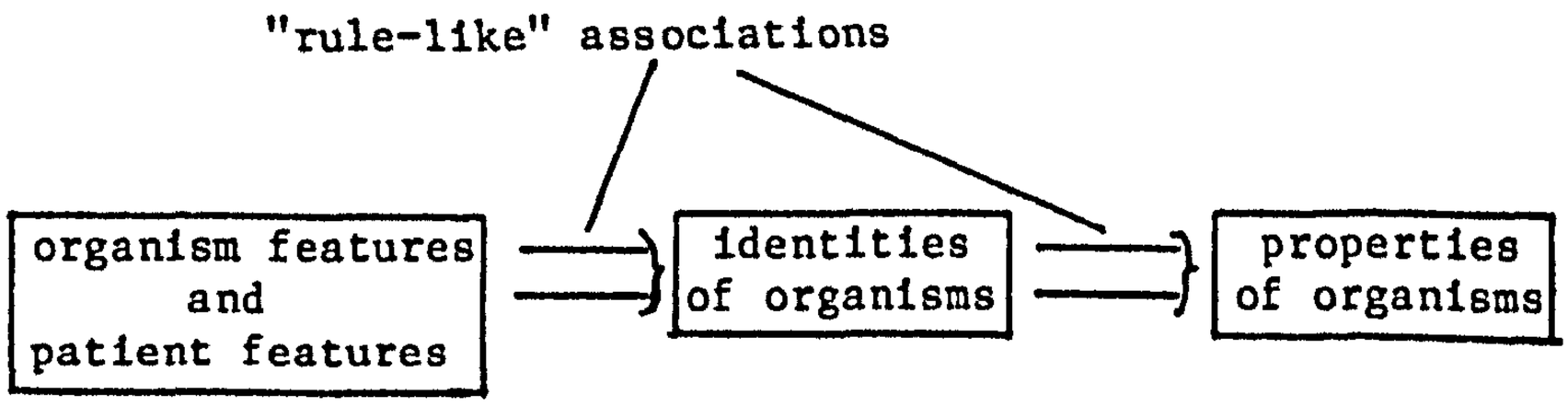


fig 4.8

The organisation of knowledge in MICRO-ORGANISMS-1

Given a patient suffering from an infection caused by a micro-organism, the first part of the organisation of knowledge consists of associations between features of that patient (symptoms etc.),

features of the organism that are visible under a microscope and the likely identity of the organism. This knowledge is represented in a number of rules for identifying unknown organisms; rules such as

```
R1  if  the stain of the organism is gram neg
    and the morphology of the organism is rod
    and the patient has been burned
    then
        there is evidence that the organism is
        pseudomonas(.6).
```

```
R2  if  the stain of the organism is gram neg
    and the morphology of the organism is rod
    and the site of the culture is throat
    then
        there is evidence that the organism is
        klebsiella(.5).
```

Each rule has an associated certainty factor. In this example and in the next we assume MYCIN's certainty factor calculus (Shortliffe, 1976, chapter four).

The second type of "rule-like" association consists of knowledge of properties of particular organisms; properties such as the pathogeny (disease causing ability) of the organism and so on. The rule-base therefore also contains the rules

```
R3 if  the identity of the organism is pseudomonas
   then
       the pathogeny of the organism is high(.7).
```

```
R4 if  the identity of the organism is klebsiella
   then
       the pathogeny of the organism is high(.4).
```

The four rules above, R1 to R4, are shown in fig 4.9 in a notation that is convenient for showing the way in which they interact.

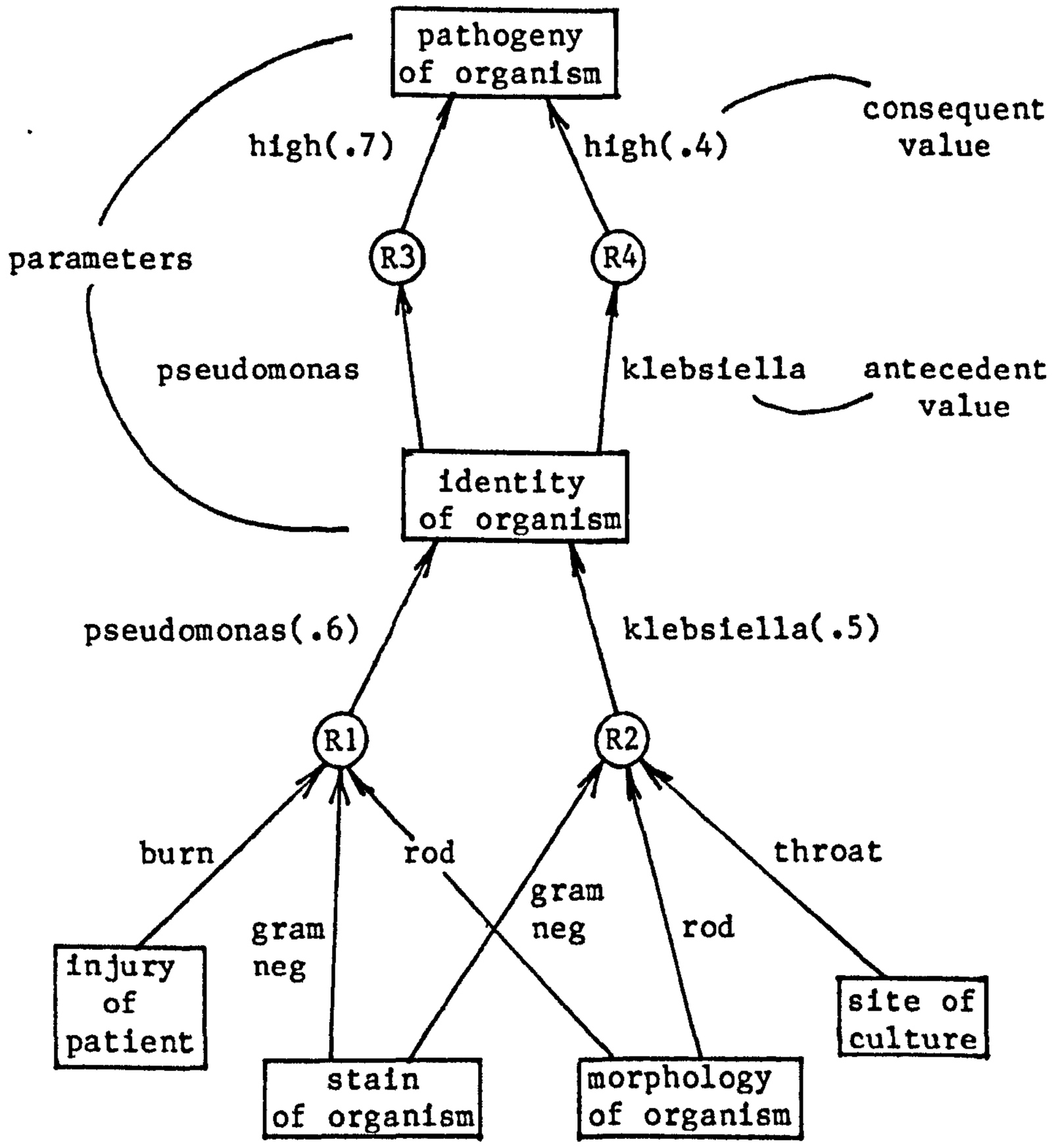


fig 4.9

The rules of MICRO-ORGANISMS-1

In this notation parameters[19] of objects are enclosed in boxes. A rule is shown as a circle with a number of incoming arrows, the antecedents of the rule, and an out going arrow, the consequent of the rule. Arrows are labelled with the values of parameters. MICRO-ORGANISMS-1 was programmed in the KES[20] expert system building shell and appears as appendix D.

For an example of how the rules shown in fig 4.9 operate, consider the following situation. Suppose that an organism has been

---

[19] Parameter is the MYCIN term for attribute.

[20] KES is a product of Software, Architecture and Engineering Inc.

recovered from a culture, grown from a swab taken from an infected patient, and the pathogeny of this organism is required. The backward chaining rule interpreter determines that this information can be supplied by the rules R3 and R4 and therefore attempts to invoke these rules. The premises of the rules R3 and R4 however, cannot be satisfied unless the identity of the organism is known. The identity of the organism becomes the new goal for the rule interpreter. To find the identity of the organism both the rules R1 and R2 are invoked and the user is asked to supply facts about the organism and patient, i.e. the stain and morphology of the organism, whether the patient has been burned and so on. Once R1 and R2 have concluded on the identity of the organism, the value of the identity may be used by the rules R3 and R4 to deduce the pathogeny of the organism.

Modification to MICRO-ORGANISM-1

Having described part of a rule representation of knowledge about infectious organisms and the way in which it operates we now consider the modifiability of this representation. More specifically, we consider how knowledge of groups or categories of organisms, as opposed to individual organisms, can be added to the representation. As an example of the need for such a modification consider the following situation.

Assume that an organism has been isolated from a culture and is found to be a gram negative rod. However, neither of the premises of the rules R1 nor R2 are satisfied because the patient has not suffered burns and the culture was not taken from the throat. Hence the identity of the organism cannot be found and if the identity cannot be found neither the rule R3 nor R4 can be applied to find the organism's pathogeny.

The expert, however, knows that the majority of gram negative rods recovered from cultures have a high pathogeny irrespective of other facts about the patient and would like to add this knowledge to the rule-base. Hence the expert adds the rule

```
R5  if  the stain of the organism is gram neg
    and the morphology of the organism is rod
    then
        there is evidence that the pathogeny of
        the organism is high (.8).
```

Notice that here the expert is not adding knowledge about properties of individual organisms but of a category of organisms, the gram negative rod organisms. The certainty factor of 0.8 for this rule is a value that will be representative of the most pathogenic gram negative rods. The rule, therefore, may cause the system to conclude that the pathogeny of the organism is higher than it is in reality (for example in the case of the organism klebsiella) but this is acceptable because in doubtful cases the system is required to err on the side of caution.

The expert now tests the system with the new rule using only information that the stain of the organism is gram negative and its morphology is rod. The result of the test is that the pathogeny of the organism is found to be high with a certainty factor of 0.8. This is all as it should be. However, in testing the case in which the site of the culture is known to be the throat, a case previously handled correctly by the rule R2, the system now concludes that the pathogeny of the organism is high with a certainty factor greater than 0.8 instead of the correct value of 0.2 (=0.4x0.5). The operation of the rules R1 to R5 which produce the result described above can be seen in fig 4.10.

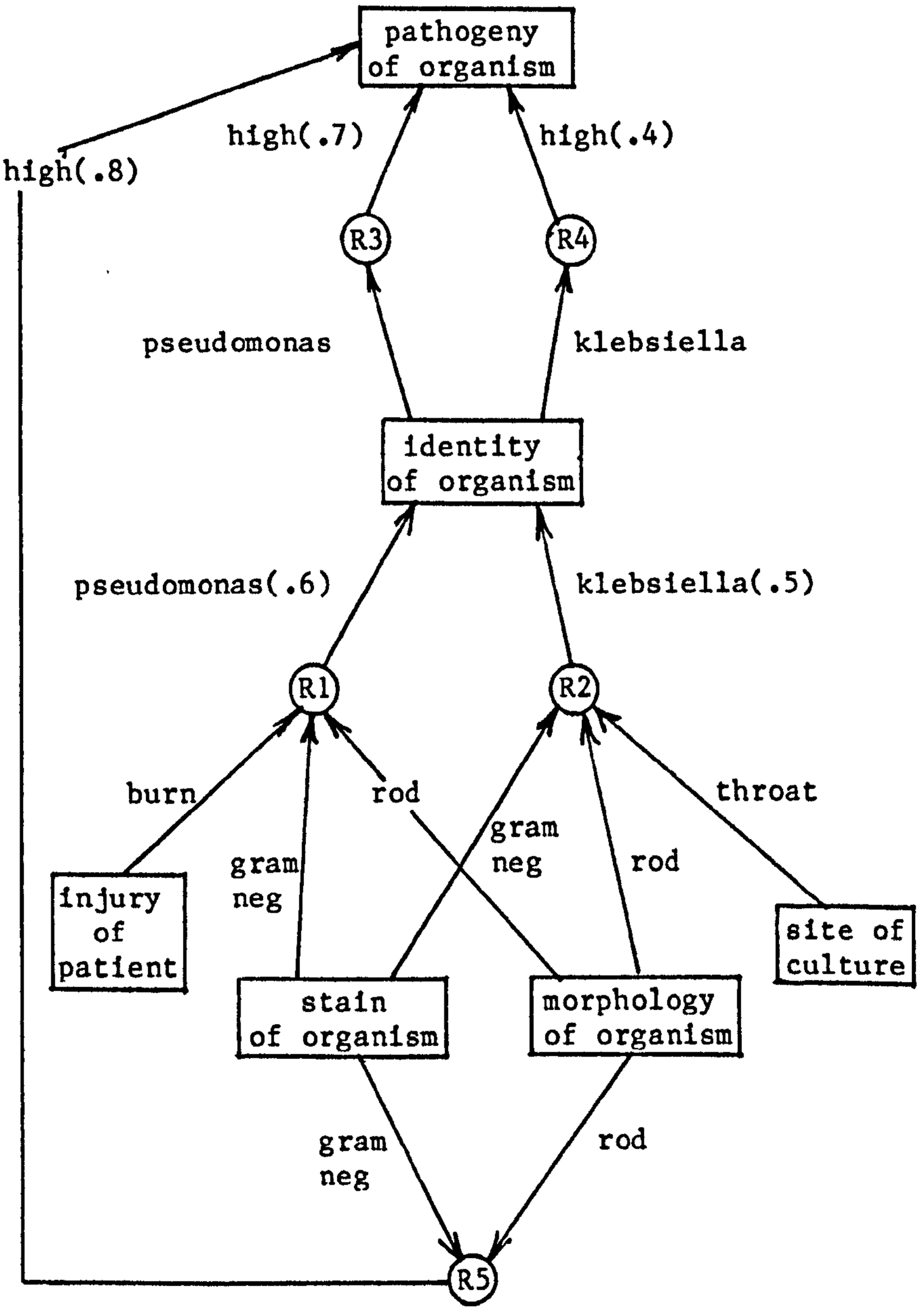


fig 4.10

A first attempt at modifying MICRO-ORGANISMS-1

Notice that the rule interpreter initially attempts to invoke the three rules R3, R4 and R5. R3 cannot be invoked but R4 will be since the identity of the organism can be found to be klebsiella by the invocation of R2. R4 concludes that the pathogeny of the organism is high(.2). R5 is also invoked because the stain and the morphology of the organism are known, hence R5 concludes that the pathogeny of the organism is high (.8). The two values of pathogeny, high(.2) and high

(.8) are "summed" to produce an overall value of high with a certainty factor in excess of 0.8.

The error is caused by the invocation of R5 and R2 in circumstances where only R2 should be invoked. Clearly, the invocation of R5 in this case was not something the expert had intended. The expert had intended that R5 should be used only when there is not sufficient information to determine the identity of the organism using R1 or R2. In other words, the intention was that individual organisms should not inherit the properties of the category of organisms to which they belong if more specific information is available.

To sum up, we have described a rule representation of knowledge about infectious organisms. We have also described an attempted modification to the representation, the addition of knowledge about a property of a category of organisms, and shown that the modification cannot be performed as an extension.

Before we present the second rule representation of knowledge, the representation in which the modification described above can be performed as an extension, it is interesting to note that there is an expedient method of ensuring that R5 is not incorrectly invoked. Since R5 should not be invoked when the identity of the organism is known, the premise of R5 should be modified to include an antecedent that is false if the identity of the organism is known. The modified rule, renamed R5.1, is shown below.

```
R5.1  if  the stain of the organism is gram neg
      and the morphology of the organism is rod
      and the identity of the organism in not known
      then
          there is evidence that the pathogeny of
          the organism is high (.8).
```

If either R1 or R2 are invoked the identity of the organism will be

known and the third antecedent of R5.1 will be false. Therefore, although the backward chaining interpreter attempts to invoke the rule R5.1 even though the identity of the organism is known, the premise of the rule is not satisfied, and the rule is not invoked, the system thereby producing the correct value for the pathogeny of the organism.

R5.1 represents, in addition to the knowledge represented in R5, knowledge that the pathogeny of the gram negative rod organisms should be used only if the identity of the organism cannot be found. All this knowledge can be added as an extension to the knowledge-base in the form of rule R5.1 In the organisation of knowledge in MICRO-ORGANISMS-1, knowledge of the properties of a category of organisms cannot be added to the representation without also adding knowledge of when to use those properties. Consequently, knowledge of the properties of a category cannot be added as an extension.

It is worth pausing at this stage in the discussion of the example in order to generalise the previous point. We attempted to add knowledge to the rule-base but found that the new knowledge was incompatible with that already represented there. Specifically it was necessary to also included knowledge about how the newly added knowledge should be used. The need to add this extra knowledge goes against what Shortliffe (1976, p70) has to say of the rule representation scheme.

> A modular system, on the other hand, permits knowledge to be acquired as isolated facts and allows the consultation program itself to decide under what conditions the new information is relevant.
>
> We accomplish modularity of system knowledge by storing all information in decision rules.
>
> ibid, p71.

The root cause of the inability to add rules representing properties of categories of organisms lies in the organisation of

knowledge. In this specific example there there is no knowledge in the knowledge-base that properties of categories of organisms should only be used when the specific identity of the organism is unknown. Notice that the addition of the antecedent

the identity of the organism is not known

in the rule R5, to produce R5.1, only adds knowledge of how to treat properties of organisms within the specific category of gram negative rods.

For example, consider the following rules, similar to those considered above but about gram positive rods.

```
R6 if  the stain of the organism is gram pos
   and the morphology of the organism is rod
   and the patient has recently had an operation
   then
       there is evidence that the organism is bacillus.A(.6).
```

```
R7 if  the stain of the organism is gram pos
   and the morphology of the organism is rod
   and the site of the culture is skin
   then
       there is evidence that the organism is bacillus.D(.5).
```

```
R8 if the identity of the organism is bacillus.A
   then
      the pathogeny of the organism is moderate(.7).
```

```
R9 if the identity of the organism is bacillus.D
   then
      the pathogeny of the organism is moderate(.4).
```

Again the expert wishes to add a rule

```
R10 if  the stain of the organism is gram pos
    and the morphology of the organism is rod
    then
        there is evidence that the pathogeny
        of the organism is moderate(.8).
```

that encodes the pathogeny of gram positive rod organisms in general.

The addition of R10 into the rule-base will introduce essentially the

same error that was introduced with the addition of R5.

The organisation of knowledge we require must include knowledge about the correct inheritance of properties of any category of organisms. This organisation of knowledge is now presented as MICRO-ORGANISMS-2.

MICRO-ORGANISMS-2

An organisation of knowledge that allows knowledge of the properties of categories of organism to be added as extensions is illustrated in fig 4.11.

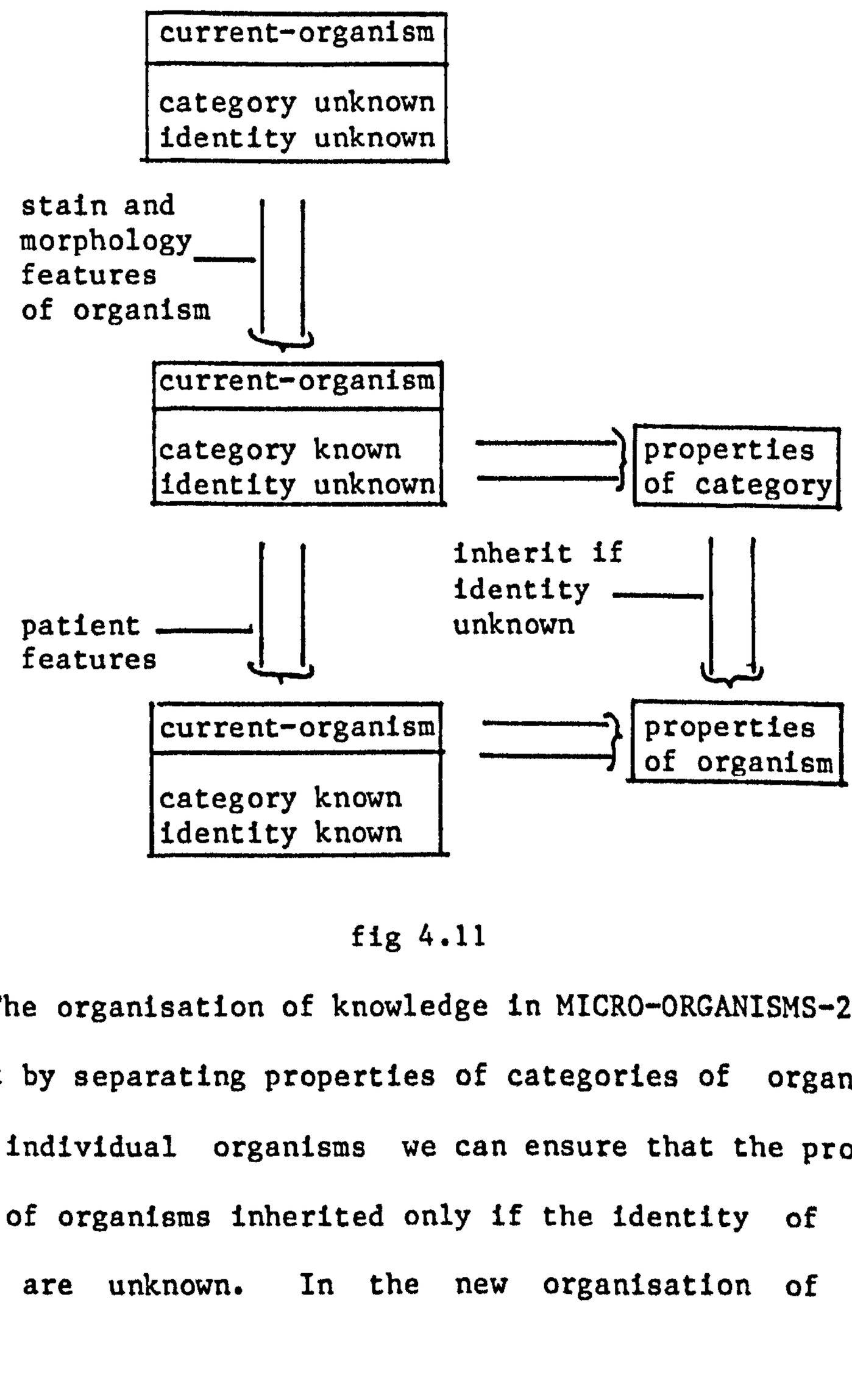


fig 4.11

The organisation of knowledge in MICRO-ORGANISMS-2

Notice that by separating properties of categories of organisms from those of individual organisms we can ensure that the properties of categories of organisms inherited only if the identity of individual organisms are unknown. In the new organisation of knowledge,

categories of organisms are treated in a similar manner to individual organisms. For example there are rules to identify categories of organisms, such as the rules

```
R6 if  the stain of the organism is gram-neg
   and the morphology of the organism is rod
   then
        the identity of the category of organisms
        is gram-neg-rods (1.0).

R7 if  the stain of the organism is gram-pos
   and the morphology of the organism is rod
   then
        the identity of the category of organisms
        is gram-pos-rods (1.0).
```

and there are rules to represent the properties of categories, rules such as

```
R8 if  the identity of the category of organisms
       is gram-neg-rods
   then
       the pathogeny of the category of organisms is high(.8).

R9 if  the identity of the category of organisms
       is gram-pos-rods
   then
       the pathogeny of the category of organisms is moderate(.4).
```

The identities of individual organisms are established by making use of the category to which the organism belongs and features of the patient, hence the rules R1′ and R2′ below (analogous to R1 and R2 in MICRO-ORGANISMS-1).

```
R1′ if  the identity of the category of organisms is gram-neg-rods
    and the patient has been burned
    then
        the identity of the organism is pseudomonas(.6).

R2′ if  the identity of the category of organisms is gram-neg-rods
    and the site of the culture is throat
    then
        the identity of the organism is klebsiella(.5).
```

The interaction of these rules can be seen in fig 4.12

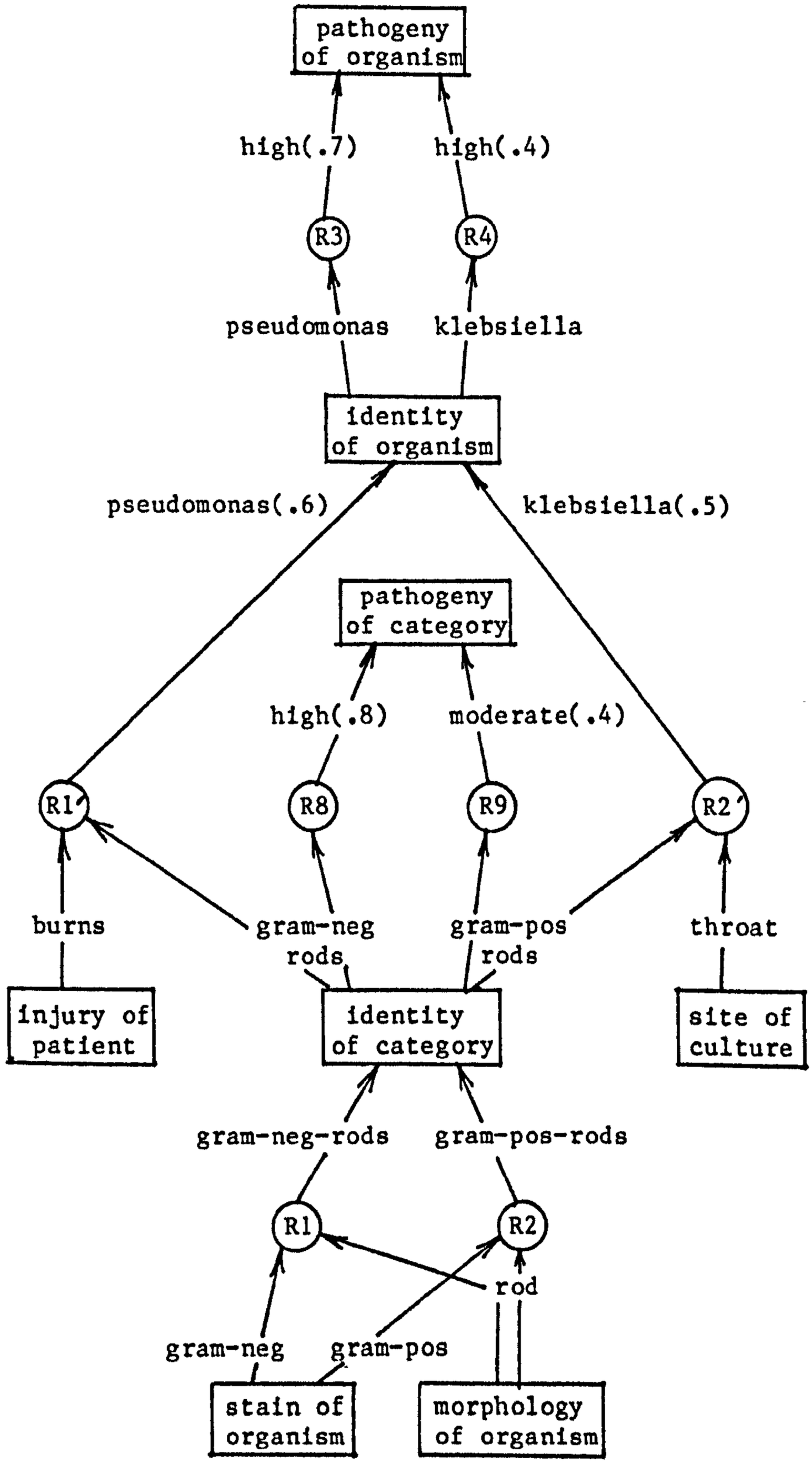


fig 4.12

The rules of MICRO-ORGANISMS-2

In this organisation of knowledge whenever the attempt is made to find the pathogeny of the organism the backward chaining interpreter

retrieves only those rules that conclude on the pathogeny of the the organism and not the pathogeny of the category. This is fine if the identity of the organism can be found. However, if the identity of the organism cannot be found but the identity of the category is known, then the pathogeny of the category should be taken as the pathogeny of the organism. The following rule allows the properties of the category to be used for the properties of an unknown organism.

```
R10 if  the identity of the organism is unknown
    and the identity of the category is known
    and the property parameters of the category are known
    then
        let the values of the property parameters of the organism
        be the values of the property parameters of the category.
```

Notice that R10 above cannot be invoked when the identity of the organism is known and hence cannot be invoked incorrectly in the way that R5 could be in the previous representation. MICRO-ORGANISMS-2 has been implemented in the KES expert system building shell and appears as appendix E.

Modification to MICRO-ORGANISM-2

Now that the improved rule representation has been described we can demonstrate the addition of properties of categories of organisms as extensions to the knowledge-base. For example, assume that the rule-base contains the following rule that encodes the aerobicity of an organism.

```
R11 if  the identity of the organism is pseudomonas
    then
        the aerobicity of the organism is aerobic (.8).
```

In this representation it is now possible to add knowledge about the aerobicity of the gram-neg-rods category of organisms as the rule

```
R12 if  the identity of the category of organisms
        is gram-neg-rods
    then
        the aerobicity of the category of organisms
        is aerobic (.6).
```

The inheritance rule, R10 above, ensures that the aerobicity of the gram negative rods category can be taken to be the aerobicity of the organism but only in the case that the identity of the organism cannot be found.

To summarise, in this second example we have described two rule representations of knowledge, only one of which allows the addition of a particular kind of knowledge as an extension. Clearly the mere fact that our representation is in rule form cannot account for the addition of this knowledge as an extension. The extension of the knowledge-base with knowledge about properties of organisms is made possible by treating properties of individual organisms as distinct from properties of categories of organisms and by allowing properties to be inherited if required. In short, the particular extensibility of MICRO-ORGANISMS-2 is due to its organisation of knowledge.

In general, we conclude that rule representations of knowledge need not necessarily be extensible. In other words extensibility does not merely follow from representing knowledge as rules. Rule representations that are extensible, are so by virtue of the organisation of knowledge in the representation.

CHAPTER FIVE

THE EXTENSIBILITY OF NON-RULE REPRESENTATIONS OF KNOWLEDGE

INTRODUCTION

In the previous chapter we argued that a rule representation of knowledge is only suitable for a particular extension if it represents an appropriate organisation of knowledge. However, the notion of an extension is not limited to rule representations of knowledge. Indeed, one way of examining claims about the modifiability of rule representations is to consider the modifiability of these representations as opposed to non-rule representations. For instance, there is the question of whether non-rule representations of knowledge can be extended or not.

As far as rule representations are concerned we argued that extensibility arises from the particular organisation of knowledge in the representation. The question arises as to whether the rule representation plays any part at all in the extensibility of the representation. If, for example, the organisation of knowledge in a rule representation does not depend on the representation of that organisation as rules then we can expect the extensibility of a non-rule representation of that organisation to be the same as that of the rule representation. In this chapter we indeed show that as far as extensibility is concerned, there in no difference between rule and non-rule representations of knowledge.

We show this equivalence between rule and non-rule representations by showing the irrelevance of the rule representation scheme in the extensibility of the examples used in the previous chapter. We show that the organisation of knowledge in NAVIGATE-2 need not be represented as rules in order to be extensible. Secondly we

demonstrate the extensibility of a non-rule representation of the organisation of knowledge in MICRO-ORGANISMS-2.

NAVIGATE-3: A NON-RULE REPRESENTATION OF NAVIGATE-2

Instead of representing the navigational knowledge of NAVIGATE-2 as rules we represent this knowledge as frame-like data structures. The frame scheme for the representation of knowledge is described by Minsky (1975) as a data structure consisting of a network of nodes and relations for representing a stereotypical situation or object. A frame can be thought of as a structured collection of slots. Slots may contain information about the properties of a situation or an object or knowledge in the form of a procedure. The procedure may reason with the knowledge represented in other slots of the frame or with knowledge represented in other frames. A slot may contain default values and may itself be a frame.

For example, in NAVIGATE-2, navigational knowledge about the towns that are joined to a particular town is represented in a 'next-stage' rule. In NAVIGATE-3, however, navigational knowledge about a town, it's name and towns joined to it, is represented in the frame-like data-structure shown below.

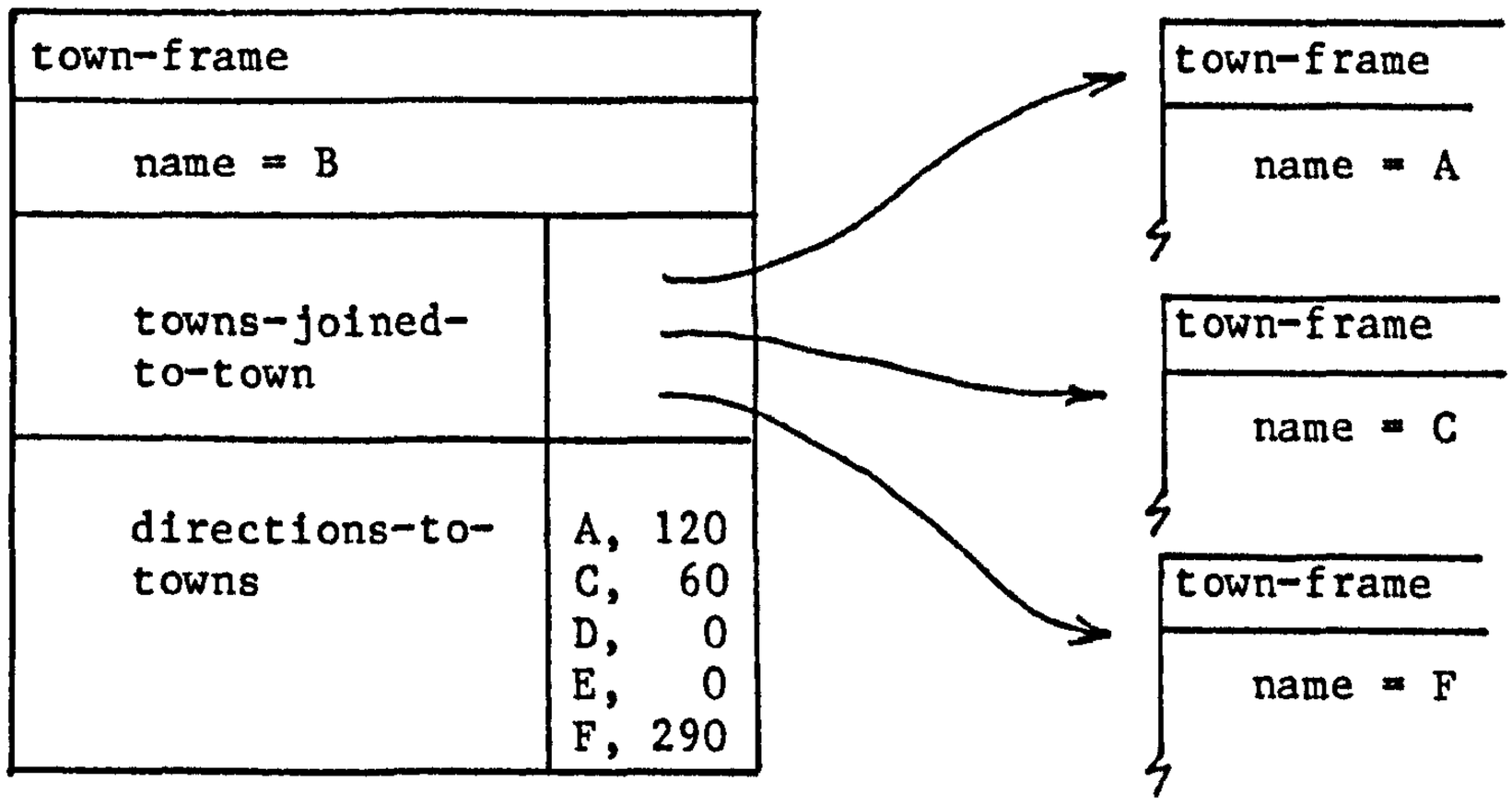


fig 5.1

Frame representation of a town

Each frame has a number of slots for holding specific kinds of knowledge. Some slots may contain other frames for example, the 'towns-joined-to-town' slot contains the frames corresponding to each of the towns joined to the town. The 'direction-to-towns' slot contains a list of the directions to all other towns.

The knowledge of how to go about finding a route between two towns is also represented as a frame-like data-structure.

route-frame	
input	starting town and destination town
output	sequence of towns beginning with the starting town and ending with the destination town
find-route	procedure for finding the the route between two towns

fig 5.2

Frame representation of route procedure

The procedure for finding the route between two towns is represented as a procedure that takes as arguments two towns and returns the route between them. The procedure finds a route by iteration of a number of

steps and is as follows.

```
find-route (starting-town, destination) =
  current-town := starting-town;
  route := starting-town;
  until end-route(route, destination)
  do
    next-towns := current-town.towns-joined-to-town;
    best-town := best-stage(current-town, next-towns, destination);
    route := route + best-town;
    current-town := best-town
  od.
```

The function `end-route' is true if the ultimate town in `route' is equal to the destination and false otherwise. The expression

```
current-town.towns-joined-to-town
```

signifies the contents of the `towns-joined-to-town' slot of this frame, i.e. the list of frame representing the towns joined to the current town. The `best-stage' function compares the directions from the `current-town' to each of the `next-towns' and selects the `next-town' whose direction is most suitable for the overall direction of travel. This function is analogous to the `best-stage' rule in NAVIGATE-2. The operation of NAVIGATE-3 as it finds a route from A to D is shown below.

starting town = A destination = D

current-town	next-towns	best-town	route	end-route
A	B	B	A B	false
B	A C F	C	A B C	false
C	B D E	D	A B C D	true

fig 5.3

The operation of NAVIGATE-3

NAVIGATE-3 has been implemented in the ML functional language and appears as appendix F.

EXTENSIBILITY OF NAVIGATE-3

Given that the organisation of knowledge in NAVIGATE-3 is essentially[21] that in NAVIGATE-2 we now compare the extensibility of NAVIGATE-3 with that of NAVIGATE-2. More specifically we are concerned to discover whether facts about new roads can be added as extensions to NAVIGATE-3.

Consider once again the situation that arises when a new road is built from B to E. In NAVIGATE-3 the addition of a new road from B to E requires that a modification is made to the frame data-structure representing the town B. In this frame the 'towns-joined-to-town' slot must be modified to include the new town E, and since no other knowledge in NAVIGATE-3 need be modified and no additional knowledge must be elicited from the expert, the modification is an extension. The modification to the frame representing town B is shown in fig 5.4.

---

[21] We mean here that at some appropriate level of description the organisations of knowledge in NAVIGATE-2 and NAVIGATE-3 are identical. The question arises as to which level of description is an appropriate one. There is no single answer to this question, it depends very much on the purpose of the comparison and our purpose is to investigate representations of knowledge in terms of modifiability.

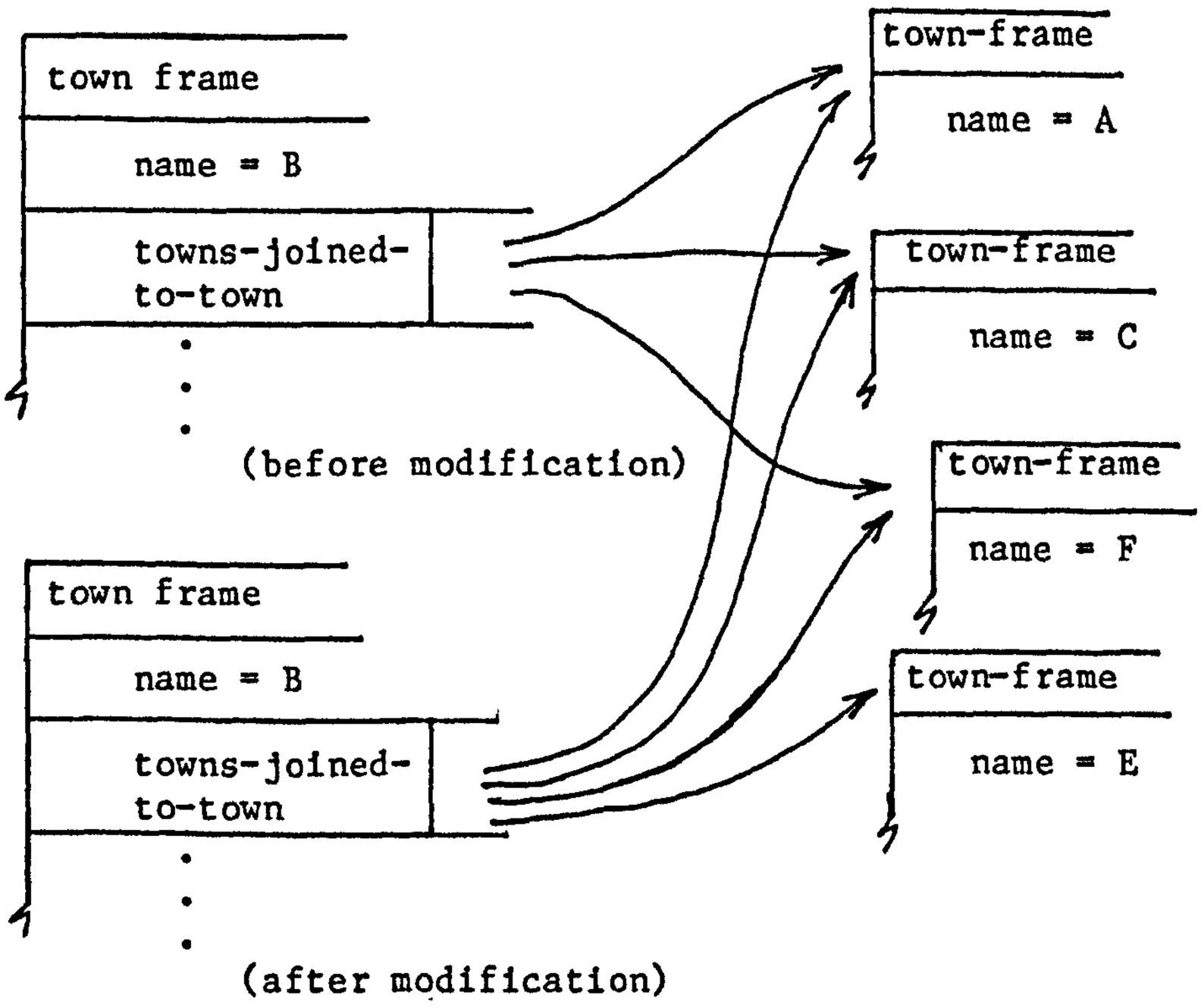


fig 5.4

The modification of NAVIGATE-3

This modification is analogous to the modification of the next-stage rule in NAVIGATE-2.

In NAVIGATE-2, it is the rules representing the roads leaving a particular town can be extended. In NAVIGATE-3, slots representing the roads leaving a particular town can be extended. Surely the representation of roads as rules or slots in irrelevant to the extensibility of the knowledge. NAVIGATE-2 and NAVIGATE-3 are both representations of the same organisation of knowledge and in this organisation of knowledge facts about new roads can be added as extensions. In other words, it is because facts about new roads can be added as extensions to this organisation of knowledge that, in NAVIGATE-2 and in NAVIGATE-3, rules and slots (as appropriate) can be extended.

MICRO-ORGANISMS-3: A NON-RULE REPRESENTATION OF MICRO-ORGANISMS-2

In this section, we continue to show the equivalence, in terms of extensibility, between rule representations of knowledge and non-rule representations. We do this by demonstrating the extensibility of MICRO-ORGANISMS-3, a frame representation of the organisation of knowledge in MICRO-ORGANISMS-2.

In MICRO-ORGANISMS-3, the knowledge about each organism is represented in a frame. Similarly the knowledge about each category of organisms is represented as a frame. Figure 5.2 shows the frames that represent a taxonomy of organisms.

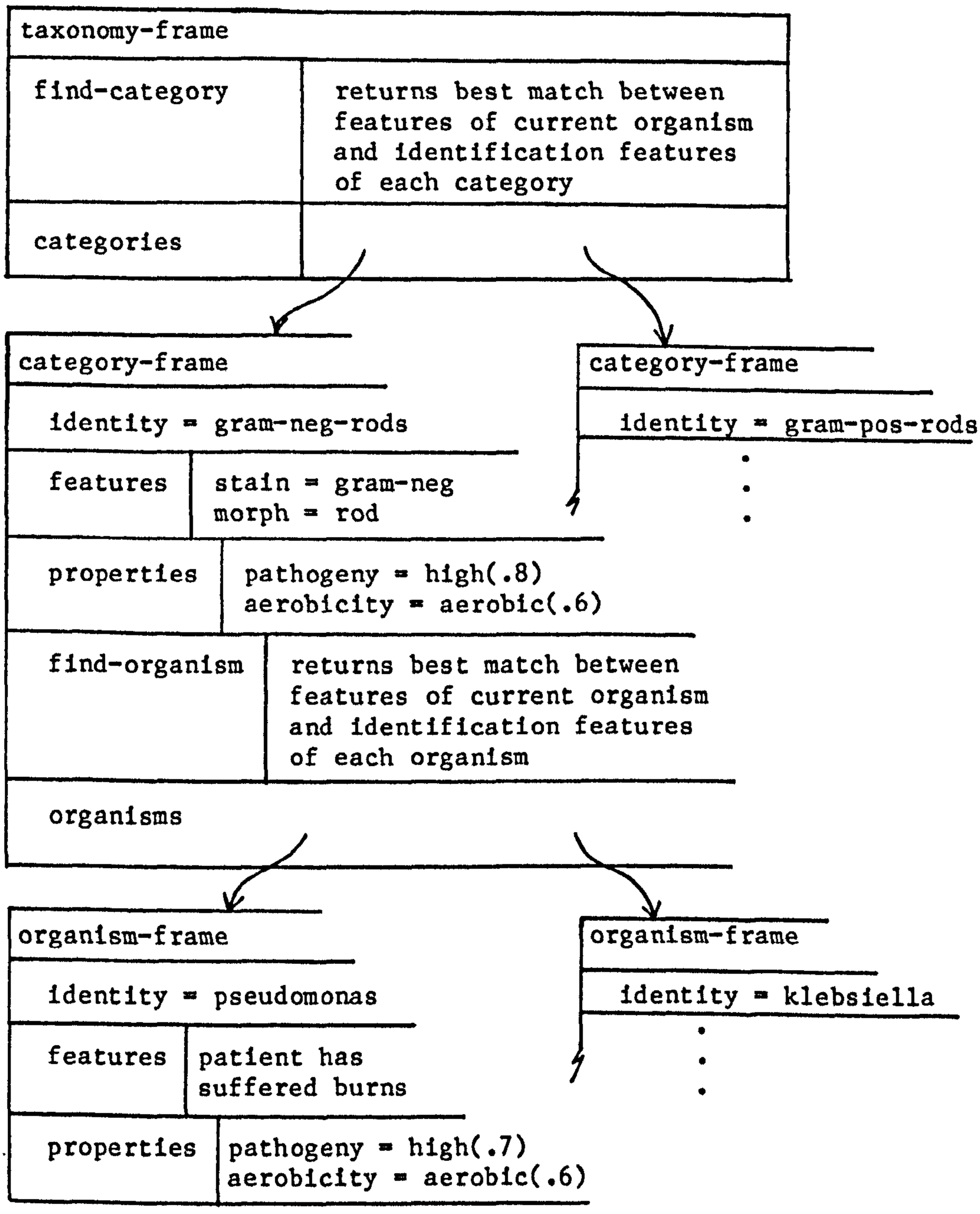


fig 5.5

Frame representation of organisms and categories of organisms

The taxonomy of organisms consists of the collection of categories of organisms and each category in turn consists of a collection of individual organisms. The taxonomy frame also contains a procedure for identifying an unknown organism as a member of a particular category of the taxonomy. This procedure is supplied with features of the unknown organism and is able to match these features against the characteristic features of each category and select the category that

provides the best match. The identification or characteristic features of each category are represented together with other facts about that category in a category frame. A category frame represents knowledge of the properties of that category, the organisms that belong to that category and a procedure for identifying the unknown organism of the category as a particular individual of the category. This procedure is analogous to that described above for identifying the category of the organism. The knowledge of each organism is represented as a frame as shown in fig 5.5.

Knowledge about the patient is represented in a patient-frame, shown in fig 5.6.

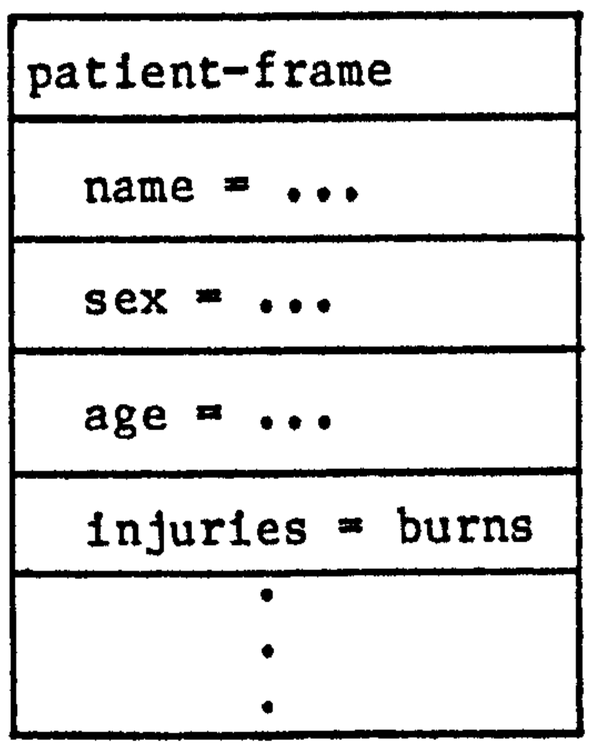


fig 5.6

Frame representation of patient data

The information about each organism recovered from a culture is also represented in a frame-like data-structure, an example of which is shown in fig 5.7.

current-organism-frame	
label = current-organism-3	
site-of-culture	throat
identity	procedure to identify current-organism
category	procedure to identify category of current-organism
features	stain = gram-neg morph = rod
properties	procedure to find properties

fig 5.7

Frame representation of an organism under investigation

The features by which the organism may be identified are provided by the user of the system.

The 'category' slot of the current organism frame contains a procedure for establishing the category of the organism. The procedure simply matches features of the unknown organism against the identification features of each category in the taxonomy. For example an unknown organism with a gram negative stain and a rod morphology will be categorised as a gram negative rod because its stain and morphology match the identification features of the gram negative rods category. In general the procedure for identifying the category to which the current organism belongs is

```
current-organism.category =
  taxonomy.find-category(current-organism.features, patient)
```

The procedure simply supplies the features of the current organism and patient to the category identification procedure in the taxonomy frame.

The procedure for finding the individual identity of the current organism consists of two steps. In the first step the category to which the organism belongs is found. This is done by invoking the procedure for identifying the category of the current organism just described above. If the category of the organism is successfully identified then the organism identification procedure proceeds to the second step. Knowing now the category to which the organism belongs, the second step is to identify the particular organism in that category. This identification is made by supplying the features of the current organism to the identification procedure in the frame representing the current organism's category. Again, features of the unknown organism and patient are simply matched against the identification features of each organism in the unknown organism's category. If the features of the current organism are insufficient for a complete identification then the category of the organism is returned. The procedure just described is shown below.

```
current-organism.identity =
  current-category := current-organism.category;
  if known(current-category)
  then
    (current-identity := current-category.find-organism
                              (current-organism.features, patient);
     if known(current-identity)
     then current-identity
     else current-category
    )
  else category-unknown.
```

Finally we can describe the procedure for obtaining the value of a property of the current organism. The first step is to establish the identity, or the category, of the organism and retrieve the corresponding frame. The second step is to retrieve the value of the required property slot in this frame. The procedure is shown below.

```
current-organism.properties(prop) =
  (current-organism.identity).properties.prop.
```

Notice that this procedure has the all important behaviour of not retrieving the properties of the category of an unknown organism if the identity of that organism can be found.

THE EXTENSIBILITY OF MICRO-ORGANISMS-3

We are now able to demonstrate the extensibility of the above representation of knowledge in terms of adding knowledge about properties of categories of organisms. Recall that in the previous chapter a problem arose if the expert tried to add knowledge of a property of a category of organisms which in certain circumstances could be taken to be the property of the organism. To add knowledge of a property of a category in MICRO-ORGANISMS-3 the expert need only modify the appropriate property slot of the category. For example if gram positive organisms have an aerobicity of aerobic (.6) then this can be simply added to the properties slot of the frame representing the gram positive rods category. No other knowledge in the representation need be modified, or additional knowledge elicited, in order to ensure that properties of categories are used correctly.

The properties of categories are always used correctly because the procedure for finding the identity of the current organism returns the category of the organism only in the case that the identity cannot be found, consequently any property of a category is inherited by an individual organism only if the identity of the current organism cannot be found.

In this chapter we have demonstrated the extensibility of frame representations of the example organisations of knowledge introduced in the previous chapter. Although we chose to use frame representations the reader will appreciate that our analysis does not depend on the choice of this scheme. Our argument would have

proceeded equally well had we used a semantic net or predicate calculus representation scheme. Hence we conclude that the extensibility of a given representation of knowledge depends on the organisation of knowledge in that representation and not on the particular representation scheme used.

CHAPTER SIX

THE RELATION BETWEEN MODULARITY AND MODIFIABILITY

INTRODUCTION

One of the justifications advanced for the modifiability of rule representations of knowledge is their so called "modularity". For example, recall how Shortliffe (1976, p70) describes how new knowledge can be added to MYCIN.

> ...an inference model that depends on a complex decision tree is apt to be difficult to augment without a complete diagram of the tree so that all implications of additions can be observed. A modular system, on the other hand, permits knowledge to be acquired as isolated facts and allows the consultation program itself to decide under what conditions the new information is relevant.
>
> We accomplish modularity of system knowledge by storing all information in decision rules.
>
> ibid, p71.

> ... one of the major design considerations during the development of MYCIN has been the isolation of pieces of knowledge as discrete facts. MYCIN's decision rules achieve this goal. Since each rule represents a discrete packet of knowledge, the integration of new information into the system is simplified.
>
> ibid, p155.

Recall also the claim made of the production rule representation scheme by the authors of the PROSPECTOR expert system (Duda et al, 1978, p203,204).

> The advantages of this [use of production rules] approach stem from the fact that the representation is modular and declarative. This ... encourages incremental development, ...

Waterman (1978, p278,9), also refers to the modularity of the production rule encoding of knowledge.

> The modularity of production systems is viewed as the critical factor underlying the successful approach taken to program creation.

Many authors, such as those quoted above, simply declare the "modularity" of the rule scheme as a justification for scheme's supposed modifiability. Davis and King (1977) however, deal at length with the characteristics of production systems and attempt to reason more deeply about the "modularity" of the rule scheme. Davis and King (ibid, p306) describe the relationship between the "modularity" of production systems and the ease with which the system can be modified as follows.

> This inherent modularity of pure production systems eases the task of programming in them. Given some primitive action that the system fails to perform, it becomes a matter of writing a rule whose LHS matches the relevant indicators in the data base, and whose RHS performs the action.

It is this sort of justification for the modifiability of rule representations that we will now examine.

THE CONCEPT OF A MODULE

Before examining the relationship between modularity and modifiability in rule representations we discuss the concept of a module. The module is an important programming languages and systems concept. The concept of a module is captured well by Dennis' (Dennis, 1973, p128) simple example from the construction materials trade.

> In the United States floor tile comes (sic) in nine-inch squares (the modules) which may be conveniently adjoined to fill up any shape of floor area with just a bit of trimming at the boundary. A great variety of patterns may be produced by using modules of differing color and texture.

Wirth describes the concept as it applies to software systems, (Wirth, 1980, p9).

> The module is effectively a bracket around a group of (type, variable, procedure, etc) declarations establishing a scope of identifiers. In the first instance, we may regard this bracket as an impenetrable wall, objects declared outside of the module are invisible inside it, and those declared inside are invisible outside. This wall is punctured selectively by two lists of identifiers: the import list contains those

identifiers defined outside which are to be visible inside too, and the export list contains the identifies defined inside that are to be visible outside.

A module will have a particular function to perform. In the case of the floor tiles example, the function of each tile is to cover a portion of the floor, whereas in the case of the software module, the function of the module is the module's input/output function.

Modules have the important property that their functions do not change as a result of á change in the context in which the module is used. For example, in the case of the floor tiles, the size and colour of a tile does not depend on the position of the tile nor on the tiles immediately adjacent to it. In the case of the software module, its input/output function should not depend on the context of the modules use, i.e. the output of the module should depend only on the input.

MODULES AND MODIFIABILITY

The use of modules plays an important part in the modifiability of programs, software systems and systems in general. For a system constructed of modules there is the possibility that a modification to that system can be performed by adding deleting or replacing a few modules in a straightforward way. In this case the modification will be straightforward and convenient.

However, there is also the possibility that the modification cannot be performed by the simple addition or deletion of a few modules. As a simple example consider a floor covered by plain white tiles. Suppose further that the floor is to be modified to include a single black stripe across it. If the stripe is to be nine inches

wide then this is fine[22] but if it is to be ten inches wide then the modification is troublesome. Given that the floor covering is to be modified to incorporate ten inch stripes it would have been better to have used ten inch tiles.

In general there is a number of ways in which modules can be chosen for the construction of a given system. However, relative to a particular class of modifications only some of these choices will lead to programs that are conveniently modifiable. Parnas (1972) makes exactly this point about the choice of modules for the construction of a software system. He describes two programs, each of which consist of a number of modules. Only one of the programs however has the "appropriate" modules to make modifications convenient. Parnas (ibid, p1053) concludes

> The effectiveness of a "modularization" [particular choice of modules] is dependent upon the criteria used in dividing the system into modules.

MODULARITY OF PRODUCTION SYSTEMS

Returning now to the so called "modularity" of the rule scheme, we notice that each rule in the rule-base is a candidate for a module of the system. However, depending on the characteristics of the rule language, a rule may or may not be a module. The input/output function of a rule is the mapping between its LHS and its RHS. Consequently, if a rule is to qualify as a module this function should be independent of the context of the application of the rule. Now, if the spirit of the production system architecture is adhered to then a rule should only be sensitive to the state of the common database through the mechanism of matching its LHS, and then if the rule is applied, it should simply execute its RHS. In this situation the

---

[22] Recall that floor tiles are nine inches square.

function of a rule will indeed be independent of the context of its use and hence the rule will qualify as a module. For example, the rules of NAVIGATE-1 are modules because each rule performs its function, the provision of a single route, irrespective of the context in which the rule is used.

However, in practice it is possible to implement a rule language with rules that make use of, and modify, "hidden" variables. These are variables that are not accessed through the common database via the LHS and RHS of rules but can nevertheless influence the behaviour of particular rules. If the function of the rule (the mapping between the values satisfying its LHS and the values of the RHS) depends on such "hidden" variables then the rule is not a module.

Given our view of the relationship between the concept of a module and the rules in a production system, consider now the way in which Davis and King (1977, p316) define the "modularity" of the rule scheme.

> We can regard the modularity of a program as the degree of separation of its functional units into isolatable pieces. A program is highly modular if any functional unit can be changed (added, deleted, or replaced) with no unanticipated change to other functional units. Thus program modularity is inversely related to the strength of coupling between its functional units. The modularity of programs written as pure production systems arises from the important fact that the next rule to be invoked is determined solely by the contents of the data base and no rule is ever called directly.

This conception of modularity could well conform to the software systems conception of modularity if the phrase

'no unanticipated changes to other functional units'

is interpreted in a particular way. Specifically the interpretation hinges on the reason why changes to other functional units are required. For instance, suppose that a modification to a program

requires that an additional function be performed at a certain place in the program. When a functional unit with the required function is added at the appropriate place it is subsequently discovered that the functional unit produces side effects in addition to its function. In other words, the newly added functional unit is not a module because there is no 'impenetrable wall' between its internal workings and the rest of the program. If the changes made to counter these undesirable side effects correspond to the 'unanticipated changes' mentioned by Davis and King above then these authors mean 'modularity' to be understood in the conventional software systems sense.

However, immediately following the passage quoted above

> Thus the addition (or deletion) of a rule does not require the modification of any other rule to provide for (delete) a call to it.

This characteristic saves the programmer a chore. However, Davis and King go on to argue that the absence of a need to amend calls to newly added or deleted rules is "symptomatic" of a particular program organisation.

> We might demonstrate this by repeatedly removing rules from a PS[production system]: many systems will continue to display some sort of "reasonable" behaviour, up to a point.

For example, NAVIGATE-1 will continue to behave perfectly as rules are deleted unless the system is asked for a route that has been deleted. Davis and King contrast this situation with that commonly found in Algol-like programs.

> By contrast, adding a procedure to an ALGOL-like program requires modification of other parts of the code to insure that it is invoked, while removing an arbitrary procedure from such a program will generally cripple it.

We can illustrate the sort of organisation of knowledge that Davis and King are describing here by considering the organisation of knowledge

in NAVIGATE-3. The deletion of any single line in the procedure that calculates routes, viz.

```
find-route (starting-town, destination) =
  current-town := starting-town;
  route := starting-town;
  until end-route(route, destination)
  do
    next-towns := current-town.towns-joined-to-town;
    best-town := best-stage(current-town, next-towns, destination);
    route := route + best-town;
    current-town := best-town
  od.
```

will (as Davis and King point out) completely destroy it.

> Note that the issue here is more than simply the "undefined function" error message which would result from a missing procedure. The problem persists even if the compiler or interpreter were altered to treat undefined functions as no-ops.

In other words, there is more at stake here than relieving the programmer of the chore of inserting calls in the procedures that should call a newly added procedure and deleting calls in all the procedures that call a non-existent procedure. Davis and King elaborate this point as follows.

> The issue is a much more fundamental one concerning organization of knowledge: programs written in procedure-oriented languages stress the kind of explicit passing of control from one section of code to another that is characterized by the calling of procedures. This is typically done at a selected time and in a particular context, both carefully chosen by the programmer. If a no-op is substituted for a missing procedure, the context upon returning will not be what the programmer expected, and subsequent procedure calls will be executed in increasingly incorrect environments. Similarly, procedures which have been added must be called from somewhere in the program, but the location of the call must be chosen carefully if the effect is to be meaningful.

Davis and King contrast the kind of organisation demonstrated above with that which they claim is present in production systems.

> Production systems, on the other hand, especially in their pure form, emphasize the decoupling of control flow from the writing of rules. Each rule is designed to be ideally, an independent chunk of knowledge with its own statements of

relevance (either the conditions of the LHS[left hand side], as in a data-driven system, or the action of the RHS, as in a goal-directed system). Thus where the ALGOL programmer carefully chooses the order of procedure calls to create a selected sequence of environments, in a production system it is the environment which chooses the next rule for execution. And since a rule can only be chosen if its criteria of relevance have been met, the choice will continue to be a plausible one, and system behaviour remain "reasonable," even as rules are successively deleted.

The particular organisation of knowledge that Davis and King are describing is an organisation of knowledge in which the knowledge represented in rules can, to some extent, be freely added and deleted. These are the organisations of knowledge that we have so far described as extensible.

A possible interpretation of the initial part of Davis and King's discussion is that they are pointing out that the rules of a production system are modules (in the conventional sense); this is a simple consequence of the production system architecture. They then discuss the feature of the production system architecture that ensures that the rules of the system are indeed modules (conventional sense), namely the mechanism of indirect calls between rules via the common data base. This feature obviates the need for explicit calls between rules. Davis and King however, make much more of this feature and claim that it leads to a particular kind of organisation of knowledge, which they also describe as "modular". Having begun by describing the features of production systems that lead to modularity (conventional sense) they mistakenly slip into claiming that the production system architecture leads to what is in effect a suitable "modularisation" (having appropriate modules) in the sense of Parnas described above.

Davis and King do not make clear the nature of the relationship between the fact that in a production system rules are modules and the supposed "modularisation" that is said to result from this fact. Two

important possibilities for the nature of this relationship should be examined. Firstly, the architecture may "mechanically determine" the "modularisation". Secondly, the architecture may only facilitate a particular style of program organisation.

The first point of view has far reaching implications that we should make clear. According to this view, program modularity becomes the same sort of property as say, program portability. The portability (between the same compiler at least) of a program can be ensured by writing it in a high level language. This method of using a high level language to gain portability is effective irrespective of the organisation and content of the program. This is the crucial point, no design effort is involved in creating a portable program using this method. Similarly, according to the first view, no design effort would be required in creating a rule representation consisting of appropriate modules for modification if this kind of "modularity" is imposed by the production system architecture. Consequently, according to this view, it can be said that the use of the production system architecture solves the substantial problem of finding a "modularisation" that provides the modifiability necessary for incremental development.

Our argument, in opposition to the first view, is that any so called "modularity" (appropriate modules) that allows rules to be modified independently of each other (as occurs when extensions are made) is indeed a result of a particular organisation of knowledge. However, we would argue that the organisation of knowledge in a representation is not due to the use of a particular representation scheme. More specifically, we would argue that the mechanism of indirect calls among rule does not impose the sort of organisation of knowledge that allows extensions to be made to the representation.

Our argument consists of three parts. Firstly, there is an alternative to attributing the "modularity" (appropriate modules) of production system representations to the use of a particular architecture. This so called "modularity" can instead be attributed to the organisation of the knowledge represented. For instance, recall that the rules of NAVIGATE-1 could be said to be "separated" or "loosely coupled" in the sense meant by Davis and King. Rules representing new routes can, after all, be added as extensions to the rule-base. However, there is no a priori reason to attribute the "loose coupling" among rules to the mechanism of indirect calls. The rules in NAVIGATE-1 could equally well be "loosely coupled" because the routes, which the rules represent, are "loosely coupled" in NAVIGATE-1's organisation of knowledge. The same kind of explanation could equally well account for the "loose coupling" among the next-stage rules[23] of NAVIGATE-2 that was used to advantage in adding knowledge of new roads. Namely that within the organisation of knowledge in NAVIGATE-2, facts about the roads that leave a given town are independent of each other.

More generally, this problem of how to account for the performance of a program is one of the fundamental problems of artificial intelligence. Ritchie and Hanna (1984, p249,250) discuss this problem in connection with Lenat's (1976) AM program that "discovers" elementary concepts in mathematics.

> What has happened over recent years is that some attempt is now made to outline the principles which a program is supposed to implement. That is, the worker still constructs a complex program with impressive behaviour, but he also provides a statement of how it achieves this performance. Unfortunately, in some sense, the written "theory" may not correspond to the program in detail, but the writer avoids emphasising (or sometimes even conceals) this discrepancy. The "theory" is

---

[23] Recall that the next-stage rules of NAVIGATE-2 represent the roads that leave a given town.

> supposedly justified, or given empirical credibility, by the presence of the program (although the program may have been designed in a totally different way); hence the theory is not subjected to other forms of argument of examination.

This is not simply a problem of "loose" research standards, finding an appropriate theory that explains the behaviour of a complex program is a genuinely difficult problem. Furthermore, if should not be forgotten that merely because a program "works" this does not mean that there must be some general theory that explains the program's performance. A program may "work" simply, and only, because of the actions performed by each line of code in the specific sequence in which each line is executed. This sort of explanation is the sort that Marr (1981) calls a type 2 theory.

The second part of our argument is that by attributing the "modularity" (appropriate modules) of a rule representation to the organisation of knowledge we can account for differences in the way that the rules in different representations are independent of each other. For example, although NAVIGATE-1 and NAVIGATE-2 both make use of the mechanism of indirect calls, NAVIGATE-1 does not have the same organisation of knowledge that is present in NAVIGATE-2. Consequently, for a particular modification, (adding a new road, say) the rules of NAVIGATE-1 do not exhibit the independence exhibited by the rules of NAVIGATE-2.

The third part of our argument is that the "modularity" (appropriate modules) of rule representations can also be exhibited by representations not based on the production system architecture. For example, neither NAVIGATE-3 nor MICRO-ORGANISMS-3 are rule representations and hence do not make use of the mechanism of indirect calls among rules and yet both of these representations have the organisations of knowledge that make them each as extensible as their

respective rule-based counterparts. By attributing the "modularity" (appropriate modules) of a representation of knowledge to the organisation of that knowledge we can account for the "modularity" (appropriate modules) of representations not bases on the production system architecture.

The view that the production system architecture "mechanically determines" the "modularity" (appropriate modules) of a rule representation appears to be untenable. Consequently one might take the view that the architecture merely facilitates a particular style of representation.

There is some evidence to suggest that Davis and King take this latter view where they claim (quoted above) that the architecture

> 'emphasise[s] the decoupling of control flow from the writing of rules.'

This latter view is more plausible but correspondingly less interesting since the architecture is now "doing much less" for the knowledge engineer. For example, Davis and King (quoted above) state that

> 'Each rule is designed to be ideally, an independent chunk of knowledge ...'

The crucial question here is, 'who or what does the designing?' If it has to be done by the knowledge engineer then he is faced with a problem that can be arbitrarily difficult. Whereas, if the mere use of the production system architecture imposes the correct modular design then the knowledge engineer faces no problem at all in ensuring the modifiability of the rule-base.

Even though, however, the production system architecture does not automatically solve the modifiability problem for the knowledge engineer it may nevertheless contribute to its solution. Any

representation scheme that facilitates the construction of modules designed by the knowledge engineer will be a useful tool. As a tool of this kind, the rule scheme may or may not be useful. Its usefulness depends very much on the kind of modules that the knowledge engineer wishes to construct. Consequently to determine the usefulness of the rule scheme it is at least necessary to examine expert systems built with the scheme. We say 'it is at least necessary' because an examination of the finished product does not necessarily reveal the way in which it was constructed. The evaluation of a tool must in part involve the practical use of that tool. This last point not withstanding, we examine two rule-based expert systems in the next two chapters and to the extent that the examination of an expert system can reveal something of the usefulness of the tool used to construct it, we show that the rule representation scheme has crucial flaws.

CHAPTER SEVEN

REPRESENTATION, ORGANISATION AND MODIFICATION OF KNOWLEDGE: A PUFF/CENTAUR CASE STUDY

INTRODUCTION

Our aim in this chapter is to broaden the discussion of the concepts of an organisation of knowledge and extensibility. In particular we consider the role of these concepts in the practical business of modifying an expert system. For each of the expert systems PUFF and CENTAUR we describe their organisation and representation of knowledge. We then consider various modifications to each of the systems. Some of the modifications can be performed as extensions in one system but not in the other. Wherever a modification can be performed as an extension we show how the extension is made possible by a particular organisation of knowledge.

In this chapter we examine representations that clearly reveal the underlying organisation of knowledge and representations that obscure it. In considering modifications to obscure representations of knowledge, we illustrate the interaction between the concept of an extension and the more pragmatic aspects of knowledge-base modification.

As a starting point for the consideration of these pragmatic aspects, recall that for each example representation of knowledge that we considered we found that:

a) Those modifications that could be performed as extensions were very convenient to perform in practice.

b) Those modifications that could not be performed as extensions were arbitrarily convenient or difficult in practice.

Difficulty was measured by the amount of additional knowledge elicitation that was necessary. For an example of a convenient modification that could not be performed as an extension, refer back to the modification of MICRO-ORGANISMS-1. The addition of the rule R5.1 is a convenient modification but it cannot be performed as an extension to the rule-base. The modification is convenient because a little more knowledge from the expert allows the modification to be performed. Adding knowledge of a new road to NAVIGATE-1 is also an example of a modification that cannot be performed as an extension. In this case however, the modification is extremely difficult to perform because the expert must provide knowledge about every route in the system.

A given organisation of knowledge can be represented in many ways, some representations more perspicuous than others. The frame-like representation, NAVIGATE-3, for example, is more perspicuous than the production rule representation, NAVIGATE-2. Obviously, in any knowledge-base that is to be extensively modified, the representation of the organisation of knowledge should be as clear as possible.

DESCRIPTION OF PUFF AND CENTAUR

Introduction to PUFF

MYCIN, (Shortliffe, 1976) an early example of a rule-based expert system, became the "mold" for a number of subsequent expert systems through the use of the EMYCIN shell (van-Melle, 1979). One of these was the PUFF system (Kunz, et al, 1978) for interpreting pulmonary function test results. MYCIN and PUFF both achieved good problem solving performance but were unable to provide good explanations and were found to be difficult to modify (see below). Dissatisfaction with simple rule representations of knowledge led to the construction

of so called "second generation" systems. NEOMYCIN was constructed with the aim of overcoming the problems encountered with MYCIN and CENTAUR was constructed with a similar aim in relation to PUFF.

The problems with PUFF

Aikins (Aikins, 1983, p 167) describes a number of problems with the rule representation of knowledge in PUFF.

> Although PUFF's performance results were satisfactory, there were difficulties in working with the knowledge represented in the system: ... [one of which was] ... adding or modifying rules to represent additional knowledge, ...

Aikins directly attributes the difficulty of modifying PUFF to the use of the rule representation scheme.

> These characteristics of modularity and uniformity have also caused problems in rule-based systems. There are implicit groupings of rules that apply in specific situations and at certain stages of the consultation, but there is no explicit indexing to these rules by situations and by stages, ...
>
> In rule-based systems the modularity of the rules prevents organisation of the knowledge base in a way that would identify groupings of similar rules and would be useful in making modifications to sets of rules or in identifying interactions between rules. Adding or modifying rules may have indirect effects on other rules that are difficult to predict without these explicit groupings. The uniformity of structure often forces different types of knowledge to be represented using the same syntax, and therefore hides the function of the knowledge in the system. For example, rules that are written to control the invocation of other rules, to set default values, or to summarize data, are not distinguishable from rules used to infer new information.
>
> (ibid, p168,169).

In short, Aikins claims that the rule representation in PUFF, obscures the organisation of knowledge and in addition fails to encourage the sort of organisation in which related facts are grouped together, hence the rule representation makes PUFF difficult to modify.

CENTAUR, however, is not merely PUFF's organisation of knowledge represented more clearly. To overcome the problems mentioned above, Aikins (ibid) constructed a representation of knowledge organised

around pulmonary disease prototypes. Each disease prototype represents the typical pattern of pulmonary test results associated with that disease. Essentially the program produces a diagnosis by matching the patient's pulmonary test results with disease prototypes. Aikins (ibid, p199) describes the beneficial effects of CENTAUR's representation and organisation for knowledge acquisition and modification.

> CENTAUR's explicit representation and organisation of its knowledge-base into groups of knowledge dealing with prototypical situations facilitates knowledge acquisition. The prototypes represent blocks of basic knowledge, and include clearly defined 'hooks' for any additional rules necessary to elaborate upon this basic knowledge. The purpose of the knowledge attached to these hooks is explicit, making the effect of such modifications readily predictable.

Description of PUFF

PUFF's expertise lies in the domain of pulmonary function testing. In this domain, a number of tests are made of the patient's pulmonary function; including the total lung capacity, the rate at which air is expelled from the lungs, the vital capacity (total lung capacity less the residual capacity) and so on. There is also a measurement made of the rate at which carbon monoxide can diffuse into the blood. The measurements are normalised for the height, age and sex of the patient.

Provided with a set of such measurements it is PUFF's task to interpret them and provide a diagnosis of pulmonary disease if any. For example, a typical pattern of measurements for Obstructive Airways Disease is a residual volume of 140% of normal with a total lung capacity of normal or above and reduced flow rates during forced exhalation.

More than one lung disorder may coexist in a single patient with the effects of one disorder interacting with the effects of the other. Measurements are also made after the use of bronchodilating drugs. Any increase in the expiratory flow rates after the use of such drugs indicates that the progress of the disease is reversible.

PUFF is an EMYCIN system and therefore the domain knowledge is represented as rules in an exhaustive backward chaining system. The following is a typical PUFF rule which concludes on a subtype (Emphysema) of Obstructive Airways Disease (Kunz, ibid, p17).

```
if  the degree of Obstructive Airways Disease is
    greater than or equal to mild
and the degree of Diffusion Defect is
    greater than or equal to mild
and the total lung capacity as measured by body
    plethysmography is greater than or equal to mild
then
    The low diffusing capacity, in combination with
    obstruction and a high total lung capacity, would
    be consistent with a diagnosis of Emphysema.'
```

At a general level the behaviour of the system is controlled by the goal rule (Kunz, ibid, 1978, p9).

> Thus, to initiate system operation, a rule had to be written in the general form "IF degree of OAD [Obstructive Airways Disease] is known and ... THEN print and interpretation." Starting with this initiating rule, the system processed rules that defined the presence and severity of OAD; these rules in turn invoked rules to interpret actual physiological measurements. Once degree of OAD was finally determined, the system processed succeeding clauses of the initiating rule to diagnose the presence and severity of other diseases such as RLD [Restrictive Lung Disease].

The general form of the goal rule is shown below.

```
if   the degree of Obstructive Airways Disease is known
and  the degree of Restrictive Lung Disease is known
and  the degree of ...
                .
                .
                .
then
     print an interpretation
```

This rule, which performs a control function, is syntactically indistinguishable from rules that interpret test results, the former have been called control rules by Aikins (ibid, p186)

PUFF not only diagnoses the category of lung disease such as Obstructive Airways Disease or Restrictive Lung Disease but also the severity and subtype. For instance the category of Obstructive Airways Disease is divided according to four degrees of severity and three subtypes Bronchitis, Asthma and Emphysema.

PUFF always attempts to deduce the severity of Obstructive Airways Disease before attempting to deduce the subtype. Again a control rule (Aikins, ibid, p187) shown below

```
if 1) An attempt has been made to deduce the degree of
      Obstructive Airways Disease of the patient,
   2) An attempt has been made to deduce the subtype
      of Obstructive Airways Disease, and
   3) An attempt has been made to deduce the findings
      about the diagnosis of Obstructive Airways Disease
then
      It is definite(1.0) that there is an interpretation
      of potential Obstructive Airways Disease.
```

determines this part of PUFF's behaviour. Since, in this rule, the parameter `degree of Obstructive Airways Disease' occurs before the parameter `subtype of Obstructive Airways Disease' the rules that conclude on the former parameter are invoked before those that conclude on the latter.

PUFF's knowledge is essentially organised around a problem space of lung diseases. The space of diseases has a taxonomic structure since general categories of disease are subdivided into types of disease and these are again subdivided into subtypes of disease. For example in this taxonomy the category of Obstructive Airways Disease is first subdivided according to the severity and then each of these divisions is further subdivided according to the type i.e. Asthma,

Bronchitis or Emphysema.

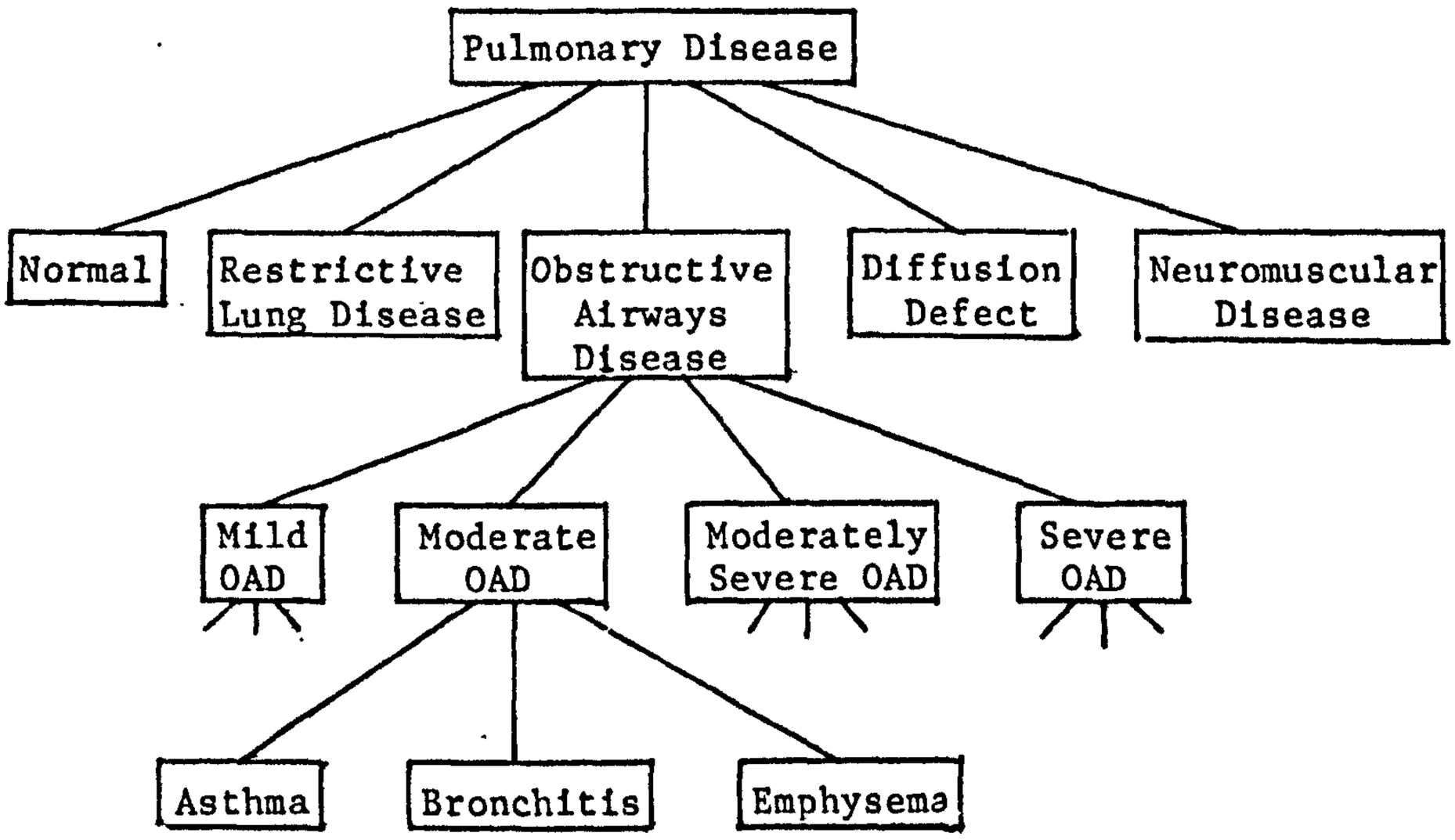


fig 7.1

Part of PUFF's taxonomy of diseases

PUFF diagnoses a patient's lung disorder by searching the taxonomy of diseases in a top-down manner.

At the second level of the taxonomy shown in fig 7.1, each of the diseases are tried in a fixed order (as determined by the goal rule, discussed above). This ordering is based on the a priori likelihood of the disease.

At the third level of the taxonomy the way in which the various categories of severity for Obstructive Airways Disease are examined is not determined by a control rule analogous to the goal rule. The different degrees of severity do not correspond to individual parameters. Instead the categories of severity are represented by different values of a single parameter, namely the parameter that appears in the following clause of the Obstructive Airways Disease control rule.

> 'An attempt has been made to deduce the degree of Obstructive Airways Disease of the patient, ...'

Whenever a number of goals in the search space appear as parameters in a control rule, necessarily in a particular order, the goals are pursued in the order in which they appear in that rule. However, since each degree of severity is represented as a value of a single parameter, this means that PUFF need not attempt to determine the degrees of severity in a fixed order.

For instance, if all degrees of severity of Obstructive Airways Disease are equally likely then there nay be no need to determine them in any particular order, indeed some rules conclude on more than one degree of severity. For example in the case of loss of alveolar capillary surface the following PUFF rule

```
if  the diffusing capacity for carbon monoxide is
    between zero and eighty percent of normal
then
    Low diffusing capacity indicates loss of
    alveolar capillary surface which is
        if the diffusing capacity is between
           seventy and eighty percent; mild
        if the diffusing capacity is between
           sixty and seventy percent; moderate
        if the diffusing capacity is between
           less than sixty percent; severe.
```

concludes on three degrees of severity. Here the three degrees of severity are essentially considered simultaneously.

Although representing goals as values of a single parameter enables PUFF to pursue these goals simultaneously, a fixed order strategy could be enforced without using a control rule analogous to the systems goal rule. For example it could be enforced by ordering the rules that conclude on the severity parameter so that all the rules that conclude that the severity is mild are tried before the rules that conclude the severity is moderate and so on. This tactic is illustrated in fig 7.2.

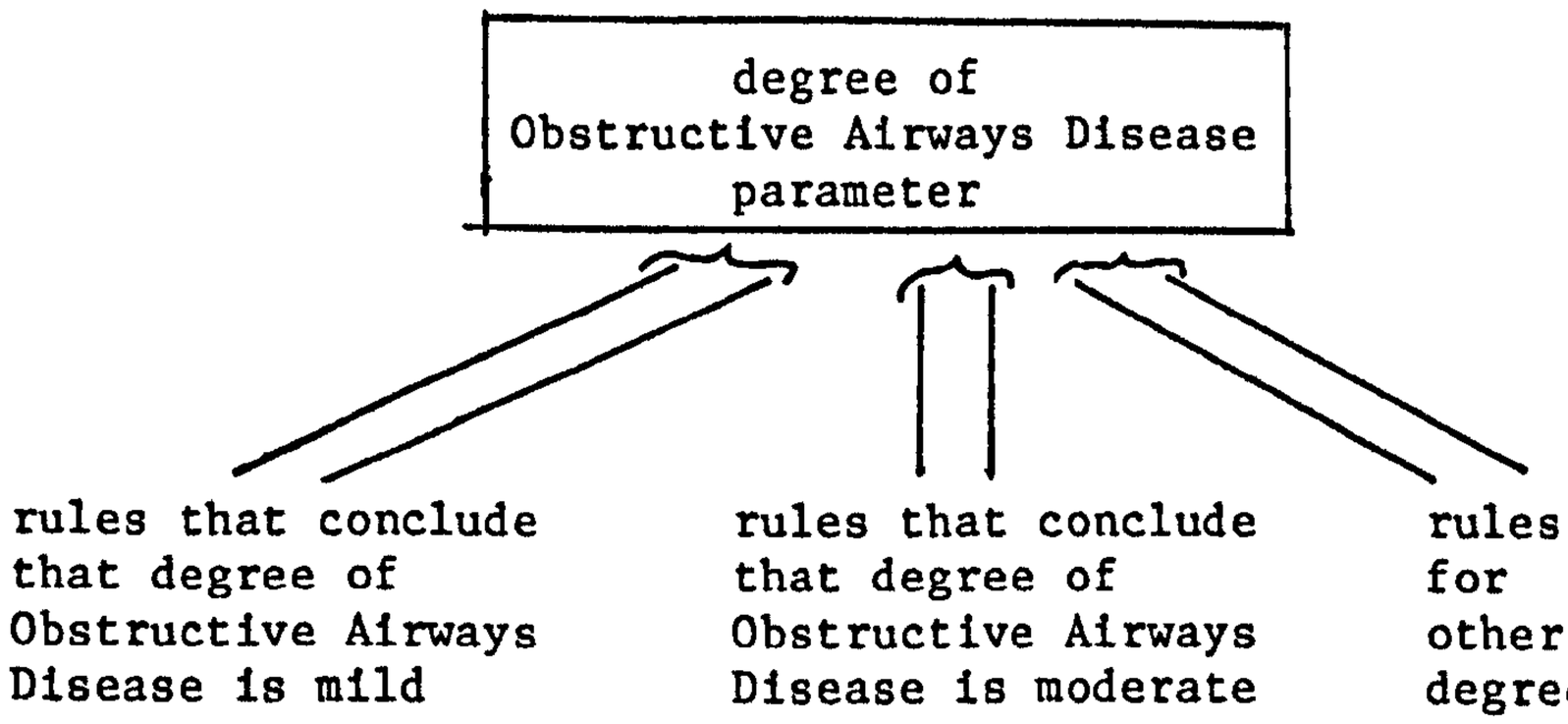


fig 7.2

Pursuing degrees of severity in a fixed order

Finally the subtypes of disease that appear at the bottom of the taxonomy are again values of a single parameter. Therefore the above discussion, describing the way in which the various degrees of severity are determined, also applies in this case.

It is important to appreciate the exhaustive and rigid nature of the strategy in PUFF's organisation of knowledge. The strategy is exhaustive because it fully pursues every category of disease on each level of the taxonomy. The strategy is rigid because the order in which the diseases at a level are pursued is fixed throughout a consultation. As we shall see later, this strategy is in marked contrast to CENTAUR's diagnostic strategy.

Description of CENTAUR

CENTAUR's representation of knowledge is, as Aikins describes it, explicitly organised in terms of pulmonary diseases. The knowledge of each pulmonary disease, since it consists essentially of the typical values of pulmonary function measurements associated with that disease, is called a disease prototype. General disease prototypes contain pointers to more specialised disease prototypes and thereby the network of prototypes models the taxonomy of lung diseases. At a

general level of description, CENTAUR performs a diagnosis by matching the patient's pulmonary test values against those associated with each disease, essentially searching the same taxonomy of diseases searched by PUFF.

Each disease prototype is represented as a frame. The characterising pulmonary measurements of a particular disease are represented as slots in the corresponding frame. Slots are themselves frame-like structures called components. For example, the Obstructive Airways Disease prototype has the following form (Aikins, ibid, p178).

Author: ,Date: ,Source:,	
Pointers: (degree mild-OAD) (degree moderate-OAD) ... (subtype Asthma) (subtype Emphysema) ...	
Hypothesis: There is Obstructive Airways Disease.	
If-confirmed: Deduce the degree of OAD. Deduce the subtype of OAD.	
Action: Deduce any findings associated with OAD. Print the findings associated with OAD.	
Fact-residual rules: rule 157, rule 158, ... Refinement rules: rule 036, rule 038, rule 039, ... Summary rules: rule 053, rule 054, rule 055, ...	
Components:	
Total lung capacity	Plausible values: >100 Importance measure: 4
Reversibility	Inference rules: rule 019 rule 020, rule 022, ... Importance measure: 0

fig 7.3

Part of the Obstructive Airways Disease prototype

Notice the pointers to prototypes representing various degrees and subtypes of Obstructive Airways Disease.

Some components are associated with a list of inference rules for obtaining a value for that component. For example, the reversibility of the Obstructive Airways prototype is shown in fig 7.3 to be associated with a list of inference rules. Whenever a value is

required for a component, the inference rules that are attached to the component are backward chained. Given a particular component, only the inference rules that conclude a value for that component are placed on the component's list of inference rules. In other words there are reversibility rules for Restrictive Lung Disease but these are not attached here. Instead they are attached to the reversibility slot in the Restrictive Lung Disease prototype.

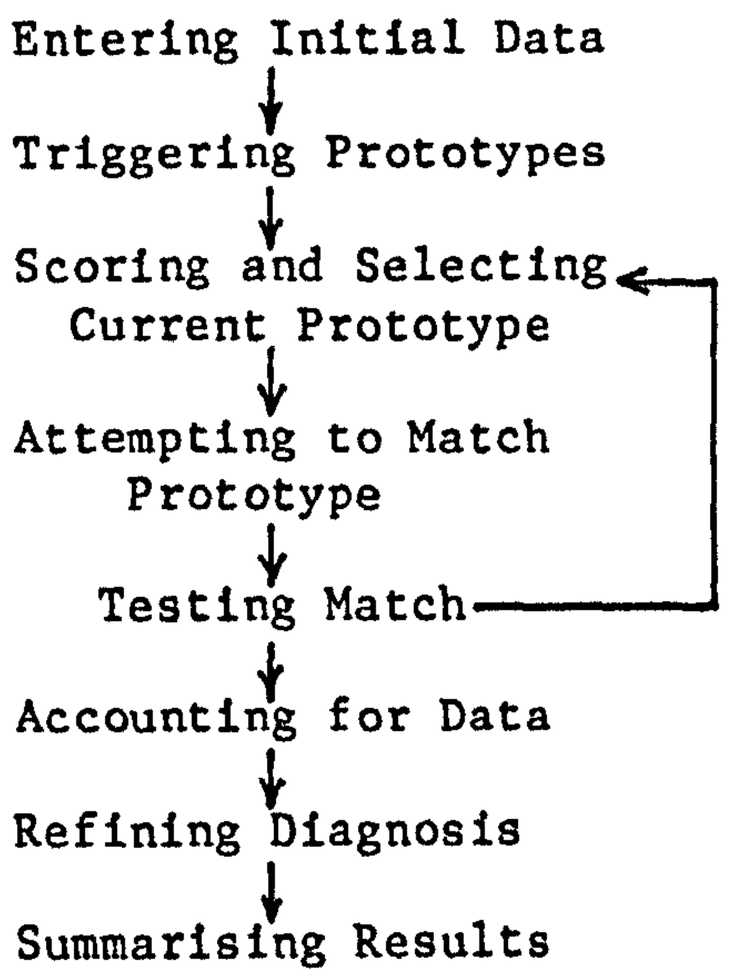
PUFF and CENTAUR use different diagnostic strategies. CENTAUR uses a small amount of initial data to suggest likely disease prototypes to pursue further rather than exhaustively considering each disease in turn in the way that PUFF does. For this purpose CENTAUR has a number of trigger rules which make use of initial data to suggest which disease category is most likely to be present in the patient. For example the following trigger rule

```
if  the diffusing capacity for carbon monoxide is
    less than 80% of normal
then
    1) Suggest Diffusion Defect with a certainty
       measure of 900,
    2) Suggest Emphysema with a certainty measure
       of 800, and
    3) Suggest Restrictive Lung Disease with a
       certainty measure of 800.
```

uses the low value of carbon monoxide diffusion to suggest three possible diseases with the certainty measures shown.

After all the initial data have been used a number of disease prototypes will have been suggested by the trigger rules. The user now has a choice in the strategy to be used for selecting the order in which these suggested disease categories should be pursued further. The user may elect to either pursue the most highly suggested disease prototype, attempt to eliminate the least highly suggested prototype or pursue them in a fixed order based on the a priori likelihood of each disease, i.e. the order in which this set of diseases would be

pursued by PUFF. An overview of the stages through which CENTAUR's consultation proceeds is shown below (Aikins, ibid, p190).



Once a prototype has been selected the prototype is matched against the patient's test results. Rules are used to interpret test results in terms of values that are characteristic of a particular disease. These rules may infer values that suggest new prototypes rather than the one currently being matched. In this situation the system backtracks, reorders the suggested disease prototypes according to their current certainty measures and pursues the new most highly suggested prototype.[24]

A prototype is successfully matched if a certain proportion of the expected features that characterise that disease are present in the patient. In addition the patient must not exhibit any features that are counter indicative of that disease. If a prototype is successfully matched, subtypes may need to be explored in which case the consultation returns to the third stage shown above i.e. the

---

[24] Aikins (ibid) does not describe how newly suggested prototypes are dealt with when the system is using the fixed strategy. Presumably backtracking does not occur unless the newly suggested prototype is not part of the set initially suggested by the trigger rules.

scoring and selecting of a current prototype. If there are no subtypes to explore then there may still be patient data unaccounted for by the hypothesis that the patient has that particular disease. Consequently there are a number of rules that are used here to attempt to account for any residual data. Finally refinement rules and summarising rules are used, respectively, to refine and summarise the interpretation.

COMPARISON OF PUFF WITH CENTAUR

Explicit context of rule application

The presence of explicit contexts for rule invocation is one of the differences between PUFF and CENTAUR which we shall now examine, considering particularly its effect on the modifiability of the two systems. Aikins (ibid, p179) describes this difference between PUFF and CENTAUR as follows.

> One distinction between the organisation of rule knowledge in PUFF and CENTAUR is apparent here. In PUFF, the Inference Rules are grouped according to parameters in the rule conclusions. When a value is needed for a parameter, the corresponding list of rules is retrieved, and the rules on that list are used in an attempt to infer the value.
>
> In CENTAUR, these lists of rules are associated with the component in the more narrowly defined context of the prototype. This not only makes explicit the context in which each rule is applied, but it also results in a smaller set of applicable rules when a value is needed for a component. For example, ... in CENTAUR, there is a reversibility component for each disease prototype where reversibility of disease is a consideration. Only that subset of rules are useful for obtaining a value for reversibility when a particular disease is being considered are (sic) listed with the reversibility component in that disease prototype.

In PUFF, a rule concluding on the reversibility of Obstructive Airways Disease would have the following form:

```
if  the patient is suffering from Obstructive Airways Disease
and finding-1 is present
and finding-2 is present
      .
      .
      .
then
    conclude that the reversibility of the disease is ...
```

In contrast the corresponding CENTAUR rule would have the form:

```
if  finding-1 is present
and finding-2 is present
      .
      .
      .
then
    conclude that the reversibility of the
    Obstructive Airways Disease is ...
prototype: Obstructive Airways Disease.
```

Notice that the CENTAUR rule is labelled with the name of the prototype on which it concludes.

Explicit context: extensibility

We will show that the difference between PUFF and CENTAUR described above has no affect on the extensibility of the corresponding representations, but that there is a difference in the way in which various parts of the representation are "accessed" by a knowledge engineer. The different ways in PUFF and CENTAUR of retrieving the facts to be modified may in practice influence the modifiability of each representation.

To examine the modifiability of this part of PUFF and CENTAUR consider how new knowledge about the reversibility of Lung Disease would be added to each of the two systems. We consider two cases. In the first, the new knowledge is an association between findings and the reversibility of a particular disease. In the second case, the new knowledge is an association between findings and the reversibility of lung disease in general.

Considering the first case, assume for example that the knowledge to be added relates a number of findings to a value for the reversibility of Obstructive Airways Disease. In PUFF this knowledge is expressed as a rule of the form

```
if  the patient is suffering from Obstructive Airways Disease
and finding-1 is present
and finding-2 is present
      .
      .
      .
then
    conclude that the reversibility of the lung disease is ...
```

This rule can be simply added to the rule-base as an extension. The rule is retrieved whenever a value for the reversibility of the patient's lung disease is required, but it is not applied without the satisfaction of the first antecedent, i.e. the patient must be suffering from Obstructive Airways Disease.

To add this same knowledge of the reversibility of the patient's Obstructive Airways Disease to CENTAUR, the knowledge engineer adds a rule of the following form:

```
if  finding-1 is present
and finding-2 is present
      .
      .
      .
then conclude that the reversibility of the
     Obstructive Airways Disease is ...
prototype: Obstructive Airways Disease.
```

to the reversibility slot of the Obstructive Airways Disease prototype. In both cases the knowledge can be added as an extension to the system so in this sense the representations are equally modifiable.

Consider now the second case, the case in which the knowledge to be added is an association between some findings and the reversibility of lung disease in general rather than any one

particular disease. This knowledge is easily added to PUFF as a rule

of the form

```
if  finding-1 is present
and finding-2 is present
      .
      .
      .
then
    conclude that the reversibility of the lung disease is ...
```

a rule that concludes on the reversibility parameter. The addition of this knowledge to CENTAUR is also straightforward. The rule would simply be attached to the reversibility slot in a prototype that represents the general category 'lung disease'.

Such a prototype however, is absent in CENTAUR, presumably because such a general category of disease is not useful to CENTAUR for the interpretation task. The absence of this prototype means that the knowledge of lung disease reversibility cannot be added as an extension. However, in order to compare like with like we can assume that CENTAUR has a general Lung Disease prototype.

Notice that, an expedient way of ensuring that CENTAUR can make use of the knowledge in the general Lung Disease rule is to add the rule to all the disease prototypes. However, the resulting redundancy would create problems if that knowledge was to require modification but more importantly it is contrary to the design principles of CENTAUR's organisation of knowledge. If the context in which inference rules are to be applied is to be represented explicitly then the knowledge about the reversibility of lung disease should be added to the Lung Disease prototype.

To sum up our examination so far of the use of explicit contexts for rule invocation, we may say that, in terms of knowledge-base extensibility, CENTAUR's separation of rules into groups according to

the context in which they apply does not seem to offer any advantages over the way in which rules are grouped in PUFF.

Explicit context: clarity of representation

The organisation of knowledge about the reversibility of lung diseases is essentially the same in PUFF and CENTAUR. In each case the knowledge is organised as an association between various findings and the reversibility of a particular lung disease. The representation of the knowledge however, is different in each system. In PUFF the particular lung disease is represented as an antecedent of the rule, in CENTAUR the particular disease is represented by attaching the rule to the prototype corresponding to that disease.

This is not to imply, however, that the difference in representation is of no practical consequence. Any knowledge engineer that is to modify an expert system must understand how it operates and how to obtain access to those parts of the knowledge that require modification. For example, consider how knowledge about the reversibility of Obstructive Airways Disease might be retrieved for examination in PUFF and in CENTAUR.

In PUFF it is straightforward to retrieve all the reversibility rules in the system since they are the list of rules that conclude on the reversibility parameter. To examine the knowledge concerning the reversibility of a particular disease such as Obstructive Airways Disease the knowledge engineer must first access all the reversibility rules and then pick out all the rules that specify the presence of Obstructive Airways Disease in their premises.

The need to search through rules can be avoided with the use of a sophisticated rule editor such as TEIRESIAS (Davis, 1979, 1982). This kind of editor allows a knowledge engineer to easily access all those

rules that conclude on a particular parameter and contain a particular antecedent. Such a rule editor could therefore be used in PUFF to access all the rules that conclude on the reversibility parameter and mention a particular disease such as Obstructive Airways Disease in their premises. Indeed an arbitrarily sophisticated "interface" program could be used to extract parts of the most obscure representations. In this situation the knowledge engineer would hardly be working with the representation but with the interface provided by the program, hence it would be doubtful if it could be claimed that it was the representation that was being modified. Consequently, our aim here is to assess the accessibility of knowledge in PUFF's rule-base in the absence of such programs.

In CENTAUR, the rules that conclude on the reversibility of Obstructive Airways Disease are attached to the reversibility slot of the Obstructive Airways Disease prototype. Consequently this set of rules can be examined without searching through all the reversibility rules of every disease.

The explicit representation of the context of a rule essentially means that PUFF and CENTAUR differ in the "access path" to particular rules.

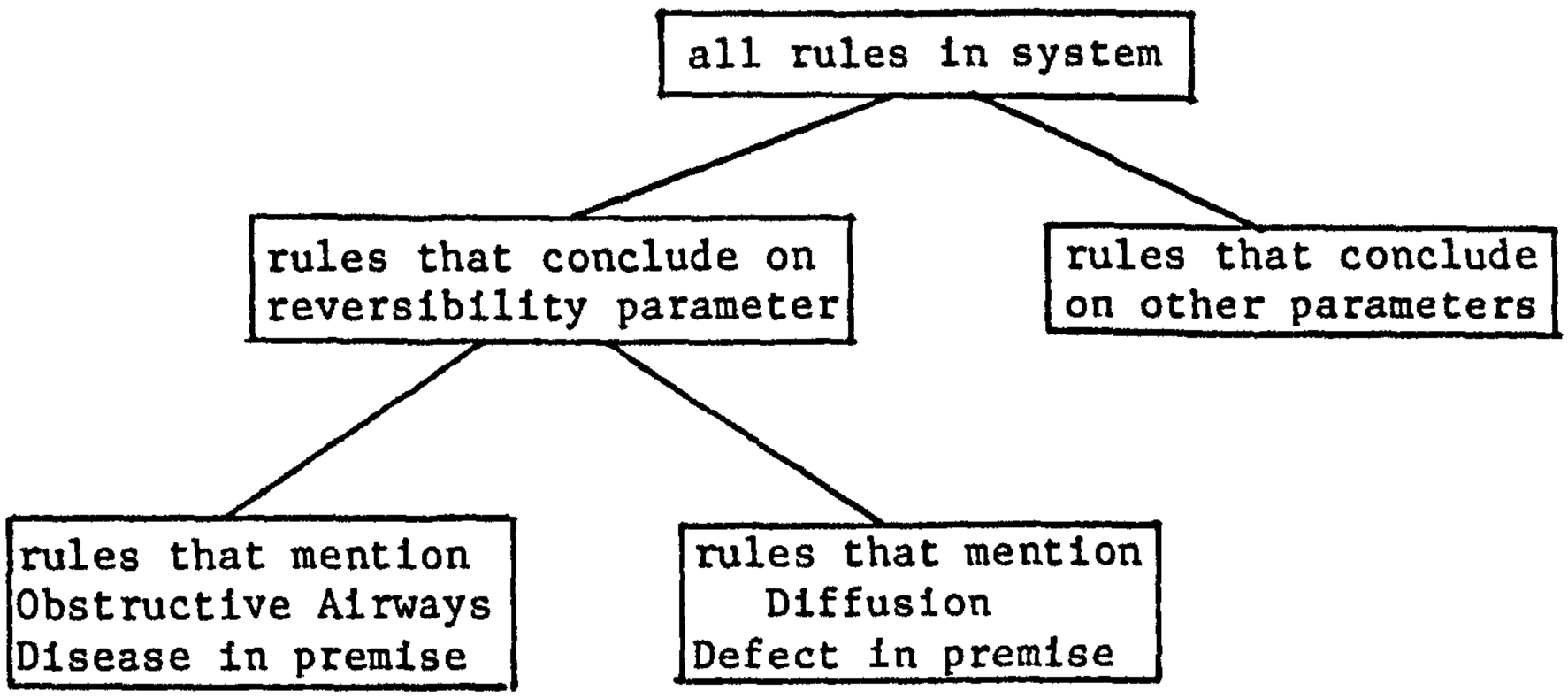


fig 7.4 part (a)

The partitioning of rules in PUFF

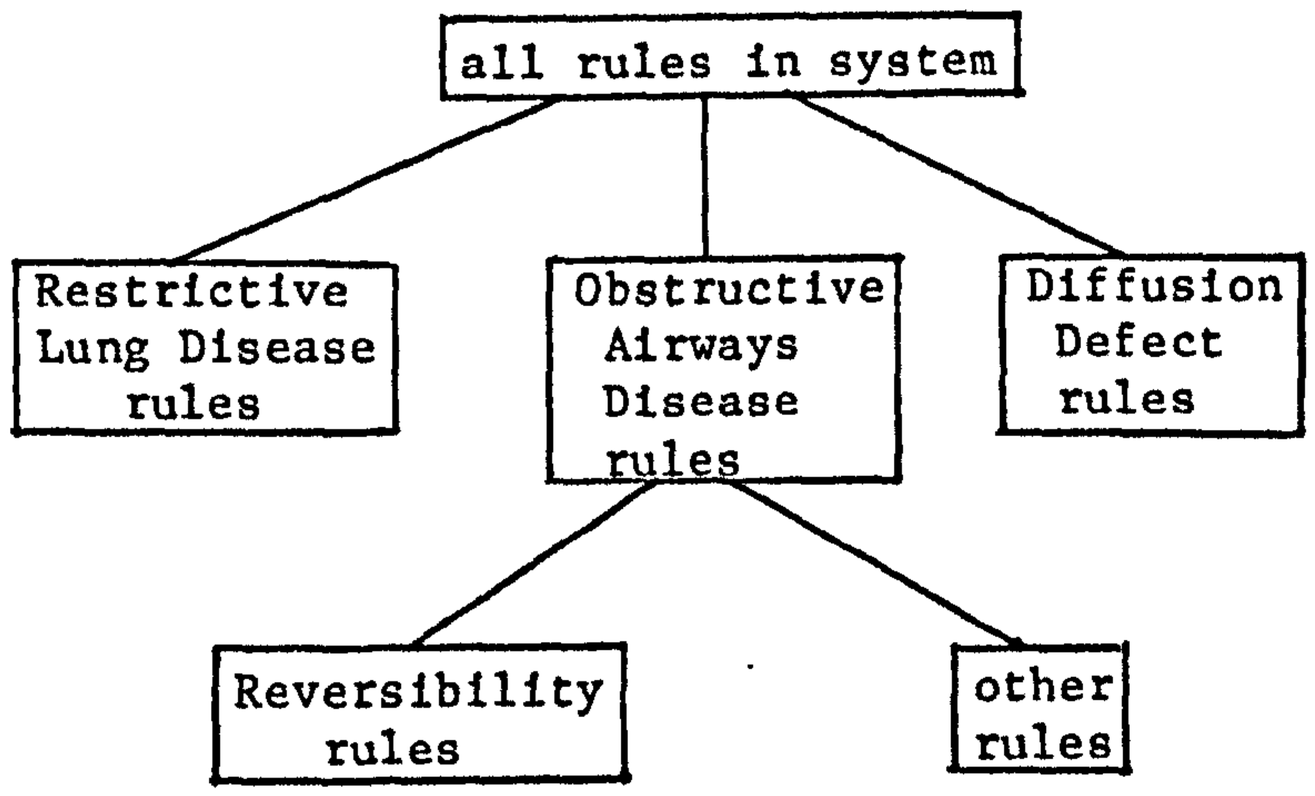


fig 7.4 part (b)

The partitioning of rules in CENTAUR

The difference between PUFF and CENTAUR in "access paths" to particular rules does not affect the extensibility of the knowledge of reversibility.

However, in general the difference in "access paths" can affect the ease with which a modification is performed since a modification that requires a change to a single rule might be difficult to perform if the rule to be changed cannot be found. Other modifications might require that all the rules of a particular kind be examined. In this case the modification is made simpler if all the relevant rules are

grouped together in the knowledge-base.

Diagnostic strategy

In this section we compare the modifiability of the diagnostic strategies in PUFF and in CENTAUR. We show that PUFF and CENTAUR have different organisations of diagnostic strategy, CENTAUR's strategy essentially subsumes that of PUFF. Consequently, there are modifications that are can be performed as extensions in CENTAUR but not in PUFF. Although PUFF's strategy is simpler than CENTAUR's it's representation is obscure. Hence PUFF's strategy is difficult to comprehend from a study of its representation. In CENTAUR, the use of frames and attached tasks, is invaluable in clearly representing a complex strategy. At the end of this chapter, we illustrate this point by showing the difficulty of clearly representing CENTAUR's strategy in PUFF's EMYCIN representation language.

One of the problems with PUFF's representation of diagnostic strategy described by Aikins, is that control rules are indistinguishable from other rules and hence the strategic knowledge component of a rule-base can be difficult to find. Furthermore control knowledge is "implicit" in the order in which rules are invoked to conclude on a parameter. Aikins' approach in trying to solve these problems is to represent strategic knowledge as tasks (represented as lisp functions) attached to control slots in a prototype. Aikins' (ibid, p165) describes the advantages of CENTAUR's representation of control as follows.

> In CENTAUR, control knowledge is represented within each prototype, allowing context-specific control, and separating control knowledge from other knowledge in the system. ... At the highest level in CENTAUR, the 'typical consultation' is represented as a prototype with the various stages of the consultation listed in control slots. Not only does this explicitly define the consultation's control process, but it also allows the flexibility of adding or omitting a stage, and

of more easily experimenting with control modifications.

In the following section we investigate how easily one can experiment with control modifications in CENTAUR and in PUFF.

Diagnostic strategy: extensibility

For example, suppose that it is required to modify the order in which each system pursues the possible disease interpretations[25]. In PUFF, this part of the diagnostic strategy is controlled by the order of the corresponding antecedents of the goal rule. If for example the first two antecedents of the goal rules were reversed to become

```
if   the degree of Restrictive Lung Disease is known
and  the degree of Obstructive Airways Disease is known
and  the degree of ...
           .
           .
           .
then print an interpretation
```

then PUFF would pursue Restrictive Lung Disease before Obstructive Airways Disease and so on. Clearly PUFF's strategy can be modified to pursue the diseases at this level in the taxonomy in any order.

A direct comparison of the modifiability of the consultation strategies of CENTAUR and PUFF is inappropriate since, as we have seen, there are differences between them. However if we assume for the moment that the user always selects the fixed order strategy option then CENTAUR only makes use of its trigger rules to suggest possible disease prototypes but not the order in which they should be followed up. In this situation CENTAUR's strategy can be regarded as being essentially the same as PUFF's. With this proviso in mind, consider now how the same strategy modification would be performed in CENTAUR.

---

[25] That is the diseases that appear on the second level of the taxonomy in fig 7.1 above.

In CENTAUR the overall consultation strategy is determined by tasks in the consultation prototype. This prototype also contains a list of pulmonary diseases (the hypothesis list) ordered according to the a priori likelihood of each disease. The control task attached to the 'if confirmed' slot of the consultation prototype attempts to establish each of the disease prototypes in the order given in the hypothesis list. Therefore to change the order in which CENTAUR pursues disease interpretations, it is simply a matter of changing the order in which disease prototypes appear in this list.

In the case discussed above the way in which CENTAUR is modified is analogous to the way in which PUFF is modified. In PUFF modifications are made to the order of the antecedents of the goal rule whereas in CENTAUR modifications are made to the order of the disease prototypes in the hypothesis list. This kind of modification can be performed as an extension in PUFF and in CENTAUR.

In the same way, the order in which PUFF considers the subcategories of a disease can be easily modified by changing the order of the corresponding antecedents in the control rule for that disease. For example, if the antecedents of the Obstructive Airways Disease control rule were reversed to become

```
if 1) An attempt has been made to deduce the subtype
      of Obstructive Airways Disease,
   2) An attempt has been made to deduce the degree of
      Obstructive Airways Disease of the patient, and
   3) An attempt has been made to deduce the findings
      about the diagnosis of Obstructive Airways Disease
then
      It is definite(1.0) that there is an interpretation
      of potential Obstructive Airways Disease.
```

then PUFF would consider the subtypes of Obstructive Airways Disease i.e. Asthma, Bronchitis and Emphysema before considering the severity of the disease. PUFF's diagnostic strategy is now a top-down search of the taxonomy shown in fig 7.5

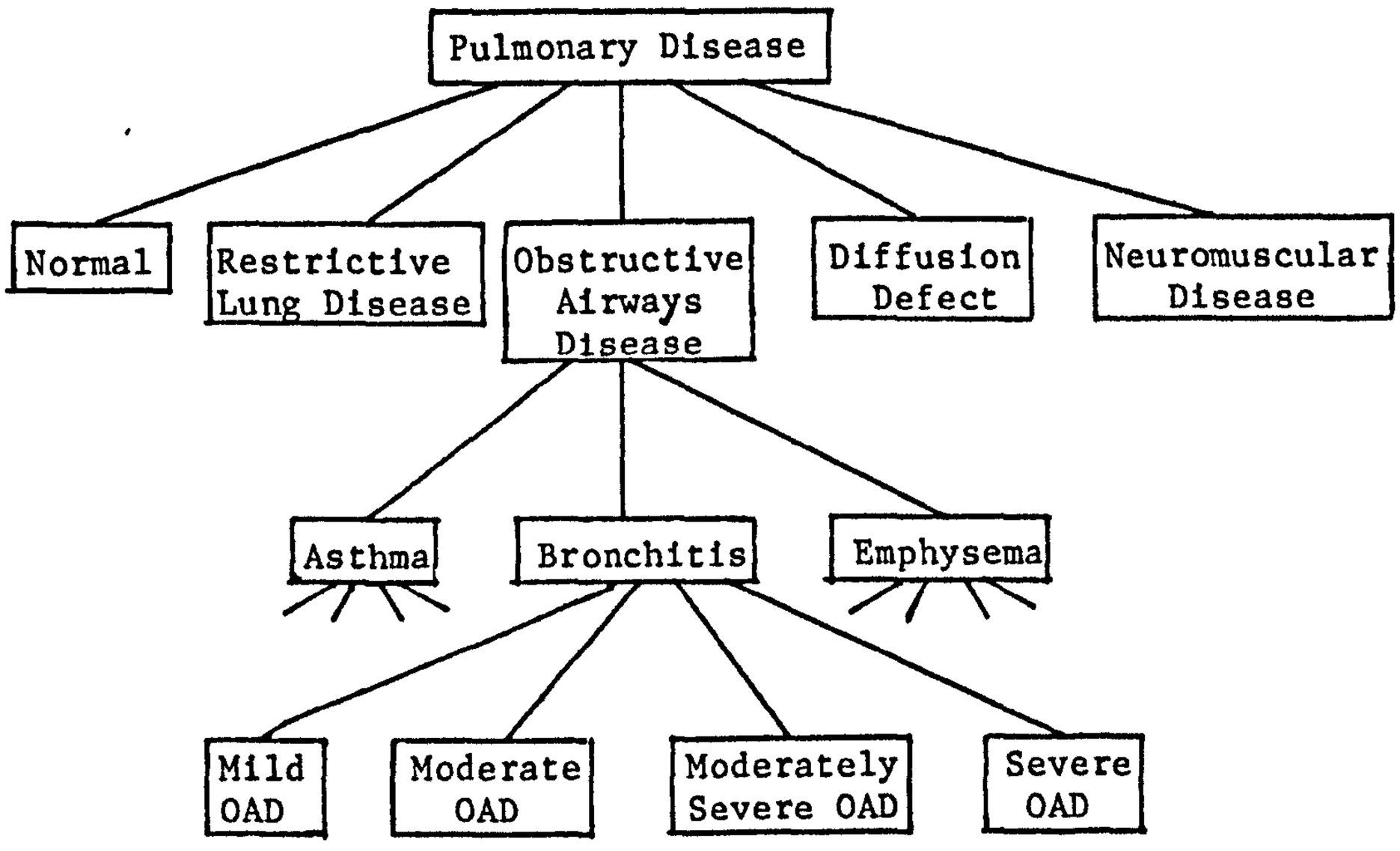


fig 7.5

The categories of Obstructive Airways Disease divided according to subtype then severity

In CENTAUR, the order in which the different kinds of sub category of a disease are pursued is determined by control slots in that disease prototypes. For example, the Obstructive Airways Disease prototype contains the control slot

If-confirmed: Deduce the degree of OAD.
Deduce the subtype of OAD.

with the values shown, specifying that the degree of Obstructive Airways Disease should be deduced before its subtype. Reversing the order of the two tasks in this control slot would reverse the order in which they are executed.

Again, the modifications to each system are analogous in that the antecedents of the Obstructive Airways Disease control rule correspond to control tasks in the 'if-confirmed' slot of the consultation prototype. The modifications in PUFF and CENTAUR described above are extensions in both systems and in this sense the representations of

knowledge are both equally convenient to modify.

Recall that in PUFF, the order in which disease hypotheses are pursued can be determined by rule order. Figure 7.2 (above) showed how in PUFF, it is possible to ensure that the various degrees of severity of Obstructive Airways Disease are pursued in a particular order, namely by arranging that all the rules that conclude that the severity is mild (say) are ordered ahead of the rules that conclude that the severity is moderate (say), and so on. Given that this method is used to represent strategic knowledge, assume that we wish to modify the order in which PUFF's pursues the degree of severity of Obstructive Airways Disease. For the sake of example, assume that PUFF should pursue the hypothesis that the degree of severity is moderate before the hypothesis that the degree is mild.

To perform this modification we must modify the order of the rules that conclude on the 'degree of Obstructive Airways Disease' parameter. Notice that although this modification requires changing the order of a number of rules the modification is nonetheless an extension. The reason for this is that we are assuming that the knowledge engineer working with PUFF is aware of the use of rule order to represent strategic knowledge.[26] In this situation, the ordering of many rules is used to represent the single fact that the various degrees of severity should be pursued in a particular order. That the modification requires changes to the order of many rules is simply a result of the lack of conciseness in PUFF's representation of this part of its diagnostic strategy.

---

[26] In the next section we argue that this is an obscure method of representing strategic knowledge, nevertheless a knowledge engineer could understand this method.

In CENTAUR the modification just described is also an extension, it is performed by simply modifying the order of the pointers to prototypes that represent the various degrees of severity. The following slot of the Obstructive Airways Disease prototype (see fig 7.3 above)

Pointers: (degree mild-OAD) (degree moderate-OAD) ...
(subtype Asthma) (subtype Emphysema) ...

would be modified to

Pointers: (degree moderate-OAD) (degree mild-OAD)...
(subtype Asthma) (subtype Emphysema) ...

We may summarise our examination so far of the modifiability of the diagnostic strategy in PUFF and CENTAUR as follows. There are a group of modifications that consist essentially of alterations to the order in which diseases or disease categories are pursued. These modifications are all extensions in both systems and, to that extent, are conveniently performed in both PUFF and CENTAUR.

Diagnostic strategy: clarity of representation

Again, the difference, in practice, between the modifications to each system is in the way that the parts of each representation that need modification are accessed. On first sight the rule-base appears as a uniform unstructured collection of rules. In reality the rules conclude about objects in a fairly well structured problem space, the most important of which are the various pulmonary diseases. The uniform appearance of the rules therefore tends to obscure this structure and the way in which it is searched, the diagnostic strategy. Control rules such as the goal rule are indistinguishable from other rules in the system and Aikins (ibid) points out the obscurity of this method of representing strategic knowledge. The modifications to PUFF are only straightforward provided that the

knowledge engineer understands that there is such a thing as a goal rule and the relationship between the order of the antecedents in that rule and the order in which disease interpretations are pursued. The use of rule order is a particularly obscure method of representing strategic knowledge.

In CENTAUR the structure of the pulmonary disease search space is represented as a network of frames. In each frame there are designated slots for the representation of diagnostic strategy to search this space, in particular the representation of the order for pursuing the "top level" diseases is represented in a control slot in the consultation prototype. To the extent that the diagnostic knowledge represented in this way is easier to find, CENTAUR must be considered more convenient to modify.

Triggering associations

In all the modifications discussed so far we have kept to an overall strategy that is exhaustive and top down, diseases at each level pursued in a predetermined order. These modifications are all of a restricted kind and don't fully test the modifiability of CENTAUR's organisation of knowledge.

CENTAUR is able to depart from the fixed strategy used in PUFF because of the use of triggering associations that suggest possible pulmonary diseases on the basis of a few initial data values. The collection of further, more extensive, data is therefore guided in that data that is not relevant to the suggested diseases will not be examined.[27]

---

[27] CENTAUR is able to backtrack if further data shows that the diseases suggested by the triggering associations are inappropriate.

This aspect of CENTAUR's organisation of knowledge, namely the separation between initial data to be used to suggest diseases and subsequent data to be used in confirming a diseases, is markedly different to that present in PUFF. In PUFF, no distinction is made between data that could be useful at the beginning of the consultation to suggest likely diseases and data that could then be used to confirm a disease. Whenever PUFF pursues a disease interpretation it attempts to apply all the rules that conclude on that interpretation.

Triggering associations: extensibility

As an example of a modification that one might attempt to make to PUFF and CENTAUR, consider how each system could be modified to make use of a new triggering association in its strategy.

A triggering association is an association between findings in the patient and diseases that he or she may be suffering. A particular finding may suggest a number of diseases, some more so than others. for example CENTAUR uses the following trigger rule (Aikins, ibid, p182).

```
if  the diffusing capacity for carbon monoxide
    is less than 80% of normal
then
    1) suggest Diffusion Defect with certainty
       measure of 900
    2) suggest Emphysema with certainty
       measure of 800, and
    3) suggest Restrictive Lung Disease with
       certainty measure of 800
```

Assume that the triggering association that we wish to add to CENTAUR and PUFF is an association between breathlessness after climbing stairs and Obstructive Airways Disease. This association can be directly represented in a rule form,

```
if  the patient suffers breathlessness after
    climbing stairs
then
    suggest Obstructive Airways Disease with
    a certainty measure of 700.
```

This rule can be added to CENTAUR by simply attaching it to the list of trigger rules. The rule will be applied appropriately since the task that suggests possible prototypes to pursue on the basis of the initial data applies all the trigger rules on the trigger rule list. Clearly the addition of triggering associations can be performed as extensions in CENTAUR but not so in PUFF as we now show.

The trigger rule that we wish to add to PUFF, shown above, concludes on the presence of Obstructive Airways Disease. Consequently if this rule is added to PUFF's rule-base then it will be invoked (by the backward chaining interpreter) whenever PUFF is seeking to establish the presence of Obstructive Airways Disease. This is not how we intended the trigger rule to be used. The rule should have been used to suggest the possibility of Obstructive Airways Disease rather than be used to confirm that disease. Furthermore, since the choice of disease to pursue is determined by the goal rule, PUFF continues to relentlessly test for the presence of each disease in turn, irrespective of the conclusions made by trigger rules such as the one we are attempting to add to the rule-base.

In PUFF's organisation of knowledge, it "doesn't make sense" to add triggering associations. PUFF does not have the sort of diagnostic strategy that can make use of triggering associations and therefore they cannot be added as extensions. PUFF's strategy is to search a taxonomy of lung diseases exhaustively, in a fixed order. As part of this strategy, whenever a particular disease hypothesis is being pursued the strategy invokes all the knowledge that bears on that hypothesis. The strategy therefore treats triggering

associations just as it does all other knowledge that bears on a disease hypothesis, i.e. as knowledge to be used in confirming a disease interpretation. In order for triggering associations to be used to suggest likely hypotheses to pursue, modifications are necessary to PUFF's diagnostic strategy, hence the addition of triggering associations cannot be performed as extensions.

We have now shown that there are modifications that can be performed as extensions in CENTAUR but not in PUFF but this is not surprising since the systems have different organisations of knowledge, especially diagnostic strategy.

Triggering associations: clarity of representation

Given the complexity of CENTAUR's diagnostic strategy it is important to represent it clearly, and for this purpose CENTAUR's use of frames and attached tasks is invaluable. We can demonstrate the value of using a clear representation by sketching how CENTAUR's diagnostic strategy might be represented in PUFF. Incidently, since triggering associations cannot be added to PUFF until PUFF is modified to represent CENTAUR's diagnostic strategy (or similar strategy) then we are at the same time demonstrating the difficulty of adding triggering associations to PUFF.

It should be noted that the EMYCIN rule language is perhaps not the most suitable instantiation of the rule scheme for our purpose. Later we will show that the sort of rule language used in PROSPECTOR (Reboh, 1981) is more suitable for a rule representation of CENTAUR's diagnostic strategy.

If PUFF is to pursue a variety of disease hypotheses in more than one order then PUFF requires a number of "pseudo" goal rules[28], each

[28] An EMYCIN system only ever has one goal rule; here we mean rules

of which, specifying a number of disease hypotheses to be pursued in a prescribed order. Consequently, PUFF would be able to choose an order for searching the taxonomy by selecting a particular "pseudo" goal rule.

The initial data and the triggering associations determine the way in which the taxonomy of disease hypotheses should be searched. Hence, on the basis of the results obtained from the initial data, a suitable "pseudo" goal rule must be selected. One way to achieve this is to use "pseudo" trigger rules[29] that conclude on a parameter 'suggested-disease-hypotheses-order', say, rather than trigger rules that conclude on particular disease hypotheses. The value of this parameter could then be used to select amongst the variety of "pseudo" goal rules, each one of which representing a particular order for searching the taxonomy.

More specifically, the "pseudo" trigger rules would have the form

```
if  finding-1 is present
and finding-2 is present
and   .
      .
      .
then the suggested-disease-hypotheses-order should be
     Obstructive Airways Disease
     followed by
     Restrictive Lung Disease
     followed by
          .
          .
          .
```

---

that perform the same function as PUFF's present goal rule, i.e. controlling the order in which disease hypotheses are pursued.

[29] Strictly, the organisation of knowledge we are representing in PUFF is not CENTAUR's diagnostic strategy because we are not using triggering associations from initial data to disease hypotheses. To represent genuine triggering associations in EMYCIN would require modification of the rule interpreter, allowing the non-exhaustive invocation of rules.

The value of the suggested-disease-hypotheses-order parameter could then be used to determine the order in which PUFF pursues disease hypotheses by including "pseudo" goal rules of the form

```
if  the suggested-disease-hypotheses-order is
    Obstructive Airways Disease, followed by
    Restrictive Lung Disease, followed by ...
and the degree of Obstructive Airways Disease is known
and the degree of Restrictive Lung Disease is known
and the degree of ...
                .
                .
                .
then there is an interpretation of the patient's
     test results.
```

One such "pseudo" goal rule is required for every possible ordered subset of disease hypotheses that are suggested by the trigger rules.

Since the "trigger rules" should be applied before the taxonomy is searched, a "real" goal rule is required of the form

```
if  the suggested-disease-hypotheses-order is known
and there is an interpretation for the patient's test results
then
    print that interpretation
```

The first antecedent of this goal rule would invoke the "pseudo" trigger rules and the second antecedent would invoke the "pseudo" goal rules that directly control the order in which the taxonomy is searched.

Although we have not yet represented CENTAUR's entire diagnostic strategy[30] the representation sketched out so far demonstrates how obscure an EMYCIN representation of CENTAUR's diagnostic strategy would be.

Although the EMYCIN rule language is particularly unsuitable for the representation of the strategy we have been considering this

---

[30] We have not represented CENTAUR's ability to backtrack in the case that the disease hypotheses suggested by the initial data are not established.

should not be taken as a criticism of the rule scheme in general[31]. For example, the sort of "rule language" used in the PROSPECTOR system would allow a far more perspicuous representation of CENTAUR's diagnostic strategy than is possible in EMYCIN. This kind of "rule language" has two important features that would greatly simplify the task of representing a strategy such as CENTAUR's. Firstly, such a "rule language" allows the data directed invocation of rules. This facility can be used to implement trigger rules that are indeed "data-directed" rather than using a backward chaining interpreter to exhaustively attempt to apply all the "pseudo" trigger rules. Furthermore, PROSPECTOR's rule interpreter does not pursue the values of parameters exhaustively. Consequently, we may conveniently implement the ability to backtrack in the event that the hypotheses suggested by the trigger rules prove incorrect.

Figure 7.6 shows how triggering associations would be represented as data directed rules and how other data/hypothesis associations, used for confirming the presence of a particular disease, would be represented as goal directed rules.

---

[31] Recall the distinction, made in chapter two, between a representation scheme and a representation language.

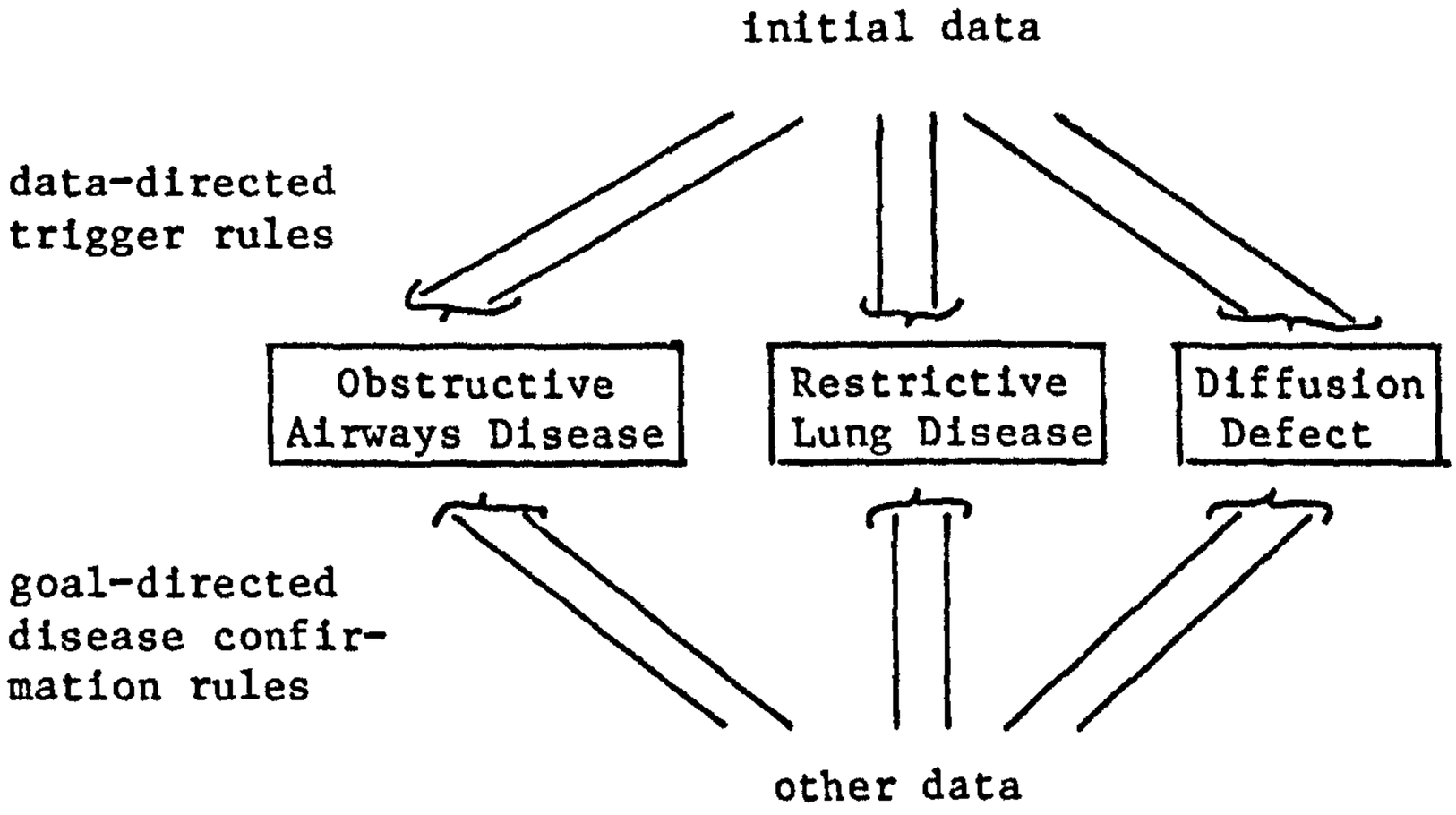


fig 7.6

The representation of CENTAUR's diagnostic strategy
in a suitable rule language.

The above representation would function as follows. The initial data is presented to the trigger rules, these rules update the certainty measures of each disease hypothesis and the rule interpreter attempts to pursue the hypothesis that is most highly suggested by using the goal directed rules.

Furthermore, PROSPECTOR's "rule language" includes the facility to define a group rules that should be pursued exhaustively before pursuing any other rules. This facility can be used to ensure that PUFF does not "wander" among various hypotheses (constantly changing the hypothesis under examination) in its pursuit of an interpretation.

In practice the presence of structuring features in a "rule language" can allow it to cope with many of the problems described in this chapter. The question arises as to whether such a "rule language" is not more properly thought of as an instantiation of a "frame/rule" scheme (the type of scheme advocated by Aikins) rather than as an instantiation of a rule scheme.

In this chapter we have considered the problem of modifying the large and complex organisations of knowledge that are used in expert systems. We have discussed two important factors that largely determine the convenience of a modification, the organisation of knowledge and the clarity with which that organisation is represented.

We have shown that modifications that can be performed as extensions need not necessarily be convenient in practice since there is the problem of "uncovering" the organisation of knowledge in an obscure representation. The "dual" problem of "uncovering" the organisation of knowledge from an obscure representation is the problem of representing an organisation of knowledge in an unsuitable representation language. We demonstrated the difficulty of representing a more "focused" diagnostic strategy in a rule language limited to an exhaustive backward chaining strategy.

More generally, we would argue that the choice of representation scheme for representing knowledge should be based on the organisation of the knowledge to be represented. At first sight, this appears as an obvious enough point and in keeping with the conventional view. For example Davis and King (1977) appear to be making a very similar point.

> Program designers have found that PSs[production systems] easily model problems in some domains, but are awkward for others.
>
> ibid, p307.

However, Davis and King construe 'domain' in a very different way from our concept of an organisation of knowledge. For Davis and King, a domain is characterised in very "loose" and general terms, as can be seen from their characterisation of the kind of domains that are suitable for production systems.

PSs therefore appear to be useful where it is important to

> detect and deal with a large number of independent states, in a system which requires a broad scope of attention, and the capability of reacting quickly to small changes. In addition, where knowledge of the problem domain falls naturally into a sequence of independent 'recognize-act' pairs, PSs offer a convenient formalism for structuring and expressing that knowledge.
>
> ibid, p309.

Davis and King implicitly recognise the existence of domains where 'knowledge of the problem domain falls naturally into a sequence of independent ''recognise act'' pairs, ...'. In contrast to this view, we have seen that even within a single domain, pulmonary function test interpretation, it is possible to use alternative problem solving methods with significantly different organisations of knowledge. This would suggest that the issue of suitable and unsuitable domains for the use of the rule scheme is more involved than Davis and King suggest. As a result, the rule representation scheme may not be as widely applicable as Davis and King seem to suggest.

CHAPTER EIGHT

REPRESENTATION, ORGANISATION AND MODIFICATION OF KNOWLEDGE: A MYCIN/NEOMYCIN CASE STUDY

INTRODUCTION

It could be argued that the conclusions drawn in the previous chapter are based only on the comparison of two systems and as such their general validity might be in doubt. In this chapter we essentially repeat the investigation performed in the previous chapter but with two other expert systems. This investigation produces the same conclusions that were drawn previously. This chapter therefore serves to give more general validity to the analysis of the previous chapter.

The general form of this chapter is the same as that of the previous chapter. The particular organisations of knowledge are different of course, NEOMYCIN's diagnostic strategy is yet more sophisticated than CENTAUR's. As a result we consider a different set of modification problems to those considered previously.

DESCRIPTION OF MYCIN AND NEOMYCIN

The problems with MYCIN

The MYCIN system (Shortliffe, 1976) is designed to help a physician diagnose the cause of a patient's infection and recommend a suitable therapy. One of the aims of the MYCIN project was to be able to use the knowledge-base for teaching students. It was assumed that students could profit considerably from the "wealth" of knowledge contained in the rule-base. To exploit this knowledge, Clancey (1979) built a teaching program called GUIDON with access to MYCIN's rule-base. However, it was found (Clancey, 1983) that GUIDON could not

explain important aspects of MYCIN's behaviour.

> GUIDON cannot fully articulate MYCIN's problem solving approach because the structure of the search space and strategy for traversing it are implicit in the ordering of rule concepts.
>
> ... the expert's diagnostic approach and understanding of rules have not been explicitly represented. ...
>
> The rules are more then simple associations between data and hypothesis. Sometimes clause order counts for everything (and the order can mean different things), and some rules are present for effect, to control the invocation of others. The uniformity of the representation obscures these various functions of clauses and rules.
>
> ibid, p216.

If an expert system is to be used for teaching then it must be able to provide good explanations. However, good explanations are also valuable during the development of an expert system. Although it may be readily apparent during testing that the knowledge-base contains an error, without a good explanation facility the source of the error will be difficult to find.

> ... the representation has serious limitations: people other than the original rule authors find it difficult to modify the rule set, ...
>
> ibid, p215.

> A knowledge-base is like a traditional program in that maintaining it require having a good understanding of the underlying design. That is, you need to know how the parts of the knowledge-base are expected to interact in problem solving. Depending on the representation, this includes knowing ... whether rule clauses can be reordered, ... However, problems encountered in understanding traditional programs--poorly-structured code, implicit side-effects, and inadequate documentation--carry over to knowledge-based programming and naturally limit the capabilities of explanation programs.
>
> Clancey, 1983b, p74.

Since MYCIN's explanations are inadequate because of the obscure way in which knowledge is represented in it's rule representation, Clancey's solution is to separate the different kinds of knowledge that appear in MYCIN's rules according to an epistemological

framework. The epistemological framework characterises a variety of distinct forms of diagnostic knowledge.

> The epistemology that evolved from attempts to reconfigure MYCIN's rules is NEOMYCIN's etiological taxonomy, multiple disease process hierarchies, data that trigger hypothesis, etc., plus the domain-independent task hierarchy of meta-rules.
>
> ibid, p248.

The idea being that, by unpacking MYCIN's rules into the forms of knowledge from this framework they will be more understandable.

One important aspect of this epistemological framework is that it contains a "domain-independent" strategy component. Whenever a strategy is used that is valid in more than one domain, it is represented in general terms. This approach eliminates much redundant repetition in the representation of "local" instantiations of a general strategy. Furthermore, there is the possibility of explaining and teaching a single general strategy rather than its many domain dependent instantiations.

Clancey (1983) describes in detail the value of his approach for producing better explanations, but only briefly describes the advantages for modifiability and system development.

> A rule-base is built and extended like any other program. Extensive documentation and a well-structured design are essential, as in any engineering endeavor. The framework of knowledge types and purposes that we have described would constitute a `typed' rule language that could make it easier for an expert to organise his thoughts.
>
> ibid, p242.

> The knowledge-base is easier to construct because the expert needn't specify every situation in which a given fact or relation should be used. New facts and relations are added in a simple way; the abstract meta-rules explicitly state how the relations will be used.
>
> Clancey, 1983b, p76.

'The explicit design is easier to debug and modify.'

ibid, p77.

In this chapter we investigate in more depth the modifiability advantages of Clancey's approach in the construction of NEOMYCIN.

Description of MYCIN

MYCIN's expertise lies in the domain of infectious disease diagnosis and treatment, namely the diagnosis and treatment of meningitis and bacteremia. To perform its task, MYCIN makes use of two broad categories of data. Firstly, cultures are taken to identify organisms possibly causing the infection. The cultures may take a day or two to grow but then various features of the organism such as the stain, morphology and growth conformation are visible under a microscope. For example, the following MYCIN rule (ibid, p71) makes use of laboratory data.

```
if  1) the stain of the organism is grampos
    2) the morphology of the organism is coccus
    3) the growth conformation of the organisms is clumps
then
    there is suggestive evidence(.7) that the identity
    of the organism is staphylococcus.
```

This laboratory data can provide the conclusive evidence for the identity of the offending organism, however some patients require treatment before such data is available and for various reasons[32] there is the possibility that the organisms identified from the culture are not the organisms that are responsible for the infection.

Consequently, there is a second broad category of data used by MYCIN which consists essentially of circumstantial evidence. Patients that have suffered burns for example would be predisposed to infection by certain kinds of organisms and may be immediately treated for these, even though no laboratory data is yet available. In addition, clinicians pursuing a cautious strategy might prescribe drugs to

---

[32] Namely there is a risk that cultures may become contaminated.

cover for the organisms normally associated with burns even though the available laboratory data showed no sign of these organisms.

Many different factors can constitute circumstantial evidence, such as the drugs a patient is receiving, visits to geographic regions, injuries sustained and so on. For example, the following MYCIN rule encodes (amongst other things) a relationship between the taking of a particular drug and infection by various organisms.

```
if   1) the infection that requires therapy is meningitis
and  2) only circumstantial evidence is available for this case
and  3) the type of meningitis is bacterial
and  4) the patient is receiving corticosteroids
then
     there is evidence that the organisms which might be causing
     the infection are e.coli (.4), klebsiella-pneumoniae(.2),
     or pseudomonas-aeruginosa (.1)
```

fig 8.1

The "steroids" rule

At the most general level of description the behaviour of MYCIN is a follows. MYCIN firstly attempts to identify the organisms responsible for an infection and secondly formulates a treatment. In attempting to identify the offending organisms, MYCIN will firstly check whether laboratory data is available and if so, make use of this data. Secondly, MYCIN considers circumstantial evidence.

This part of MYCIN's behaviour is a direct consequence of the order of the antecedent conditions in the goal rule (ibid, p92), figure 8.2 below.

```
if:  1) there is an organism which requires therapy

 and 2) consideration has been given to the possible existence of
        additional organisms requiring therapy, even though they
        have not actually been recovered from any current cultures
 then:
      do the following
      1) compile the list of possible therapies which based upon
         sensitivity data may be effective against the organisms
         requiring treatment,
  and 2) determine the best therapy recommendations from
         the compiled list
otherwise:
        indicate that the patient does not require therapy.
```

fig 8.2

MYCIN's goal rule

MYCIN finds out if there is an organism which requires therapy (the first antecedent above) by invoking rules that make inferences from laboratory data. MYCIN's exhaustive strategy of rule invocation ensures that all the available laboratory data will have been assessed before the second antecedent becomes an active goal. The rules that bear on the second antecedent make inferences from circumstantial evidence.

Here, as in PUFF, the order of the antecedents of the goal rule is crucially important. If these two antecedents were reversed then MYCIN would collect all the circumstantial evidence before at all considering laboratory data. This behaviour would be unacceptable in situations where laboratory data was available. Furthermore the significance placed upon circumstantial evidence depends to an extent on whether or not there is any laboratory data available and if so, its quality. If the laboratory data has already been used to arrive at some very definite conclusions about the identity of the offending organisms the circumstantial evidence has little weight. Some rules are only applicable when there is no laboratory evidence at all, (for example the rule in figure 8.1 above) and therefore these rules cannot

be invoked before laboratory data is considered.

At a more detailed level of description MYCIN's behaviour is determined by the rules that are invoked by the goal rule. Many of these rules share a particular pattern of antecedents. For example, comparing the rule below with the rule of figure 8.1 above, we notice that the first three antecedents of each rule are identical.

```
if 1) the infection is meningitis
   2) only circumstantial evidence is available
   3) the subtype of meningitis is bacterial
   4) the patient is at least 17 years old
   5) the patient is an alcoholic
then
    there is evidence that that the organisms which might be causing
    the infection are diplococcus-pneumoniae (.3) or e.coli (.2).
```

fig 8.3

The "alcoholic" rule

The first three conditions of these rules appear in forty of MYCIN's rules similar to the "alcoholic" rule shown above, Clancey (1983a, p136). These rules could well be called the "circumstantial bacterial meningitis rules" since they all consider circumstantial evidence for bacterial meningitis. The repetition of these conditions throughout a number of rules reflects the fact that bacterial meningitis is an important intermediate category used in the diagnosis of meningitis. Viral and fungal meningitis are other such categories.

The various categories of meningitis form a taxonomy in which meningitis is the topmost general category and the various causative organisms appear at the bottom of the taxonomy. Part of this taxonomy is shown in figure 8.4.

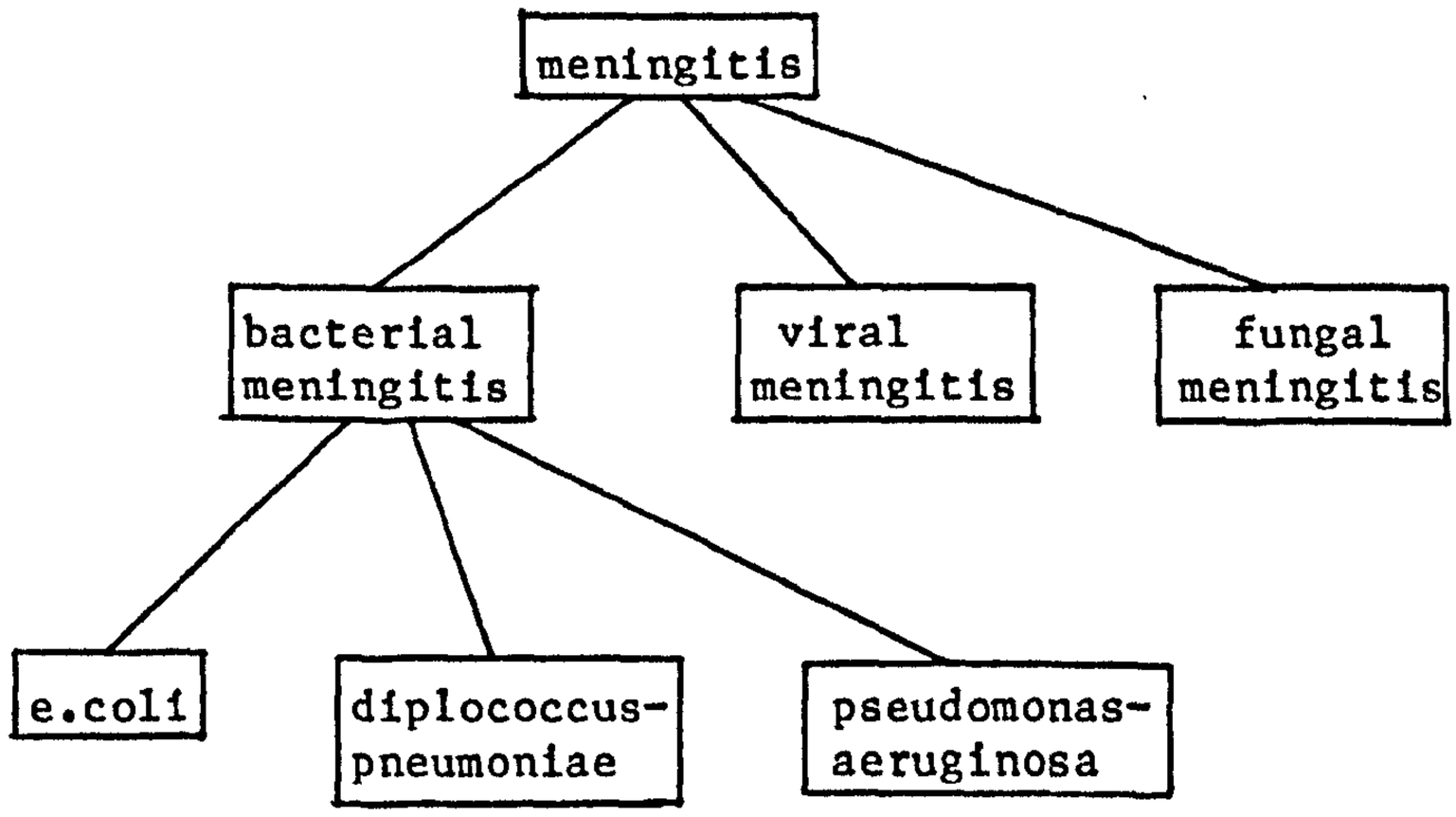


fig 8.4

Portion of the etiological taxonomy for meningitis

Notice the order of the two antecedents

if 1) the infection is meningitis
   2) ...
   3) the subtype of meningitis is bacterial

that are common to all of the "circumstantial bacterial meningitis rules". Since meningitis occurs before bacterial meningitis, MYCIN attempts to establish that the infection is meningitis before bacterial meningitis is considered. This behaviour is in accordance with a top down search of the etiological taxonomy.

In general, MYCIN diagnoses a meningitis infection is by proceeding top down through the etiological taxonomy to arrive at the category of organisms that are causing the infection. At any particular stage in the diagnosis, the category of diseases so far established can be refined to include more specialised category by using associations between problem features and categories in the taxonomy.

The etiological taxonomy and strategy to traverse it, top down refinement, are the areas of domain expertise that

determine the choice of conditions in these rules and the order in which they appear. However, MYCIN cannot explain the concept of a taxonomy and a strategy to search it by simply displaying rules containing particular conditions in a particular order.

The remaining fourth and fifth antecedents of the rule in figure 8.3

4) the patient is at least 17 years old
5) the patient is an alcoholic

represent two different kinds of knowledge. Firstly, alcoholics sometimes inhale their own vomit allowing infectious organisms access to the normally sterile lung tissue. This fact accounts for the presence of the fifth antecedent.

The presence of fourth antecedent does not mean that the rule does not apply to young alcoholics under seventeen years of age, it does. Rather it is the clinician's experience that patients under seventeen years of age are not alcoholics. Therefore if a patient is under that age[33] there is no need for MYCIN to ask the user if the patient is an alcoholic, as is required for the satisfaction of the fifth antecedent.

The use of these so called 'screening factors' helps to reduce the number of questions that MYCIN will need to ask. This is done for human engineering reasons since busy clinicians are annoyed if they are asked redundant questions.

Screening antecedents are concerned with the process of data acquisition rather than the process of making diagnostic inferences about the likely cause of the infection. The distinction between these two areas of knowledge is not easily visible in MYCIN's rule-

---

[33] The age of the patient is a fact that MYCIN collects early on in the consultation.

base. Consequently, MYCIN cannot teach the general strategy of screening for data.

Furthermore, Clancey (1983) has found that a large proportion of MYCIN's rules conform to the following pattern

```
if   in an appropriate context
and  screening factors (if any) are satisfied
and  problem feature is present
then
     make a conclusion or take some action
```

Again, MYCIN cannot explicitly teach this pattern to the student, the system can only present examples of it.

It is clear that at the organisation of knowledge level that MYCIN and PUFF have much in common. The two systems both search a taxonomy in an exhaustive top down manner. Furthermore, since both of the systems use the same representation language (EMYCIN), it is not surprising that identical representation methods for various kinds of knowledge are used in each. As an example, notice that MYCIN and PUFF use the identical method of ordering the antecedents in the goal rule to represent the same diagnostic strategy.

Description of NEOMYCIN

Broadly, NEOMYCIN consists of two mutually dependent parts, a representation of domain knowledge and a representation of a domain-independent diagnostic strategy. These parts are shown in figure 8.5 below.

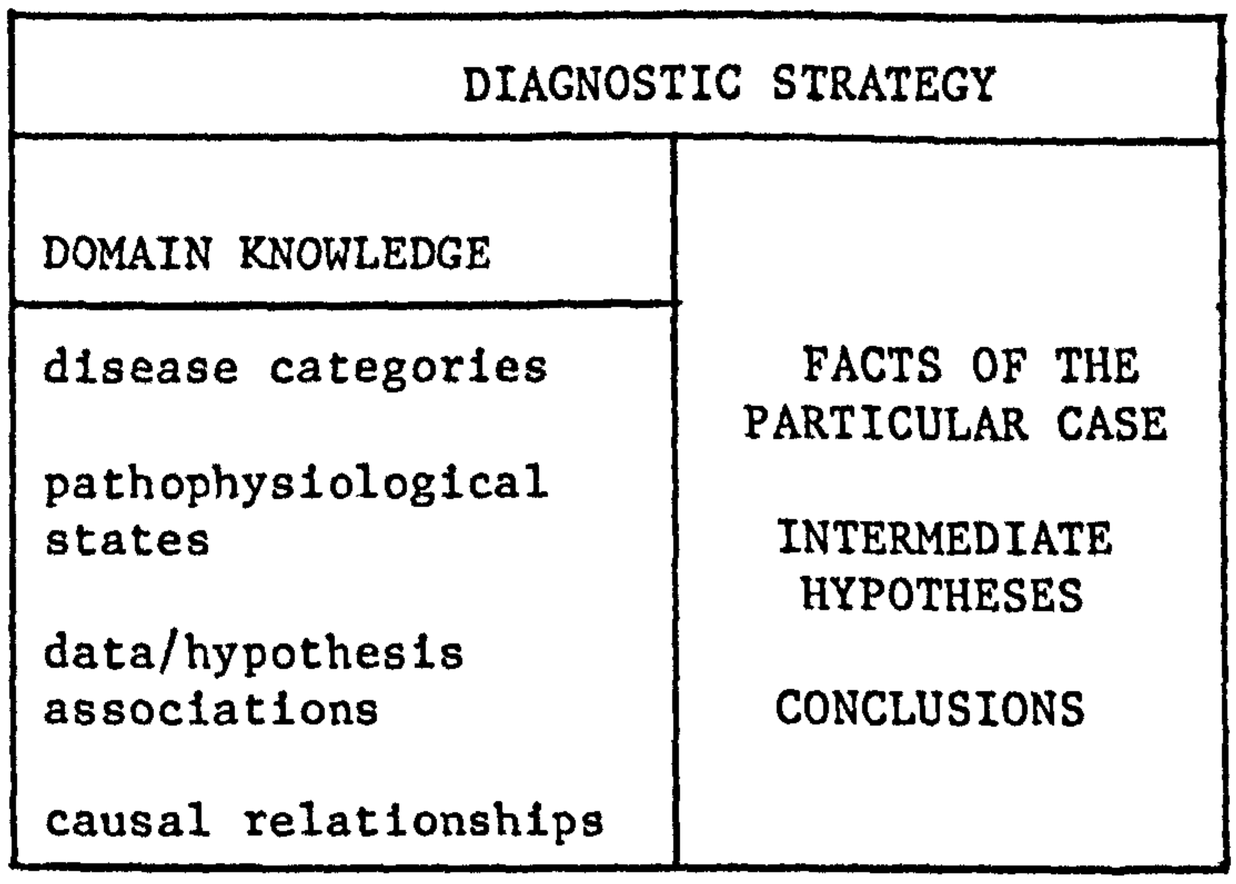


fig 8.5

The forms of knowledge used in NEOMYCIN

The domain knowledge consists of knowledge about disease categories (the etiological taxonomy), pathophysiological states, for example immunosuppressed, associations between the patient data and disease categories and states and knowledge of causal relationships between pathophysiological states. The diagnostic strategy is responsible for the application of the domain knowledge to a particular case. Below, we firstly describe the domain specific knowledge and secondly, the strategic knowledge.

NEOMYCIN contains a representation of a taxonomy of disease categories part of which is shown in figure 8.6 below, adapted from (Clancey and Letsinger, 1981, p834).

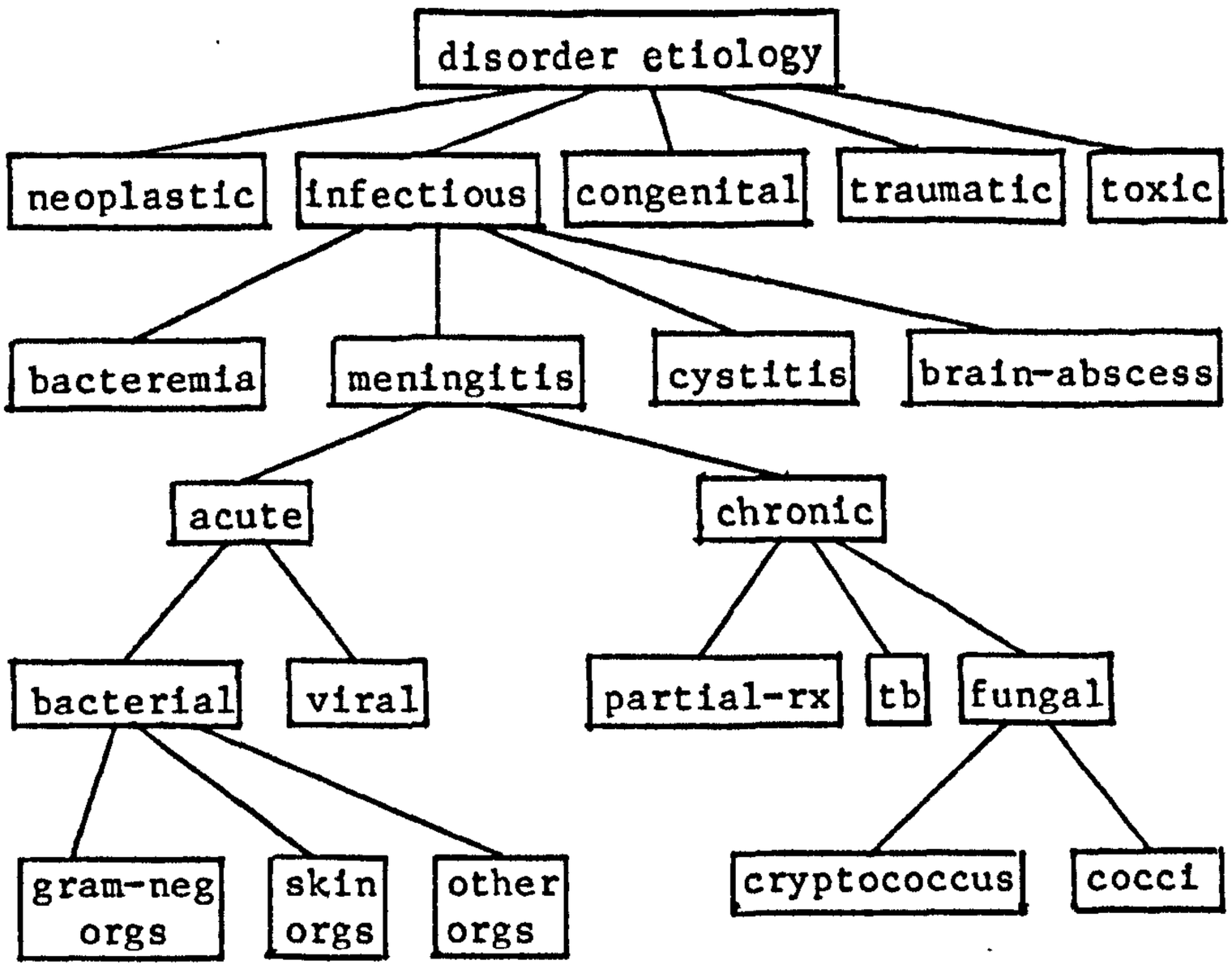


fig 8.6

Part of NEOMYCIN's etiological taxonomy

Some of the categories in the etiological taxonomy are associated with frames that represent knowledge about the process of that category of disease. This process knowledge considers a disease in terms of the location, extent and progression of symptoms. Since different diseases have different processes, the process knowledge can be used to distinguish between categories of disease. If for example, a patient has suffered from a headache and a stiff neck for thirteen days then this is evidence for the presence of chronic rather than acute meningitis.

Some of the categories of the etiological taxonomy represent pathophysical states that are linked together into a causal network. The causal network is represented by rules of the form

```
if  state-A is present
then
    state-B is present (cf).
```

where state-B causes the state-A. In medicine, observable states are usually caused by unobservable states. For this reason causal rules

are forward chained, since if state-A is observed and it is known that state-B causes state-A then it is possible to infer that state-B is probably present. A certainty factor associated with the rule indicates that the presence of state-A is not conclusive evidence for the presence of state-B[34]. A part of the causal network is shown in figure 8.7 below, adapted from (ibid, p834).

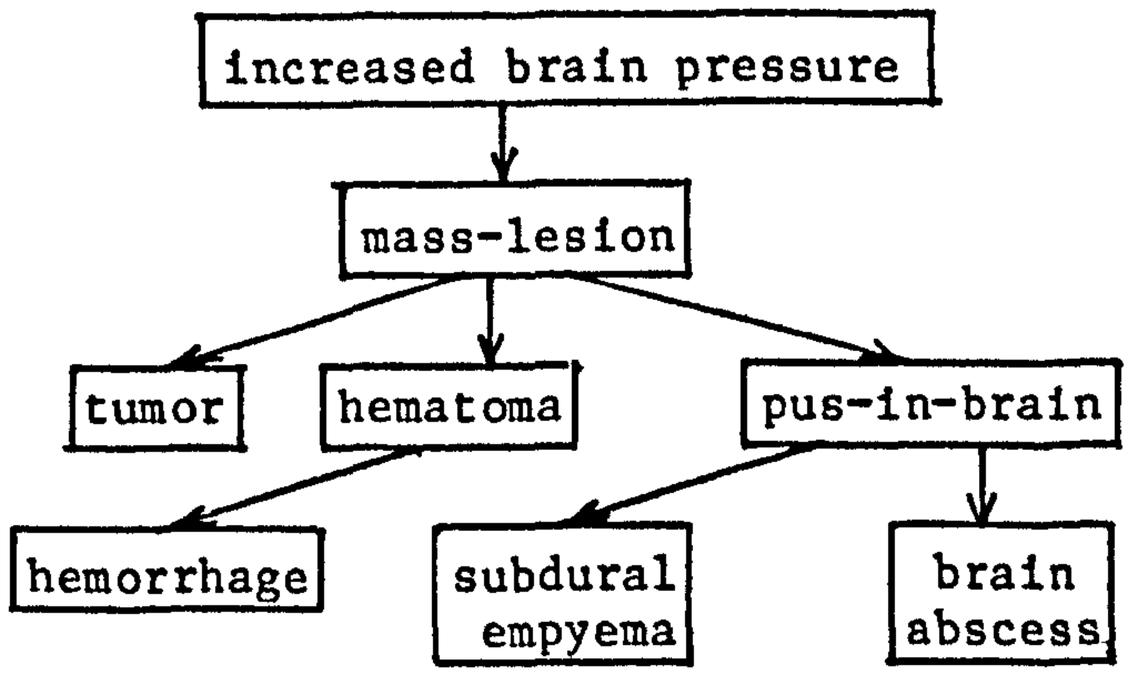


fig 8.7

A part of NEOMYCIN's causal rule network

The first rule of the network shown in figure 8.7 links the pathophysical state of increased brain pressure to mass-lesion, a possible cause of the pressure on the brain; mass-lesion in turn is caused by a tumor, hematoma or pus in the brain. Notice also that the disease state 'brain abscess' that is a state in the causal network shown in figure 8.7 above is also a category in the etiological taxonomy, see figure 8.6 above. The intersection between the etiological taxonomy and the network of causal associations is shown in figure 8.8.

---

[34] There may be other causes for state-A.

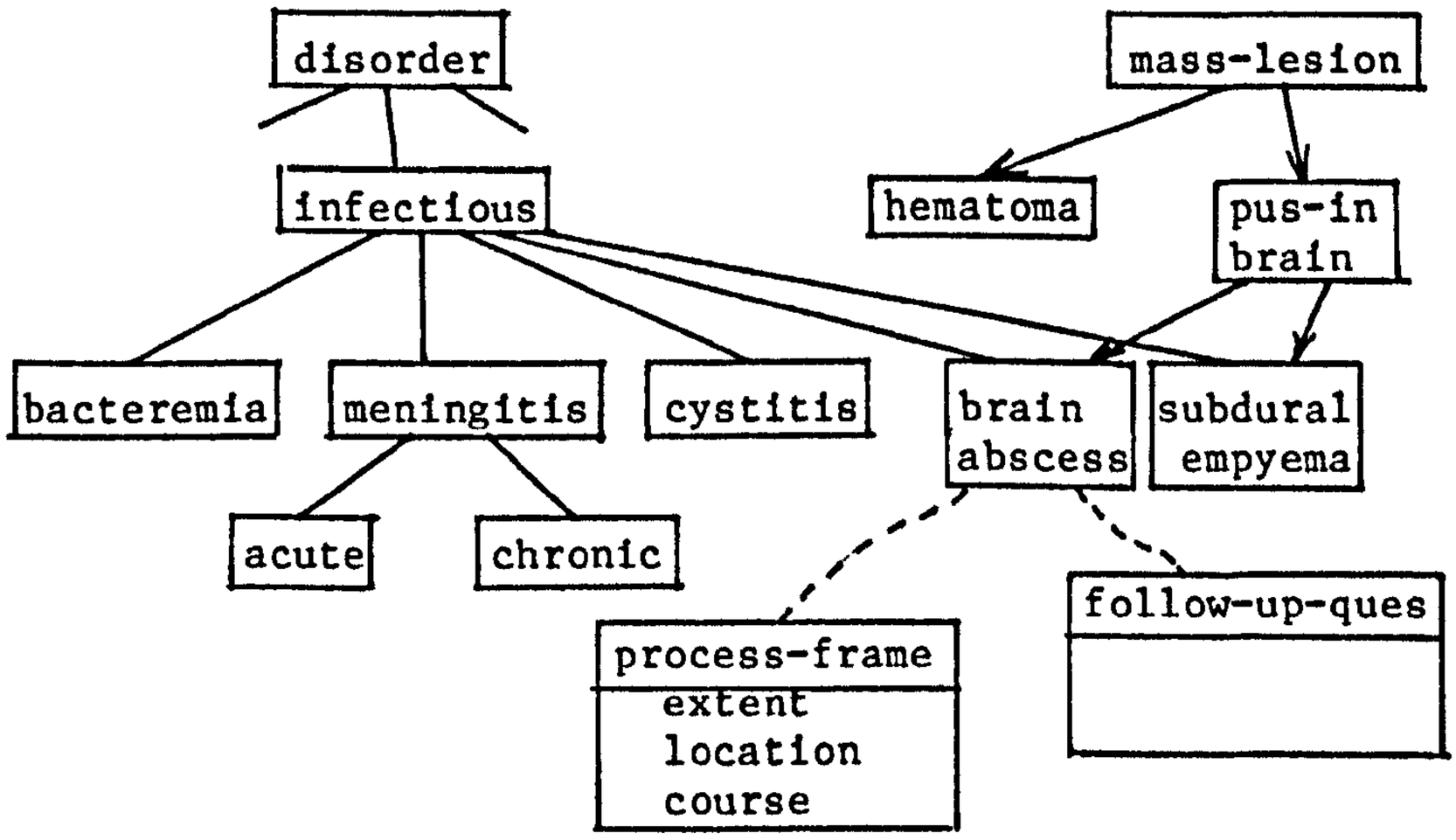


fig 8.8

The intersection of the etiological taxonomy and the causal network

Through sequences of causal associations, the causal network ultimately links patient data to disease categories in the etiological taxonomy. Figure 8.8 also shows the process knowledge and follow-up questions associated with brain abscess.

In addition to the causal links between observations and disease categories in the etiological taxonomy there are also trigger rules and data/hypothesis rules that link observations to disease categories. Trigger rules are direct links between the patient data and diseases in the etiological taxonomy. Trigger rules are intended to model the "compiled associations" that clinicians use to efficiently "jump" into the middle of the taxonomic search space. For example the trigger rule

```
if the patient is suffering from diplopia[35]
then
   consider meningitis.
```

suggests meningitis on the basis of diplopia. This rule is a "compilation" of the association between double vision, which is

---

[35] Double vision.

caused by the pressure of the accumulation of pus in the brain, which is in turn caused by reproduction of the organisms responsible for the meningitis, see figure 8.7 above.

Data/hypothesis rules also suggest hypotheses on the basis of data provided by the user. For example, the following rule

```
if the patient is an alcoholic
then
    there is evidence that that the organisms
    which might be causing the infection are
    diplococcus-pneumoniae (.3) or e.coli (.2).
```

which encodes the association between alcoholism and the possibility of infection by various organisms, might be a data/hypothesis rule.

The difference between a trigger rule and a data/hypothesis rule is that the trigger rule encodes a tentative association which is used at the start of the diagnosis to suggest possible disease categories. These categories are then pursued further by use of the data/hypothesis rules. The net effect is that a data/hypothesis rule is not invoked unless the disease suggested by the rule has already been suggested by one or more trigger rules. This part of NEOMYCIN's behaviour is controlled by the diagnostic strategy which is described later.

As is done in MYCIN, some of NEOMYCIN's requests for data from the user are screened. Notice however, that the data/hypothesis rule shown above does not contain any screening antecedents. In NEOMYCIN, the screening knowledge is separated from the other kinds of knowledge. This is done by placing all the screening knowledge as relations between categories of patient. For example, given the data/hypothesis rule above the screening knowledge about the age of alcoholics would be represented as a subtype relation such as

```
(subtype(adult, alcoholic))
```

Whenever requests for data are made from the user, subtype relations are used to avoid asking the user unnecessary questions. The way in which this is done is part of NEOMYCIN's strategy which is described below.

NEOMYCIN's strategy is domain-independent knowledge of how to use it's domain knowledge to make a diagnosis. The term `domain-independent' is somewhat of a misnomer, Clancey (1983b, p 76) explains the term

> "Domain-independent" doesn't mean that it applies to every domain, just that the term is not specific to any one domain.

Clancey (1984) describes the kinds of domain in which NEOMYCIN's strategy might apply as those where a classification problem solving method could be used. These include medical diagnosis and equipment fault finding.

In order for a strategy to be domain-independent it must be stated in terms that are not dependent on any particular domain knowledge. Clancey (ibid) makes this point clear by comparing the organisation of strategic knowledge in NEOMYCIN with that in CENTAUR. In CENTAUR the same strategic principle (if a category has been confirmed then try to confirm a subcategory of this category) is repeatedly stated in domain terms in all the disease prototypes that have subprototypes. For example, the Obstructive Airways Disease prototype contains pointers to the following subprototypes.

Pointers: (degree mild-OAD) (degree moderate-OAD) ...
(subtype Asthma) (subtype Emphysema) ...

Strategic knowledge which is attached to the `if-confirmed' slot of the prototype is as follows.

If-confirmed: Deduce the degree of OAD.
Deduce the subtype of OAD.

Notice that since `degree of OAD' and `subtype of OAD' are both subprototypes of the OAD prototype, the general strategic principle about pursuing subprototypes is stated twice, each time in terms of particular diseases i.e. domain dependent terms.

NEOMYCIN's diagnostic strategy is a method for systematically narrowing down the spectrum of possibilities that could explain a patient's symptoms. This spectrum of possibilities is called the differential. At the start of the consultation the differential is very large, far too large for an exhaustive search. The essential purpose of the strategy, therefore, is to collect that data which can be used to quickly narrow down the differential.

The strategy is composed of tasks (procedures) which in turn consists of subtasks (subprocedures). The arrangement of tasks is shown in figure 8.9.

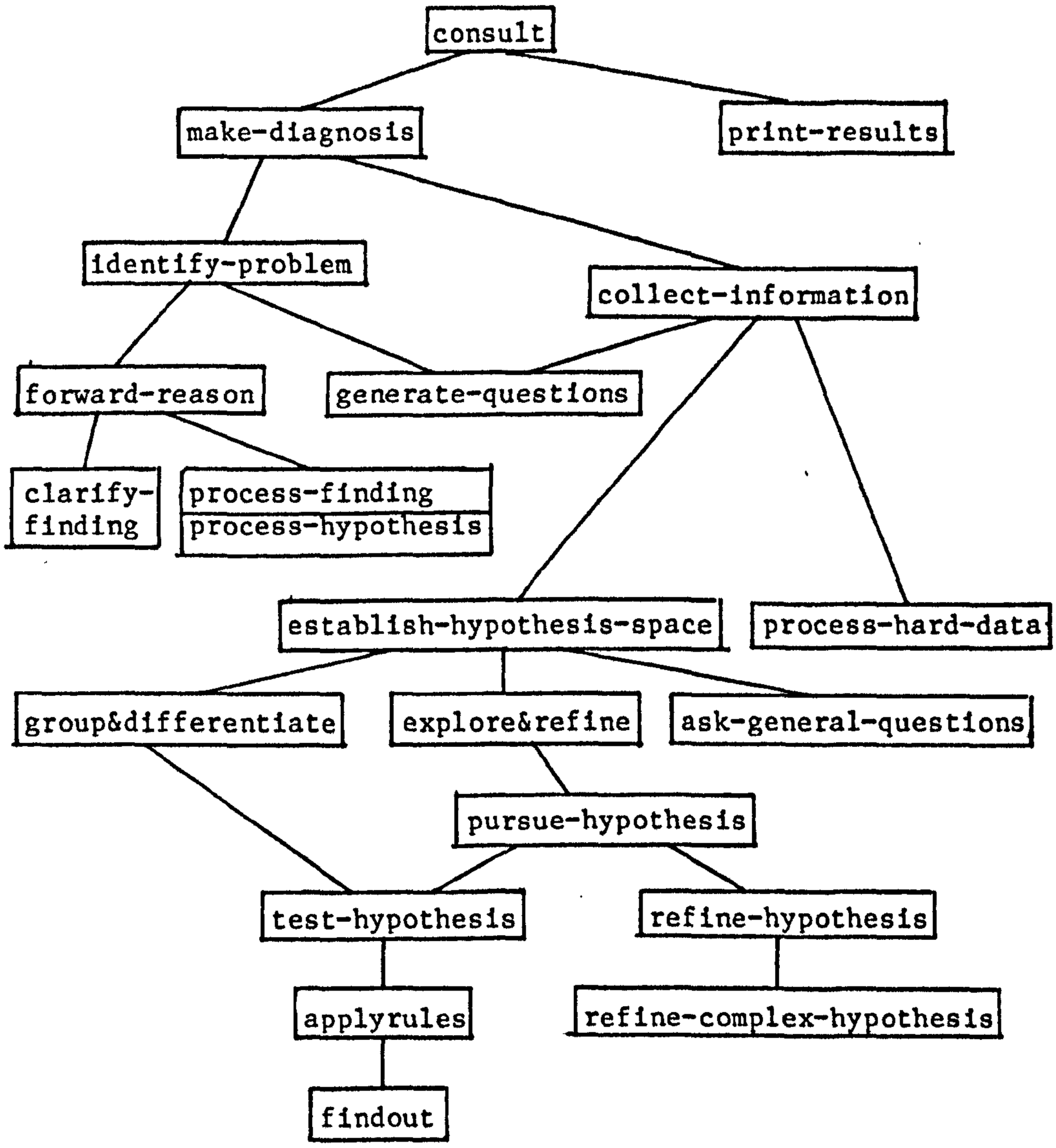


figure 8.9

NEOMYCIN's strategy (adapted Clancey, 1985, p14)

Each task shown in figure 8.9 above has a specific function to perform[36]. For example, the identify-problem task gathers initial information from the user and, making use of trigger rules, places any likely hypothesis on the differential. The establish-hypothesis-space task consists of three subtasks. The group&differentiate task is called first. This task explores any unexplored ancestor (in the etiological taxonomy) of a disease category that is on the

---

[36] The tasks were chosen with the intention of modelling the way in which a clinician would perform a diagnosis.

differential. Notice that since NEOMYCIN does not use a strict top-down strategy, the ancestor of a hypothesis will not have been explored. To explore a hypothesis the group&differentiate task calls the test-hypothesis task. The second task to be called, explore&refine, pursues any hypothesis on the differential that has not yet been pursued. A hypothesis is pursued by supplying it as the focus (argument) of the pursue-hypothesis task. Finally, any general questions that have not yet been asked are asked.

Each of NEOMYCIN's tasks contains the following components listed below.

ORDERED LIST OF METARULES
(These are the rules that perform the task's function.)

STRATEGY FOR THE METARULES
(There are four production rule control strategies for the application of the meta-rules.)

FOCUS
(This is a part of the differential that is supplied as an argument to the task.)

END CONDITION
(The end condition is evaluated, by inspecting the differential, after each rule firing. If the condition is satisfied then the task halts and control is returned to the calling task.)

An example of a meta-rule belonging to the explore&refine task is shown in figure 8.10, (Clancey, 1983b, p75).

```
task: explore&refine
focus: current-hypothesis

if the hypothesis being focused upon has a child
   that has not been pursued,
then
   pursue that child.
```

figure 8.10

One of the meta-rules representing NEOMYCIN's diagnostic strategy The meta-rule of figure 8.10 above is able to determine the "child" of a hypothesis by "looking up" the hypothesis in the taxonomy of

disease categories. This is an example of how the domain-independent strategy makes use of the domain knowledge.

One of the meta-rules of the findout task is responsible for screening questions to the user. The rule is shown in figure 8.11,

```
task: findout
focus: desired-finding
if  the desired finding is a subtype of a class of findings
and the class of findings is not present
then
    conclude that the desired finding is not present.
```

figure 8.11

A meta-rule representing NEOMYCIN's screening strategy

NEOMYCIN's domain knowledge would contain, for example, the class subtype relation

(subtype(adult, alcoholic))

The second antecedent of the meta-rule would then recursively invoke findout to determine if the patient was an adult. If the patient is an adult then the meta-rule fails and control is passed to the next meta-rule of the findout task, eventually the user is asked for the presence or otherwise of the finding.

One of the important differences between NEOMYCIN and MYCIN is the forward reasoning ability of NEOMYCIN's strategy. NEOMYCIN's forward reasoning knowledge is represented in the forward-reason task. This task contains three subtasks, clarify-finding which asks for additional information associated with a finding, process-finding and process-hypothesis.

The process-finding task includes the following ordered set of meta-rules.

```
if  there are any causal and definitional rules
    that use the finding
and the rules can be applied now
then
    apply those rules.


if there are any subtype relations in which the
   finding appears
then
   apply those relations
```

```
if  the finding under consideration appears as a
    condition in the premise of a trigger-rule
then
    applyrules (trigger-rule, with-subgoaling)
```

Notice here that the ordering of these rules is determined by the knowledge engineer and not by a meta-meta-rule. The ordering may not be significant here but if it is, no rule can be displayed to explain it. Instead explanation can be provided by "canned" text.

COMPARISON OF MYCIN WITH NEOMYCIN

In this section we compare the modifiability of MYCIN and NEOMYCIN by examining the way in which various modifications are made to each system. We show for each example that the ability to perform modifications as extensions depends on the organisation of knowledge in the system.

The domain-independence of NEOMYCIN's strategy is an important aspect of its organisation of knowledge. In the next section we describe the modifiability implications of this organisation of knowledge.

Some modifications can be performed as extensions to both MYCIN and NEOMYCIN. However, we show how in practice the modifications are not equally convenient since MYCIN's representation of knowledge serves to obscure the organisation of knowledge.

Addition of screening knowledge to MYCIN and NEOMYCIN

In this example modification we consider how knowledge of a screening relation would be added to MYCIN and to NEOMYCIN. Assume that the screening relation to be added to each system is the relation that is used to screen questions about alcoholism. Before questions about alcoholism are put to the user we require the systems to check that the patient is an adult (over seventeen years old). The knowledge to be added to each system is therefore the knowledge that only adults can be alcoholic.

MYCIN uses a backward chaining strategy in which an attempt is always made to satisfy a goal by chaining back through its rules rather than asking the user for the value of the parameter. Given that MYCIN possesses this strategy, the knowledge of the screening relation can be added as a rule

```
if  the patient less than 17 years old
then
    the patient is an alcoholic(-1.0).
```

Notice that the relation must be stated in its negative form since only in this form can the age of the patient be used make a definite conclusion about the possibility that the patient is an alcoholic. If the patient is over seventeen years old then it is not possible to make any decision about alcoholism.

Clearly, the addition of the screening relation can be performed as an extension. The rule representing the relation is simply added to the rule-base.

To account for the extensibility of MYCIN's rule-base for the addition of screening relations we must realise that a screening relation is a precondition for the existence of some situation that should be tested for before testing for the presence of that

situation. MYCIN's "if-then" rules can be thought of as modelling preconditions for the presence of particular situation. Furthermore, since MYCIN operates a strategy of using its own knowledge before asking the user for some datum and this is exactly the strategy for the application of screening relations.

Notice that the screening knowledge need not be added as an entire rule. Since the screening knowledge implies that being at least seventeen years of age is a precondition for being an alcoholic it can be added by inserting the condition

the patient is over 17 years old

into the rule

```
if 1) the infection is meningitis
   2) only circumstantial evidence is available
   3) the subtype of meningitis is bacterial
   4) the patient is an alcoholic
then
   there is evidence that that the organisms which might be causing
   the infection are diplococcus-pneumoniae (.3) or e.coli (.2).
```

The modified rule will be

```
if 1) the infection is meningitis
   2) only circumstantial evidence is available
   3) the subtype of meningitis is bacterial
   4) the patient is over 17 years old
   5) the patient is an alcoholic
then
   there is evidence that that the organisms which might be causing
   the infection are diplococcus-pneumoniae (.3) or e.coli (.2).
```

It is important that the new clause is inserted before the antecedent that specifies that the patient is an alcoholic. The reason for this is, of course, that MYCIN's rule interpreter always attempts to establish antecedent conditions in the order in which they appear in the rule.

The disadvantage of this latter method of adding the screening knowledge to the rule-base is that the new antecedent must be added to

all the rules that enquire if the patient is an alcoholic. However, using either of the two methods shown above the knowledge can be added as an extension to the knowledge-base.

Notice that further screening relations can also be added as extensions. Further relations can simply be added as further rules to the rule-base. MYCIN's backward chaining strategy will ensure that subtypes of subtypes are correctly chained together. For example, suppose that MYCIN records at the start of a consultation, whether a patient is a child, say under 5 years old, or not. We might then add the screening rule

```
if the patient is a child
then
   the patient is not over 17 years old.
```

The chaining strategy would link the screening rules and form the chain of relations shown in figure 8.12

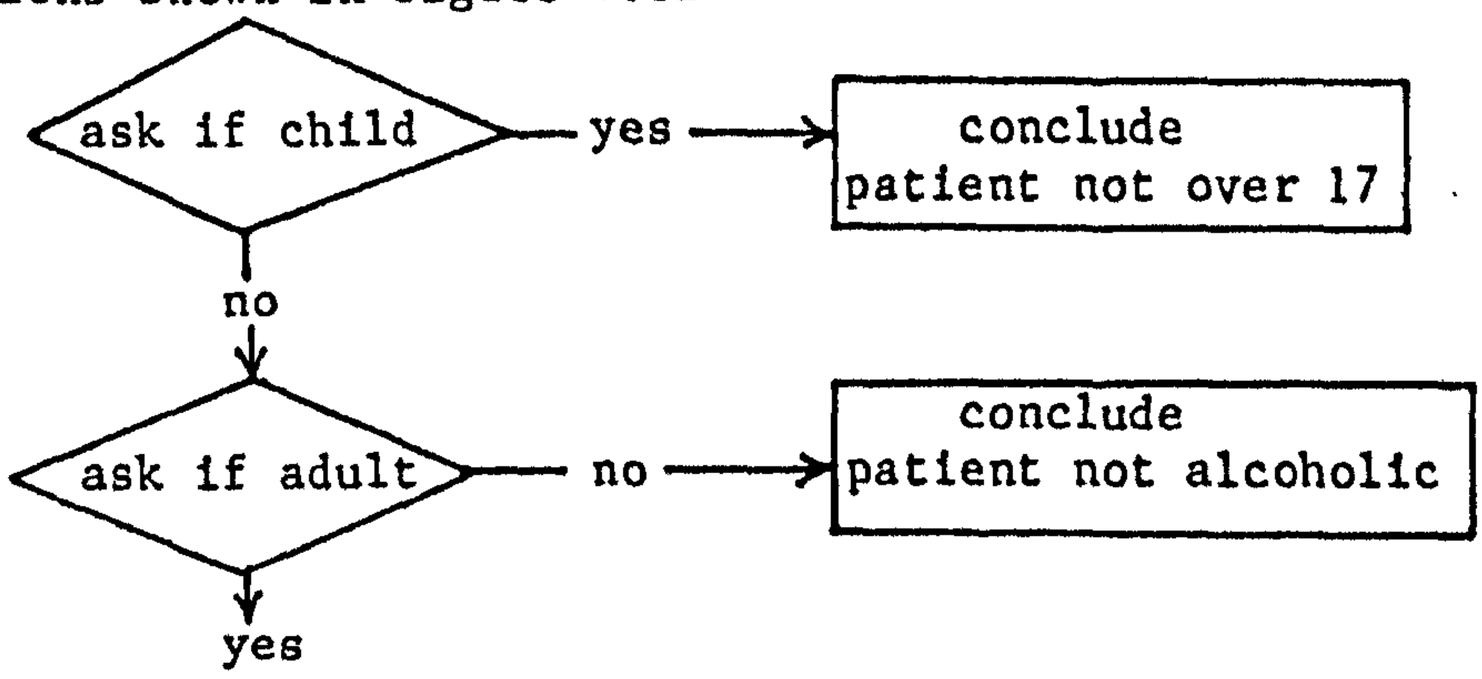


figure 8.12

Screening relations linked together

Consider now how the screening knowledge can be added to NEOMYCIN. Recall that NEOMYCIN's findout task contains the meta-rule

```
if  the desired finding is a subtype of a class of findings
and the class of findings is not present
then
    conclude that the desired finding is not present.
```

which represents an abstract strategy for using screening relations. In order for this meta-rule to be applied the first antecedent of this rule must be able to establish if the desired finding is a subtype of some class of finding. NEOMYCIN has a collection of subtype relations for representing this kind of knowledge. Consequently the screening relation can be added as

(subtype(adult, alcoholic)).

As was the case with MYCIN, further screening relations can be added to NEOMYCIN by simply adding further subtype relations.

The addition of this knowledge to NEOMYCIN can be done as an extension, as it can to MYCIN. The reason for this is that MYCIN and NEOMYCIN both use the same "don't ask if it can be avoided" strategy for asking the user about findings. In MYCIN this strategy is "wired into" the findout routine rather than represented as rules. In NEOMYCIN, the strategy for establishing a finding is represented as a strategic meta-rule. This difference in representation, although significant for explaining screening knowledge, does not affect the extensibility of the systems for this kind of knowledge.

The difference in the representation of strategic knowledge described above, however, should not be confused with the difference in the organisation of strategic knowledge that we will describe later. In particular, NEOMYCIN's organisation of knowledge is such that screening can be omitted by simply removing its meta-rule representation. To remove MYCIN's backward chaining strategy would lead to the collapse of the entire system.

Addition of screening knowledge: clarity of representation

Having considered how screening knowledge would be added to each system, we now consider the clarity of the way in which the organisation of screening knowledge is represented in each system. In MYCIN, each individual screening relation is either represented as an individual rule or as an antecedent in some other rule.

The representation of relations as individual rules is clear enough except that it is not possible to tell which rules in the rule-base represent screening relations and which represent some other kind of knowledge. This becomes a problem when the knowledge engineer wishes to determine if a particular screening relation is present in the rule-base. The knowledge engineer must search through the entire rule-base rather than through some smaller set of rules marked as screening rules[37].

The representation of relations as combinations of antecedents (inserting a screening condition before a condition in a rule) as opposed to individual rules, presents further problems. This is because individual screening relations are now represented together with other kinds of knowledge in a single rule. To find the representation of a screening relation the knowledge engineer must in this situation search through the antecedents of rules.

Additional problems may arise if the same screening relation is represented as an ordered pair of antecedents in more than one rule. This is the sort of situation described above where all occurrences of the antecedent

---

[37] Again, sophisticated rule editors such as TEIRESIAS (Davis, 1979, 1980) will be of help here. However, for the reasons given in the previous chapter we are considering the clarity of MYCIN's representation as it appears to a knowledge engineer without access to such tools.

the patient is an alcoholic

are preceded by the antecedent

the patient is over 17 years old

If a modification to this relation becomes necessary then the knowledge engineer must find and modify all occurrences of the pair of antecedents. This is unfortunate since the knowledge engineer is in reality only modifying one relation.

Although those parts of the organisation of knowledge that are concerned with screening conditions are the same in MYCIN and NEOMYCIN, NEOMYCIN's representation has the advantage of clarity. There are a number of reasons for this. Firstly NEOMYCIN's representation of screening knowledge is all collected in one place, namely the set of subtype relations. Consequently the knowledge engineer does not suffer the problem of disentangling screening knowledge form other kinds of knowledge. In particular there is no need to search a large rule-base.

Secondly, the strategy for the use of screening relations, is represented as a single meta-rule in NEOMYCIN. The advantage of this is that the strategy is conveniently available for inspection and indeed can be explained by NEOMYCIN itself. In contrast, MYCIN's backward chaining strategy is not readily observable from an inspection of the rule interpreter.

Thirdly, each individual screening relation is represented once only. The knowledge engineer working with NEOMYCIN's representation does not encounter the problem of modifying a number of representations of a single relation.

Addition of data/hypothesis knowledge

The modification to the screening knowledge described above can be performed as an extension in both MYCIN and NEOMYCIN. The following example modification can be performed as an extension in NEOMYCIN but not in MYCIN. The required modification is to add knowledge of an association between some patient data and a disease hypothesis. Assume for example that, the association to be added to the knowledge-bases of each system is an association between the alcoholic condition of a patient and the patient's disposition to infection by particular organisms.

In MYCIN, the above knowledge is represented as the following rule.

```
if the patient is an alcoholic
then
   the following organisms ... should be covered for.
```

However, such a rule cannot be added to MYCIN's rule-base without causing an error. The reason for this is that the above rule does not specify the context in which it should be applied and hence may be applied in the wrong context. To elaborate, the kind of organisms to be covered for depends on the type of the infection, bacterial, viral etc. MYCIN has a single parameter `organisms-to-be-covered-for' that must be given an appropriate value, irrespective of the type of the patient's infection. For example, in the case that the patient has a bacterial infection rules of the form

```
if 1) the infection is meningitis
   2) only circumstantial evidence is available
   3) the subtype of meningitis is bacterial
   4) the patient is an alcoholic
then
   there is evidence that that the organisms which might be causing
   the infection are diplococcus-pneumoniae (.3) or e.coli (.2).
```

would be used to determine the organisms to be covered for. In the

case that the patient has a viral infection, rules of the form

if 1) the infection is meningitis
   2) only circumstantial evidence is available
   3) the subtype of meningitis is viral
   4) the patient is an alcoholic[38]
then
   there is evidence that that the organisms which might be causing
   the infection are virus-A (.4) or virus-D (.3).

would be used to determine the organisms to be covered for. Clearly the proposed rule would be incorrectly applied in the case in which the patient has a viral infection.

Suppose that the proposed rule only applies if the patient is suffering from bacterial meningitis. As the rule stands, there is nothing to prevent its application in the case where the patient is suffering from viral meningitis, and hence the error.

Since the addition of the knowledge associating alcoholism with various bacterial infections cannot be added to MYCIN's rule-base without also requiring that the expert provide knowledge to ensure that rule is used in the appropriate context the knowledge cannot be added as an extension.

In order to provide knowledge of the context in which a data/hypothesis association should be used it is necessary to provide MYCIN with a description of the context and with the strategic knowledge for establishing this context. In other words, it is not sufficient to simply add the antecedent

if 1) the subtype of meningitis is bacterial

to the proposed rule. The above antecedent would correctly specify the context in which the association should be used, however the

---

[38] In the case of bacterial meningitis, bacteria are transferred from the stomach into the meninges by the inhalation of vomit. However, in the case of a viral infection the alcoholic's typical rundown state makes him or her susceptible to particular viruses.

following rule

if 1) the subtype of meningitis is bacterial
   2) the patient is an alcoholic
then
   there is evidence that that the organisms which might be causing
   the infection are

would still be in error. The reason for this is that an attempt could be made to establish that the infection is bacterial meningitis even though the meningitis category has not been established. The way in which strategic knowledge is added to MYCIN's organisation of knowledge is to add knowledge of the disease categories to be established and of the order in which they should be established. As described above, this knowledge is added by including antecedents such as

the infection is meningitis
the subtype of meningitis is bacterial

in all the rules that should be applied in the bacterial meningitis context.

Consider now how knowledge of the same association would be added to NEOMYCIN. In NEOMYCIN the knowledge is represented as the data/hypothesis rule

if the patient is an alcoholic
then
   the following organisms ... should be covered for

Recall that data/hypothesis rules are only invoked when there is some evidence that the disease category on which the rule concludes is present in the patient. In other words, the above data/hypothesis rule is not invoked unless there is already some evidence that the meningitis is bacterial. Consequently there is no need to specify, in the rule, the context in which the association holds. For this reason the association between patient data and organisms to be

covered for is complete "as it stands" and can be added as an extension to NEOMYCIN's knowledge-base.

MYCIN's organisation of knowledge cannot deal with associations between patient data and organisms to be 'covered for' unless the context in which the association holds is included. In other words, in every rule that concludes on the organisms that need to be covered for, it is necessary to have the context clauses

> the infection is meningitis
> the subtype of meningitis is bacterial/viral.

As Clancey (1983a, p136) states,

> ... the last two goals (shown as specific hypothesis 'meningitis?' and 'bacterial?') correspond to the first and third clauses of the 40 'cover for' rules similar to the alcoholic rule.

In NEOMYCIN's organisation of knowledge however, the knowledge used to establish the context of the diagnosis is represented abstractly in the domain-independent search strategy for the etiological taxonomy. Consequently, given that this strategy is present in the knowledge-base, data/hypothesis can be simply added also.

Addition of data/hypothesis knowledge: clarity of representation

The remarks made above concerning the clarity of the representation of screening knowledge in MYCIN and NEOMYCIN are equally applicable in this situation. MYCIN's representation of data/hypothesis associations are such that the data/hypothesis knowledge is mixed with other kinds of knowledge. Usually, part of a rule represents data/hypothesis knowledge and other parts will represent context knowledge and screening knowledge.

In NEOMYCIN however, the data/hypothesis knowledge is represented in data/hypothesis rules that are distinct from rules used to represent other kinds of knowledge. Again, NEOMYCIN's representation "mirrors" its organisation of knowledge.

Modifications to strategic knowledge

Like CENTAUR, (discussed in the previous chapter) NEOMYCIN makes use of triggering associations in its diagnostic strategy. In NEOMYCIN, triggering associations can be added as extensions to the knowledge-base. Triggering associations are simply expressed as rules and added to the collection of triggering rules in the domain knowledge.

Recall that in the previous chapter we considered the problem of adding triggering associations to PUFF. In considering how triggering associations might be added to PUFF we demonstrated the awkwardness of representing a strategy for using triggering associations in the backward chaining rule system from which PUFF is built. Since PUFF is an EMYCIN system we can expect the addition of triggering associations to MYCIN to be beset by the same sort of problems. Hence, to avoid repeating that discussion here, we will not further discuss the addition of triggering associations to MYCIN and NEOMYCIN.

However, we can illustrate an important part of the modifiability of NEOMYCIN's strategy by examining some of the modifications that Clancey (1985) has considered[39]. One of these modifications involves a change to the way in which triggering associations are used.

At the present the process-finding task will attempt to obtain the necessary patient data to apply a trigger rule whenever the

---

[39] Clancey describes the modifications under consideration only in English prose.

currently obtained patient data satisfies at least one condition of the rule. For example, the following trigger rule

```
if  the patient has a headache
and the patient has a stiff neck
then
    meningitis is a possible disease category(0.4).
```

requires that the patient has a headache and a stiff neck before the rule can be applied. However, if the user only reports that the patient has a headache, the following forward-reason meta-rule

```
if  there are trigger-rules in which the current finding appears
    as a condition.
then
    applyrules (trigger-rules, with-subgoaling).
```

will arrange for the user to be asked if the patient also has a stiff neck in the hope that the trigger rule can be applied.

The facts about the patient such as 'headache' and 'stiff neck' are called findings. There are two types of finding, specific and non-specific. The specific findings are the more reliable indications of particular disease categories.

The modification considered by Clancey requires that trigger rules are applied differently depending on whether the finding is specific or non-specific. In addition, specific findings should continue to be used as cues for the application of triggering associations. However, non-specific findings should not act as cues for the application of triggering associations unless the hypothesis is already present in the differential.

The strategies to be used for each of the two kinds of finding can be represented as the two following meta-rules.

```
if  the current finding is a specific finding
and there are trigger-rules in which the current finding appears
    as a condition
then
    applyrules (trigger-rules, with-subgoaling).


if  the current finding is a non-specific finding
and there are trigger-rules in which the current finding appears
    as a condition
and the trigger-rules conclude on a hypothesis
    that is present in the differential
then
    applyrules (trigger-rules, with-subgoaling).
```

Consequently the modification can be performed as an extension simply by replacing the previous meta-rule with the two rules shown above.

It is NEOMYCIN's organisation of strategic knowledge that allows the two new meta-rules to be added as extensions. An important part of this organisation of knowledge is the distinction between specific and non-specific findings. If this distinction were absent then it would not be possible to determine the validity of the first antecedent of the new meta-rules. In which case the meta-rules could not be added as extensions.

Another important part of the organisation of the strategic knowledge is the breakdown of each task in the strategy, into a particular sequence of steps. Depending on the particular breakdown of the task into steps, the task can be modified by the addition, deletion or modification of particular steps. The steps of the process-finding task are shown below.

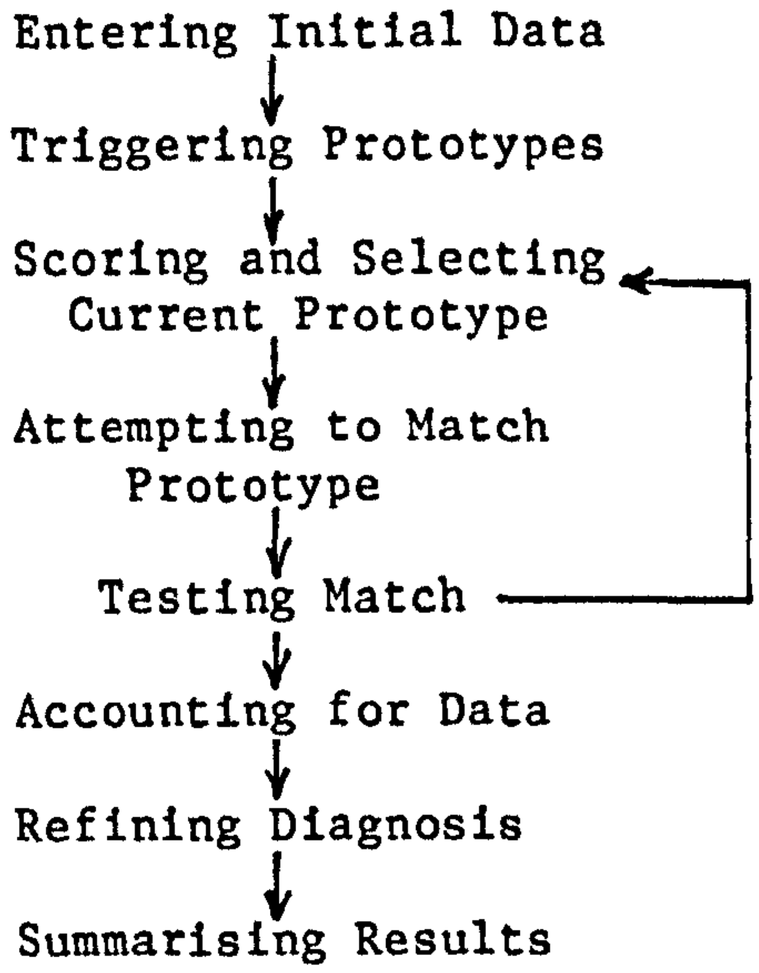
```
Apply any causal and definitional knowledge that is
currently applicable.


Apply any knowledge of generalisations of the
finding that are currently applicable.

Apply any triggering associations that use the finding.
```

Apply any data/hypothesis associations that require circumstantial evidence and conclude on a hypothesis in the differential, or on an ancestor or immediate descendent. No subgoaling is allowed.

Apply and data/hypothesis associations that require laboratory evidence and conclude on a hypothesis as described above, subgoaling is allowed.

Each step in the procedure is concerned with the application of a distinct kind of domain knowledge. The modification described above only concerns one kind of domain knowledge knowledge, triggering associations. Modifications can be made to the way this kind of domain knowledge is used without having to change the way in which any other kinds of domain knowledge are used. For this reason the modification does not affect any other step in the procedure. Each step in the above procedure is represented as a meta-rule but this is irrelevant to the extensibility of the procedure. They could equally well be represented as sections lisp code.

There is little point in discussing how MYCIN might be modified to accommodate the new knowledge about the use of triggering associations. This is because in this aspect, MYCIN's organisation of knowledge is so far removed from NEOMYCIN's, that the comparison becomes meaningless. However, it is illustrative to compare the strategy used by NEOMYCIN with that used by CENTAUR. Recall that, in CENTAUR the diagnostic strategy consists of the following steps



The 'triggering prototypes stage' shown above consists simply of a task that applies all the applicable trigger rules. This task is attached to a slot in the consultation prototype.

There are two ways in which CENTAUR could be modified to distinguish between specific and non-specific findings in the application of its triggering associations. Firstly, modifications could be made to the set of trigger rules. This would probably be the most convenient modification but, as we shall see, it would not be the addition of the knowledge provided by the expert and hence not an extension. Secondly, CENTAUR's strategic knowledge could be modified to apply the existing trigger rules in the new way. We describe each of these alternatives, beginning with the modification to the individual trigger rules.

Suppose that the following rule is one of CENTAUR's present trigger rules.

```
if  the diffusing capacity for carbon monoxide is
    less than 90% of normal
then
    suggest Diffusion Defect with a certainty measure of 800.
```

If the diffusing capacity for carbon monoxide is a specific finding then this trigger rule does not require modification. If however,

diffusing capacity is a non-specific finding then the trigger rule should be modified as shown below.

if the diffusing capacity for carbon monoxide is
    less than 90% of normal
and the certainty measure of Diffusion Defect is
    greater than 200[40]
then
    suggest Diffusion Defect with a certainty measure of 800.

By checking the findings used in each trigger rule and enquiring of the expert as to their specific or non-specific status, the knowledge engineer can suitably modify trigger rules as necessary.

The modification is not yet complete however, since there still remains a subtle error in the trigger rules as they stand. The error can be demonstrated by considering the following situation. Assume that we have the trigger rules

R1: S ---> P

R2: N ---> P

where S is a specific finding and N is a non-specific finding, P is the prototype suggested by the trigger rule. If R1 is applied first then the certainty measure of P is increased to a level such that R2 can be applied. However, if the attempt is made to apply R2 before applying R1 then the certainty measure of P will not be sufficiently high and R2 cannot be applied. A way around this problem is to order the trigger rules so that all the rules that use specific findings are applied before those that do not.

The modification described above is convenient to the extent that it is convenient to obtain from the expert the knowledge about the status of each of the findings used by the trigger rules and that the problem of ordering the rules is not overlooked. It is important to

---

[40] Assume that once the certainty measure of a prototype reaches 200 it is regarded as a possible hypothesis.

notice here that the knowledge being represented is domain specific as opposed to the domain-independent knowledge that was added to NEOMYCIN. This creates problems when the status of a finding changes. If the status of a finding should change, from specific to non-specific say, then it is necessary to check all the trigger rules for possible modification as described above.

In order to supply CENTAUR with the domain-independent strategy for the application of trigger rules it would be necessary to modify the task (lisp function) that applies the trigger rules. This task must be modified so that the trigger rules are applied in two stages. In the first stage, all those applicable trigger rules that make use of at least one specific finding should be applied. Once these rules are applied, they will suggest one or more prototypes as hypotheses. In the second stage, the applicable trigger rules that do not use any specific findings should be applied, with the proviso of course that the prototype on which the rule concludes has a sufficiently high certainty measure.

In order to introduce the strategy described above it is necessary to label each finding as either specific or non-specific. A function would then be required to check the status of of each of the findings used in the premise of each trigger rule. This function would be used to determine if a trigger rule is satisfied by specific or non-specific findings. Another function would be required to check that the prototype suggested by a "non-specific" trigger rule has a sufficiently high certainty measure for that rule to be applied.

Clearly, CENTAUR's organisation of knowledge is not suitable for the addition of the strategic knowledge as an extension. The main reason for this is that CENTAUR does not use an abstract (domain-independent) diagnostic strategy. Nor does CENTAUR's organisation of

domain knowledge distinguish between specific and non-specific findings. In contrast, NEOMYCIN's organisation of knowledge includes these kinds of knowledge.

Modification to strategic knowledge: clarity of representation

Before modification, CENTAUR's representation of knowledge clearly reveals its organisation. However, once CENTAUR's trigger rules have been modified, this part of the representation becomes obscure. Previously, CENTAUR's simply applied a set of trigger rules, all the trigger rules were of the same sort and the order in which they were applied did not matter. Now there are two kinds of trigger rule, one of which is always applied before the other. The knowledge engineer will notice that some of the trigger rules are of the following form

```
if  finding-N is present
and the certainty measure of prototype-P is greater than 200
then
    suggest prototype-P with a certainty measure of ...
```

and that these rules are applied last. However, the knowledge engineer has no clue from the representation that the distinction is due to the status of the findings that appear in the rule. This ignorance will lead to problems if the status of a finding changes.

The modification to CENTAUR's strategic knowledge (i.e. modifications to the tasks that apply the trigger rules) is potentially less obscure. Providing that the strategic knowledge represented by the the newly introduced tasks is made clear then the knowledge engineer can inspect these tasks to determine the way in which the trigger rules are applied. NEOMYCIN's use of meta-rules, which are "self documenting" to some extent, is one way to do this. Another way in which the knowledge represented by a task can be made clear is to associate it with some canned text that describes the

task.

In this chapter we have again considered in detail the sorts of problem that arise when modifying the large and complex organisations of knowledge that are used in expert systems. In general, we have shown the value of being able to perform a modification as an extension.

If the modification cannot be performed as an extension there may be some other knowledge that can be added to provide the required performance. For example, the addition of strategic knowledge to CENTAUR was difficult to perform as an extension. However, there is a simpler way of modifying the system in order to provide the correct behaviour. Instead of adding the domain-independent strategic knowledge, various domain specific "instantiations" of that strategy can be added. This may be a simpler way of modifying the system but it is likely to lead to an organisation of knowledge that lacks "conciseness" and is consequently more difficult to understand.

There is clearly no one correct organisation of knowledge to use. The knowledge engineer must carefully weigh up the trade off between the longer term modifiability advantages provided by a more complex organisation of knowledge and the shorter term advantage of ease of construction that is provided by a simpler organisation of knowledge.

The case study in this chapter serves to corroborate the conclusions of the PUFF/CENTAUR case study of the previous chapter. In particular, we notice the value of using a representation language that allows the perspicuous representation of a given organisation of knowledge. Such a language can also considerably offset the difficulty of constructing representations of complex organisations of knowledge.

We also saw, once again, that in a single domain it is possible to use significantly different organisations of knowledge, different enough to require very different representation languages. Feigenbaum (1977, p1027) goes part way towards recognising this point.

> Situation => action rules are used to represent expert's knowledge in all of the case studies[DENDRAL, META-DENDRAL, MYCIN, TEIRESIAS, SU/X[41], HEARSAY[42], AM, MOLGEN[43] and CRYSALIS[44]]. Always the situation part indicates the specific conditions under which the rule is relevant. The action part can be simple (MYCIN: conclude presence of particular organism; DENDRAL: conclude break of particular bond). Or it can be quite complex (MOLGEN: an experimental procedure). The overriding consideration in making design choices is that the rule form chosen be able to represent clearly and directly what the expert wishes to express about the domain.

Feigenbaum acknowledges the value of being able to represent 'clearly and directly what the expert wishes to express'. However, he is only willing to admit representations in rule form[45]. Our examination of NEOMYCIN's organisation of knowledge, and also CENTAUR's in the previous chapter, strongly suggests that these organisation of knowledge cannot be adequately represented within the rule scheme.

---

[41] Nii and Feigenbaum, 1978.

[42] Lesser and Erman, 1977.

[43] Martin et al, 1977.

[44] Englemore and Nii, 1977.

[45] In practice this need not be any sort of restriction at all if the 'situation' and 'action' parts of a rule can be arbitrarily complex as Feigenbaum suggests. However, we assume that some substantive point is being made here about the wide ranging utility of the rule scheme.

CHAPTER NINE

SUMMARY, CONCLUSIONS AND FURTHER WORK

SUMMARY AND CONCLUSIONS

Several important expert system projects, important with respect to the influence they have had in the development of this field, have made use of the rule representation scheme. One of the claims made about the use of the rule scheme is that rule representations of knowledge are conveniently modified, and it is this claim that we have examined in this thesis.

We distinguished two senses in which claims about the modifiability of the rule scheme could be understood. In the first sense, claims about the modifiability of the rule scheme assert that the rule scheme is a convenient "high level" language for the representation of knowledge. As such one can expect system builders using the scheme to be more productive than those who construct knowledge-bases directly in an "artificial intelligence programming language" such as Lisp. In the second sense, claims about the modifiability of the rule scheme assert that rule representations of knowledge allow the extension of that knowledge. This is essentially the ability to add new knowledge to the representation without modifying any knowledge already represented there.

With regard to the claim that the rule scheme provides the kind of modifiability provided by the use of a high level language, we would say that this is largely an empirical claim and, except for the the two case studies of rule-based systems, we do not consider this claim directly. Instead, the thesis concentrates on the claim that rule representations of knowledge are extensible, a more interesting claim because this kind of modifiability is potentially much more

valuable for systems constructed incrementally.

The main conclusion of the thesis is that the extensibility of particular representations of knowledge should be accounted for by considering the organisation of knowledge rather than the way in which that organisation of knowledge is represented. Consequently we claim that rule representations of knowledge are not inherently any more extensible than any other kind of knowledge representations.

A justification sometimes advanced for the claimed modifiability of the rule scheme, is its so called "modularity". The claimed "modularity" of the rule scheme is said to allow the scheme to represent knowledge in small "independent chunks". The argument here is that since knowledge is represented in small "independent chunks" the knowledge can be easily modified by the addition, deletion, etc. of individual "chunks" of knowledge.

We pointed out that although the rules of a "rule language" are usually modules in the conventional software systems sense, it does not follow that they are the particular modules necessary to make modifications convenient. We also argued that rules may indeed be taken to be independent of each other for the purposes of modification but this independence is a result of the organisation of knowledge rather than the representation of that knowledge as rules.

Although the organisation of knowledge is independent of the way in which it is articulated in some representation scheme, this is not to say that the choice of scheme does not affect the overall modifiability of the system. Since, understanding an organisation of knowledge is clearly a prerequisite for modifying the system, a suitably chosen scheme can improve the clarity and hence the modifiability of a representation.

The two case studies provided examples of the effect of the representation scheme on the clarity of a representation. For example, we saw that although modifications to PUFF's diagnostic strategy (changing the order in which disease hypotheses were pursued) were hampered by the obscurity of the representation, they were nonetheless extensions and once the encoding of knowledge was "uncovered", quite straightforward. We saw that modifications that were extensions (albeit obscure), were nonetheless much more convenient to perform than non-extensions such as the addition of triggering associations to PUFF. Similar conclusions were drawn from the MYCIN/NEOMYCIN case study.

If we view a modification to a representation of knowledge as a modification to the knowledge represented rather than simply a modification to the representation then it's clear that there is no single modification problem. Organisations of knowledge differ, and therefore so must the ways in which they can be modified. In other words, although we may casually talk of the problem of modifying representations of knowledge, as if there is but one problem, there is in reality at least as many different modification problems as there are organisations of knowledge. This is not to say that superficial differences in a organisation of knowledge are important. As we saw in the MYCIN/NEOMYCIN case study, crucial differences consist of such things as way in which an organisation of knowledge is decomposed into domain-independent and domain specific parts.

However, if contrary to the view just described, the many different knowledge modification problems are viewed as manifestations of a single underlying problem, namely the problem of 'how can knowledge be modified?', then it is natural to search for a single solution. One such "solution" requires that knowledge be represented

in a particular scheme that has special properties, ... no more need be said.

FURTHER WORK

In this thesis we have argued that the organisation of knowledge determines the modifiability of a representation. This result has important implications for the way in which expert systems are developed. Essentially, the knowledge engineer must take responsibility for the organisation of knowledge that is being constructed. He or she cannot merely extract rules from the expert and include them in the knowledge-base simply because they incrementally improve the performance of the system. A knowledge engineer who does this risks creating a knowledge-base with an organisation of knowledge that eventually makes the knowledge-base "unmodifiable". Instead, the knowledge engineer's primary task should be the elicitation and construction of a suitable organisation of knowledge. Further work should therefore be directed at researching particular abstract organisations of knowledge and methods for knowledge elicitation that lead to the specification of an organisation of knowledge without premature commitment to a particular representation scheme.

One might choose to study various (perhaps similar) organisations of knowledge in order to learn something of their general and abstract characteristics. More specifically, the modifiability of particular organisations of knowledge could be studied. Diagnosis problem solving is one area in which the organisations of knowledge are becoming well researched[46] and Clancey (1984), in particular, provides

---

[46] As an indication of the interest in this area the IEEE transaction on Systems, Man and Cybernetics has decided to devote a special issue to 'Causal and Strategic Aspects of Diagnostic Reasoning' for November 1986.

some knowledge level characterisations of the classification problem solving method and its application to diagnosis. For the knowledge engineer, the value of studying abstract organisations of knowledge lies in the possibility of using a single single, well understood, organisation of knowledge as a common framework for a number of similar systems.

If the knowledge engineer is required to elicit and construct an organisation of knowledge without premature commitment to a particular representation scheme then some abstract specification method will be required. The importance of specifying a component of a system independently of its implementation is becoming increasingly appreciated in the discipline of software engineering and has led to the notion of an abstract type specification. In an abstract type specification there is a clear distinction between the properties of the type and incidental characteristics of the representation of that type. For example, the data type `integer' can be described in terms of the set of integers and the operations of addition, subtraction and so on. The mathematician understands this type without understanding anything of the way this type is represented as binary operations on a computer. Our view is that some analogue of the notion of the abstract type specification is required by the knowledge engineer.

There are a number of methods that can be used to specify a type abstractly. One way in which different specification methods can be compared is on a scale of degree of formality. English text, used freely, can be used to produce completely informal specifications. This is a very flexible specification method although specifications can become "long winded" in certain cases and it is difficult to produce English specifications that have a precise and unambiguous meaning. At the other extreme of the scale, formal specifications can

be produced in a formal calculus. One such calculus is the algebraic method, see (Guttag, 1977). An advantage of a formal calculus is that properties of the type can be proved formally. A disadvantage however, is that can be difficult to appreciate the behaviour of a type specified in this way. As a way around this problem, Gehani (1983, p47) advocates the use of both informal and formal specifications.

> Informal and formal specifications complement each other. Informal specifications are easier to read and use while formal specifications offer more precision, more clarity and less ambiguity.
>
> ibid, p47.

This latter approach appears to be promising but clearly further work is necessary in order to determine suitable specification methods for use in knowledge engineering.

In the future, especially with the development of large scale expert system projects, it seems likely that research in the field of software engineering will become increasingly important for knowledge engineering. In addition, the work done on system development methods in the field of knowledge engineering is likely to be useful in software engineering.

REFERENCES

Aikins, J. S., "Prototypical Knowledge for Expert Systems," **Artificial Intelligence**, vol. 20, no. 2, pp. 163-210, 1983.

Anderson, R. H. and J. J. Gillogly, "The Rand Intelligent Terminal Agent (RITA) as a Network Access Aid," **AFIPS Proc. 45**, pp. 501-509, 1976.

Buchanan, B. G., G. L. Sutherland, and E. A. Feigenbaum, "Heuristic DENDRAL: a Program for Generating Explanatory Hypotheses in Organic Chemistry," in **Machine Intelligence 4**, (ed.) B. Meltzer and D. Michie, pp. 209-254, Edinburgh University Press, Edinburgh, Scotland, U.K., 1969.

Buchanan, B. G., D. H. Smith, W. C. White, R. J. Gritter, E. A. Feigenbaum, J. Lederberg, and C. Djerassi, "Applications of Artificial Intelligence for Chemical Inference XXII: Automatic Rule Formation in Mass Spectrometry by means of the META-DENDRAL Program.," **J. of the American Chemical Society**, vol. 96, pp. 6168-6178, 1976.

Buchanan, B. G. and E. A. Feigenbaum, "DENDRAL and META-DENDRAL: Their Applications Dimension," **Artificial Intelligence**, vol. 11, no. 1-2, pp. 5-24, 1978.

Buchanan, B. G., "New Research on Expert Systems," in **Machine Intelligence 10**, (ed.) P. Hayes and D. Michie, pp. 269-299, Ellis Horwood, 1982.

Buchanan, B. G., D. Barstow, R. Bectal, J. Bennett, W. Clancey, C. Kulikowski, T. Mitchell, and D. A. Waterman, "Constructing an Expert System," in **Building Expert Systems**, (ed.) F. Hayes-Roth, D. A. Waterman and D. B. Lenat, Addison-Wesley, 1983.

Chandrasekaran, B. and S. Mittal, "Deep versus Compiled Knowledge Approaches to Diagnostic Problem-Solving," **Int J. Man-Machine Studies**, vol. 19, no. 5, pp. 425-436, 1983.

Clancey, W. J., "Tutoring Rules for Guiding a Case Method Dialogue," **Int. J. Man-Machine Studies**, vol. 11, no. 1, pp. 25-49, 1979.

Clancey, W. J. and R. Letsinger, "NEOMYCIN: Reconfiguring a Rule-based Expert System for Application to Teaching," **Proc. 7th Int. Joint Conf. on Artificial Intelligence**, pp. 829-835, Vancouver, B.C., Canada, 1981.

Clancey, W. J., "The epistemology of a Rule-Based Expert System: a Framework for Explanation," **Artificial Intelligence**, vol. 20, no. 3, pp. 215-251, 1983a.

Clancey, W. J., "Advantages of Abstract Control Knowledge in Expert system Design," **Proc. Conf. American Association for Artificial Intelligence**, pp. 74-78, Washington, D.C., U.S.A., 1983b.

Clancey, W. J., "Classification Problem Solving," **Proc. Conf. American Association for Artificial Intelligence**, pp. 49-55, Austin, Texas, U.S.A., 1984.

Clancey, W. J., "Acquiring, Representing, and Evaluating a Competence Model of Diagnostic Strategy," in **The Nature of Expertise**, (ed.) Chi, Glaser and Farr, forthcoming, 1985.

Davis, R., B. G. Buchanan, and E. H. Shortliffe, "Production Rules as a Representation for a Knowledge-based Consultation System," **Artificial Intelligence**, vol. 8, no. 1, pp. 15-45, 1977.

Davis, R. and J. King, "An Overview of Production Systems," in **Machine Intelligence 8**, (ed.) E.W. Elcock and D. Michie, pp. 300-332, Edinburgh University Press, Edinburgh, 1977.

Davis, R., "Interactive Transfer of Expertise: Acquisition of New Inference Rules," **Artificial Intelligence**, vol. 12, no. 2, pp. 121-157, 1979.

Davis, R., "TEIRESIAS: Applications of Meta-level Knowledge," in **Knowledge-based Systems in Artificial Intelligence**, ed. R. Davis and D. B. Lenat, McGraw-Hill, 1982.

Dennis, J. B., "Modularity," in **Advanced Course in Software Engineering**, (ed.) F. L. Bauer, Springer Verlag, Berlin, Heidelberg and New York, 1973.

Duda, R., J. Gaschnig, and P. Hart, "Model Design the Prospector Consultant System for Mineral Exploration," in **Expert Systems and the Micro-Electronic Age**, (ed.) D. Michie, Edinburgh University Press, Edinburgh, 1979.

Duda, R. O., P. E. Hart, N. J. Nilsson, and G. L. Sutherland, "Semantic Network Representations in Rule-Based Inference System," in **Pattern Directed Inference Systems**, (ed.) D. A. Waterman and F. Hayes-Roth, pp. 203-222, Academic Press, 1978.

Edwards, W., "N=1: Diagnosis in Unique Cases.," in **Computer Diagnosis and Diagnostic Methods**, (ed.) J. A. Jacquez, pp. 139-151, Charles C. Thomas, Springfield, Ill., 1972.

Englemore, R. and H. P. Nii, "A Knowledge-Based System for the Interpretation of Protein X-Ray Crystallographic Data," **Memo HPP-77-2**, Dept. of Computer Science, Stanford, CA., U.S.A., 1977.

Feigenbaum, E. A., "The Art of Artificial Intelligence: I. Themes and Case Studies of Knowledge Engineering," **Proc. 5th Int. Joint Conf. on Artificial Intelligence**, pp. 1014-1029, Cambridge, Mass, U.S.A., 1977.

Feigenbaum, E. A., "Knowledge Engineering: the Applied Side of Artificial Intelligence," **formerly Report No HPP-80-21**, Stanford Heuristic Programming Project, Stanford, CA., U.S.A., 1983.

Gehani, N. H., "Informal and Formal Specifications with Stepwise Refinement," in **Software Engineering Development**, (ed.) P. Wallis, pp. 38-51, Infotech, 1983.

Guttag, J., "Abstract Data Types and the Development of Data Structures," **Comm. ACM**, vol. 20, no. 6, pp. 396-404, June 1977.

Hart, P.E., "Directions for AI in the Eighties," **SIGART Newsletter**, no. 79, pp. 11-16, January 1982.

Johnson, L., "The Need for Competence Models in the Design of Expert Consultant Systems," **Int. J. of Systems Research and Information Science**, vol. 1, no. 1, pp. 23-36, Gordon and Breach, London, 1985.

Kunz, J.C., R.J. Fallat, D. H. McClung, J. J. Osborn, B. A. Votteri, H. P. Nii,, J.S. Aikins, L. M. Fagan, and E. A. Feigenbaum, "PUFF: An Expert System for Interpretation of Pulmonary Function Data," **Report STAN-CS-82-931**, Dept. of Computer Science, Stanford University, 1982.

Lenat, D. B., "An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search," **Ph.D. Thesis, AIM 286, STAN-CS-76-570, HPP-76-8,**, Stanford University, Stanford, CA. U.S.A., 1976.

Lenat, D. B. and G. Harris, "Designing a Rule System that Searches for Scientific Discovery," in **Pattern Directed Inference Systems**, (ed.) D. A. Waterman and F. Hayes-Roth, Academic Press, 1978.

Lesser, V. R. and L. D. Erman, "A Retrospective View of the HEARSAY-II Architecture," **Proc. 5th Int. Joint Conf. on Artificial Intelligence**, pp. 790-800, Cambridge, Mass, U.S.A., 1977.

Marr, D., "Artificial Intelligence: A Personal View," **Artificial Intelligence**, vol. 9, no. 1, pp. 37-48, 1977.

Martin, N., P. Freidland, J. King, and M. Stefick, "Knowledge Base Management for Experiment Planning in Molecular Genetics," **Proc. 5th Int. Joint Conf. on Artificial Intelligence**, pp. 882-887, Cambridge, Mass, U.S.A., 1977.

McDermott, J., "R1: A Rule-Based Configurer of Computer Systems," **Artificial Intelligence**, vol. 19, no. 1, pp. 39-88, 1982.

Michalski, R. S. and R. L. Chilauski, "Knowledge Acquisition by Encoding Expert Rules versus Computer Induction from examples: A case study involving Soybean Pathology," **Int. J. Man-Machine Studies**, vol. 12, no. 1, pp. 63-87, 1980.

Michalski, R S and J. B. Larson, "Selection of the Most Representative Training Examples and Incremental Generation of VL1 Hypotheses: the Underlying Methodology and the Descriptions of Programs ESEL and AQ11.," **Report No. 877**, Department of Computer Science, University of Illinois, Urbana, Illinois, U.S.A., May 1978.

Michie, D., "Problems of the human window," **Proc. Conf. BCS specialist group on Expert Systems**, pp. 1-14, London, 1981.

Michie, D., "High-road and Low-road Programs," **AI Magazine**, vol. 3, no. 1, pp. 21-22, 1981/82.

Michie, D. and R. Michaelson, "Expert Systems in Business," **Datamation**, vol. 29, no. 11, pp. 240-246, 1983.

Michie, D., S. Muggleton, C. Riese, and S. Zubrick, "RuleMaster: A Second Generation Knowledge Engineering Facility," **Radian Technical Report MI-R-623**, Radian Corporation, Austin, Texas, U.S.A., 1984.

Minsky, M., "A Framework for Representing Knowledge," in **The Psychology of Computer Vision**, (ed.) P. H. Winston, pp. 211-277, McGraw Hill, New York, U.S.A., 1975.

Newell, A., "The Knowledge Level," **Artificial Intelligence**, vol. 18, no. 1, pp. 87-127, 1982.

Nii, H. Penny and E. Feigenbaum, "Rule-Based Understanding of Signals," in **Pattern Directed Inference Systems**, (ed.) D. A. Waterman and F. Hayes-Roth, pp. 483-502, Academic Press, 1978.

Parnas, D. L., "On the Criteria to be used in Decomposing Systems into Modules," **Comm. ACM**, vol. 15, no. 12, pp. 1053-1058, 1972.

Quinlan, J. R., "Discovering Rules by Induction from Large Collections of Examples," in **Expert Systems and the Micro-Electronic Age**, (ed.) D. Michie, pp. 168-201, Edinburgh University Press, Edinburgh, Scotland, U.K., 1979.

Reboh, R., "Knowledge Engineering Techniques and Tools for Expert Systems," **Linkoping Studies in Science and Technology, Dissertation**, Software Systems Research Centre, Linkoping University, S-581 83,, Linkoping, Sweden, 1981.

Ritchie, G. D. and F. K. Hanna, "AM: A Case Study in AI Methodology," **Artificial Intelligence**, vol. 23, no. 3, pp. 249-268, 1984.

Rychener, M. D. and A. Newell, "An Instructable Production System: Basic Design Issues," in **Pattern Directed Inference Systems**, (ed.) D. A. Waterman and F. Hayes-Roth, pp. 135-153, Academic Press, 1978.

Shortliffe, E. H., in **Computer-Based Medical Consultation: MYCIN**, Elsivier/North Holland, New York, 1976.

Sticklen, J., B. Chandrasekaran, and J. Josephson, "Control Issues in Classificatory Diagnosis," **Proc. 9th Int. Joint Conf. on Artificial Intelligence**, pp. 300-306, Los Angeles, CA., U.S.A., 1985.

Waterman, D., "Exemplary Programming in RITA," in **Pattern Directed Inference Systems**, (ed.) D. A. Waterman and F. Hayes-Roth, pp. 261-280, Academic Press, 1978.

Wirth, N., "The Module: A System Structuring Facility in High Level Programming Languages," in **Language Design and Programming Methodology**, (ed.) J. M. Tobias, pp. 1-24, Springer Verlag, Berlin, Heidelberg and New York, 1980.

Hayes-Roth, F., "Rule-Based Systems," **Comm. ACM**, vol. 28, no. 6, pp. 921-932, September 1985.

van-Melle, W., "MYCIN: a Knowledge-based Consultation Program for Infectious Disease Diagnosis," **Int. J. Man-Machine Studies**, vol. 10, no. 3, pp. 313-322, 1978.

van-Melle, W., "A Domain-Independent Production System for Consultation Programs," **Proc. 6th Int. Joint. Conf. on Artificial Intelligence**, pp. 923-925, Tokyo, Japan, 1979.

APPENDIX A

GLOSSARY OF TERMS

Abduction. Together with deduction and induction, abduction is a form of inference. The three forms can best be described by a simple illustration.

Given the

General law: All policemen are over five feet
and eight inches in height.

and the

fact: John is a policeman.

we may conclude by DEDUCTION that

Conclusion: John is over five feet and eight inches
in height.

Given that

fact: John is a policeman and over six feet.
fact: Tom is a policeman and over six feet.
fact: Dick is a policeman and over six feet. etc.

we may conclude by INDUCTION that

General law: All policemen are over six feet.

Given that

fact: John is over six feet.
fact: Tom is over six feet.
fact: Dick is over six feet. etc.

we may choose to account for this puzzling
fact by referring to the

General law: All policemen are over five feet
and eight inches tall.

and conclude by ABDUCTION that

fact: John is a policeman.
fact: Tom is a policeman.
fact: Dick is a policeman.

Note that only in deductive inference do we have the case where it cannot be that the premises are true and the conclusion false. Typically, additional criteria are used to support inductive and abductive inferences.

Attribute. A property or feature of an object or problem. Attributes take on values for specific objects. See **object attribute value triple.**

Backward chaining. Abductive inference in a rule-based system. The reasoning strategy is recursive and begins by identifying one or more rules that directly conclude on the goal. If the antecedent of these rules are known then backward chaining ceases and the rules are applied. If the antecedents are not known then these become the new goals of the strategy. If no rules concludes on a goal then the value of that goal is asked of the user. See also **forward chaining.**

Certainty factor. A numerical value, usually between -1 and +1, representing the degree to which a statement holds. Certainty factors may be associated with facts or with rules and are manipulated according to a calculus, usually not a probabilistic calculus.

Compiled knowledge. (Surface knowledge) Knowledge allowing an efficient "short cut" method of reasoning. For example the rule 'if accelerator is depressed then speed increases' is a complied form of the knowledge that the accelerator is opening a valve that lets more air into the engine that ... ; and so on. Experts will often use compiled knowledge unless the particular problem being tackled presents unusual difficulties. See also **deep knowledge.**

Computation. (Computational theory) A computation is a sequence of operations where the operations have the property that they are purely "mechanical" in nature. Each operation is executed in such a way that its execution depends only on the the symbols that are used to describe the operation and not on any interpretation that be given to the description. The vast majority of the operations that one performs in everyday life, for example, following a cooking recipe, driving according to the rules of the road, and so on, are not computational. The successful execution of all these operations depends on the skill and intuitive understanding of the person executing the operations.

Conflict resolution. In a production system where more than one rule is applicable, a conflict resolution strategy is used for deciding which rule to apply.

Context tree. A tree structure that constitutes the working memory of MYCIN and EMYCIN systems. The tree is constructed dynamically during a consultation according to information contained in a static context tree. Each type of object known to the system is a node in the static tree. Associated with each of these nodes is information about the attributes of that object, the rules that conclude on that object and so on.

Control. The strategy used by an inference procedure. Examples include, forward or backward chaining in production systems and various queuing (agenda) disciplines.

Data base. (Working memory) In the context of expert system architectures, the data base holds the current description of the problem. In MYCIN the data base consists of the context tree which contain patient data, facts about organisms and so on.

Deduction. A type of inference, see **abduction.**

Deep knowledge. Knowledge of fundamental facts and theories. This knowledge can be used for reasoning but is usually very redundant in straightforward cases and so expert's learn to skip over the unnecessary steps. In practice, experts are said to use a set of "short cut" methods described as compiled knowledge. See **compiled knowledge.**

Domain. A subject area of knowledge or expertise in which an expert system is designed to operate. In practice a domain can be very narrowly specified, for example `the domain of pulmonary function test interpretation'. See **generic domain.**

Expert. The person who can achieve high levels of performance at a specific task using domain specific knowledge. The knowledge engineer aims to extract this knowledge from the expert and incorporate it into an expert system.

Expert system. A system that contains representations of domain specific expert knowledge that can be used by a user as an "expert consultant" in the solution of some domain specific task.

Extension. A modification to a representation of knowledge to allow the representation of some new knowledge or the deletion of some knowledge. A modification is only an extension if the knowledge provided by the expert can be immediately incorporated into the knowledge-base without the modification of any of the knowledge that is already represented there and without acquiring any further knowledge from the expert.

Generic domain. An abstract domain intended to subsume a collection of related and similar domains. Diagnosis problem solving is an example of a generic domain.

Forward chaining. A deductive inference strategy for the application of production rules. Given some data in the data base (working memory), this data is matched against the RHS, "if-part", of the rules in the rule-base. Any rule that is successfully matched is a candidate for application and a conflict resolution strategy is used to decide which of these rules are applied. The application of rules generally leads to the addition of new data to the data base (working memory). Hence the data in working memory is again matched against the "if-parts" of the rules in the rule-base to repeat the above process.

Frame. An abstract data structure not formally defined but intended to be used to represent the large collection of related facts about a complex real world object or stereotypical situation. Unlike the notion of an "if-then" rule, a frame is a large and richly structured object consequently the computational instantiation of frames is not clear. Perhaps the notion of a frame is best thought of as a conception of the sort of organisation of knowledge that is necessary for inclusion in artificial intelligence programs rather than as a programming construct.

Goal. Some fact to be proved, demonstrated or inferred, usually by a reasoning program.

Goal directed. A system is said to be goal directed if the actions that it performs are selected on the basis that they will contribute to the achievement of a goal. For example, the backward chaining strategy used in a rule representation is goal directed because the strategy selects only those rules that conclude on the goal or intermediate goals.

Heuristic. A rough but ready rule or rule of thumb. For example, 'if the sky is overcast then take an umbrella' is a heuristic. The principle application of heuristics is in reducing the amount of computation that is necessary to find an acceptable solution to a problem. Continuing the above example; to determine the likelyhood of rain, private citizens seldom have the resources that are available to the meteorological office, hence they use a less expensive (and usually less accurate) heuristic rule. A significant part of an expert's knowledge is considered to consist of heuristics.

Induction. A type of inference, see **abduction.**

Inference. The "passage of thought" in reasoning. Abduction, deduction and induction are three types of inference, (see **abduction).**

Inference engine. (Rule interpreter) The classical architectural decomposition of components in an expert system is into three parts, knowledge-base, working memory and inference engine. To solve a problem, the inference engine applies the relevant knowledge from the knowledge-base to the data from the problem which is kept in the working memory. In selecting the knowledge that is relevant to the problem, the inference engine is guided by a control strategy. In rule-based systems the inference engine is a rule interpreter and the knowledge-base is the collection of rules. Forward and backward chaining are the usual control strategies.

Incremental development. (Iterative development) A method of system development that is usually contrasted with the "structured design" methods currently advocated for the construction of large conventional software projects. The construction and testing of a prototype, early on in the project, is a integral part of the iterative development method. In contrast to structured methods, less reliance is placed on the construction of detailed specifications that usually precede programming.

Knowledge acquisition. The process of acquiring knowledge from experts, text books and any other suitable sources. The knowledge is said to have been acquired once it is in a form that can directly be used by an expert system. See **knowledge elicitation.**

Knowledge-base. A representation of knowledge used by an expert system.

Knowledge elicitation. Many authors do not distinguish between knowledge acquisition and elicitation. Knowledge elicitation is also concerned with acquiring knowledge but not necessarily in a form that can be used directly by an expert system. At the beginning of an expert system project it is necessary to conduct some broadly based investigation into the kind of task the system will perform and the sort of knowledge that will be part of the system. At this stage, while the knowledge engineer is "determining the terms of reference" for the project, he or she is doing knowledge elicitation rather than acquisition. See **knowledge acquisition.**

Knowledge engineer. A person who designs and constructs an expert system. The knowledge engineer may be concerned only with knowledge elicitation and acquisition but may also work on the development and engineering of the representation language, programming environment and so on. A knowledge engineer usually works closely with an expert. See **expert.**

LHS. Left Hand Side of a production rule or the "if-part".

Modifiability. The ease with which a representation or program is modified.

Object-attribute-value triple. A common data structure for the representation of facts in an expert system. A particular object may have a number of attributes, each able to take a number of values. For example, the triples (block-A, colour, white) and (block-A, size, small) describe the object block-A as white and small. See **attribute.**

Problem solving. From the perspective of artificial intelligence, problem solving is usually thought of as the search of a space of sequences of operations for a particular sequence that can be used to solve the problem. In connection with expert systems, the term is used to describe the activities of an expert or an expert system while tackling some task.

Production system. A system consisting of a set of rules, a rule interpreter or inference engine and a data base or working memory. An initial set of data is placed in the data base and transformed by the application of individual rules. A rule can only be applied if it matches the data in the data base. See **rule.**

Prototype$_1$. An early version of a system. The function of the prototype is to test ideas and proposed solutions and to provide data for improvements to the system. See **incremental development.**

Prototype$_2$. A stereotypical representative of a class of objects. The prototype birthday party, for example, includes guests and a cake although such things may not be present at all birthday parties.

Representation. An expression of knowledge in a language containing particular "representational devices". These devices are instantiations of general notions such as the notion of a rule, frame, semantic relations and logical expressions.

Representation language. A specific "programming" language for the representation of knowledge. Examples include, OPS5, EMYCIN, KRL, FRL and so on.

RHS. The Right Hand Side of a production rule or the "then-part".

Rule. ("If-then" rule, production rule) A rule is intended to represent a heuristic or rule of thumb although then can of course be used to represent definitional rules of fact. The general form of a rule is usually

```
if  condition-1 is true
and condition-2 is true
and ...
        .
        .
        .
and condition-n is true
then
    suggest or conclude that some fact is true
or
    perform an action or suggest that some action
    should be performed
```

Rule-based system. A system that represents knowledge in the form of rules. The vast majority of expert systems are of this type.

Rule representation scheme. Within this scheme, knowledge is represented as individual rules, each consisting of a prescription for some action to be taken, or conclusion to be made, whenever the situation specified by the rule holds. A rule interpreter or inference engine applies the appropriate rule at a given point in the problem solving activity. See **production system.**

Slot. An entry in a frame. A slot may contain data, another frame or a procedure that can be used to perform an action or calculate a value. Slots often contain default values that are used in the event that no more accurate data is available.

Working memory. See **data base.**

APPENDIX B

THE PROGRAM NAVIGATE-1

NAVIGATE-1 is written in the YAPS[47] production rule language. YAPS is very similar to OPS5, a forward chaining production system. Comparable to OPS5 in terms of efficiency, YAPS allows greater flexibility in the construction of rules.

```
(eval-when (compile) (load 'yaps))

;routes from A

    (p R-AB
          (goal (context is find-route))
          (town start -t1)
          (town destination -t2)
     test (and (= 'A -t1) (= 'B -t2))
     -->  (remove 1)
          (fact goal (context is found-route))
          (msg "route from A to B is
          A ---> B"))

    (p R-AC
          (goal (context is find-route))
          (town start -t1)
          (town destination -t2)
     test (and (= 'A -t1) (= 'C -t2))
     -->  (remove 1)
          (fact goal (context is found-route))
          (msg "route from A to C is
          A ---> B ---> C"))

(p R-AD
          (goal (context is find-route))
          (town start -t1)
          (town destination -t2)
     test (and (= 'A -t1) (= 'D -t2))
     -->  (remove 1)
          (fact goal (context is found-route))
          (msg "route from A to D is
          A ---> B ---> C ---> D"))
```

---

[47] Allen, E. M., "YAPS: Yet Another Production System", Report TR-1146, University of Maryland, Comp. Sci. Dept., Maryland, U.S.A., 1982.

```
(p R-AE
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
 test (and (= 'A -t1) (= 'E -t2))
 -->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from A to E is
      A ---> B ---> C ---> D ---> E"))

(p R-AF
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
 test (and (= 'A -t1) (= 'F -t2))
 -->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from A to F is
      A ---> B ---> F"))

; routes from B

(p R-BA
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
 test (and (= 'B -t1) (= 'A -t2))
 -->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from B to A is
      B ---> A"))

(p R-BC
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
 test (and (= 'B -t1) (= 'C -t2))
 -->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from B to C is
      B ---> C"))

(p R-BD
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
test (and (= 'B -t1) (= 'D -t2))
-->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from B to D is
      B ---> D"))
```

```
(p R-BE
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
 test (and (= 'B -t1) (= 'E -t2))
 -->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from B to E is
      B ---> C ---> E"))

(p R-BF
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
 test (and (= 'B -t1) (= 'F -t2))
 -->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from B to F is
      B ---> F"))

;  routes from C

(p R-CA
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
 test (and (= 'C -t1) (= 'A -t2))
 -->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from C to A is
      C ---> E ---> F ---> B ---> A"))

(p R-CB
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
 test (and (= 'C -t1) (= 'B -t2))
 -->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from C to B is
      C ---> E ---> F ---> B"))

(p R-CD
     (goal (context is find-route))
     (town start -t1)
     (town destination -t2)
test (and (= 'C -t1) (= 'D -t2))
-->  (remove 1)
     (fact goal (context is found-route))
     (msg "route from C to D is
      C ---> D"))
```

```
(p R-CE
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
 test (and (= 'C -t1) (= 'E -t2))
 -->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from C to E is
      C ---> E"))

(p R-CF
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
 test (and (= 'C -t1) (= 'F -t2))
 -->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from C to F is
      C ---> E ---> F"))

;  routes from D

(p R-DA
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
 test (and (= 'D -t1) (= 'A -t2))
 -->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from D to A is
      D ---> E ---> F ---> B ---> A"))

(p R-DB
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
 test (and (= 'D -t1) (= 'B -t2))
 -->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from D to B is
      D ---> E ---> F ---> B"))

(p R-DC
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
test (and (= 'D -t1) (= 'C -t2))
-->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from D to C is
      D ---> E ---> F ---> B ---> C"))
```

```
(p R-DE
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
 test (and (= 'D -t1) (= 'E -t2))
 -->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from D to E is
      D ---> E"))

(p R-DF
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
 test (and (= 'D -t1) (= 'F -t2))
 -->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from D to F is
      D ---> E ---> F"))

;  routes from E

(p R-EA
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
 test (and (= 'E -t1) (= 'A -t2))
 -->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from E to A is
      E ---> F ---> B ---> A"))

(p R-EB
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
 test (and (= 'E -t1) (= 'B -t2))
 -->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from E to B is
      E ---> F ---> B"))

(p R-EC
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
test (and (= 'E -t1) (= 'C -t2))
-->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from E to C is
      E ---> F ---> B ---> C"))
```

```
(p R-ED
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
 test (and (= 'E -t1) (= 'D -t2))
 -->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from E to D is
      E ---> F ---> B ---> C ---> D"))

(p R-EF
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
 test (and (= 'E -t1) (= 'F -t2))
 -->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from E to F is
      E ---> F"))

;  routes from F

(p R-FA
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
 test (and (= 'F -t1) (= 'A -t2))
 -->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from F to A is
      F ---> B ---> A"))

(p R-FB
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
 test (and (= 'F -t1) (= 'B -t2))
 -->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from F to B is
      F ---> B"))

(p R-FC
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
 test (and (= 'F -t1) (= 'C -t2))
 -->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from F to C is
      F ---> B ---> C"))
```

```
(p R-FD
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
 test (and (= 'F -t1) (= 'D -t2))
 -->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from F to D is
      F ---> B ---> C ---> D"))

(p R-FE
      (goal (context is find-route))
      (town start -t1)
      (town destination -t2)
 test (and (= 'F -t1) (= 'E -t2))
 -->  (remove 1)
      (fact goal (context is found-route))
      (msg "route from F to E is
      F ---> B ---> C ---> E"))

(setq data '(
         (goal (context is find-route))
         (town start A)
         (town destination D)
         ))
```

APPENDIX C

THE PROGRAM NAVIGATE-2

NAVIGATE-1 is written in the YAPS[48] production rule language. Very similar to OPS5, YAPS uses a forward chaining rule interpreter. The conflict resolution strategy ensures that the more specific rules (greater number of antecedents) have priority over less specific rules.

```
(eval-when (compile) (load 'yaps))

    (p con-to-A
          (goal (context is find-roads))
          (current-town is -t)
       test (= 'A -t)
       -->  (fact current-town con-to B)
          (remove 1)
          (fact goal (context is find-direction-roads)))

    (p con-to-B
          (goal (context is find-roads))
          (current-town is -t)
       test (= 'B -t)
       -->  (fact current-town con-to A)
          (fact current-town con-to C)
          (fact current-town con-to F)
          (remove 1)
          (fact goal (context is find-direction-roads)))

(p con-to-C
          (goal (context is find-roads))
          (current-town is -t)
    test (= 'C -t)
    -->  (fact current-town con-to D)
          (remove 1)
          (fact goal (context is find-direction-roads)))

(p con-to-D
          (goal (context is find-roads))
          (current-town is -t)
    test (= 'D -t)
    -->  (fact current-town con-to E)
          (remove 1)
          (fact goal (context is find-direction-roads)))
```

---

[48] Allen, E. M., "YAPS: Yet Another Production System", Report TR-1146, University of Maryland, Comp. Sci. Dept., Maryland, U.S.A., 1982.

```
(p direction-AB
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'A -ct) (= 'B -ft))
   -->  (fact direction B 300)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-AC
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'A -ct) (= 'C -ft))
   -->  (fact direction C 350)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-AD
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'A -ct) (= 'D -ft))
   -->  (fact direction D 330)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-AE
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'A -ct) (= 'E -ft))
   -->  (fact direction E 320)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-AF
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'A -ct) (= 'F -ft))
   -->  (fact direction F 290)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-BA
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
test (and (= 'B -ct) (= 'A -ft))
-->  (fact direction A 120)
        (remove 1 3)
        (fact goal (context is found-direction)))
```

```
(p direction-BC
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'B -ct) (= 'C -ft))
   -->  (fact direction C 60)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-BD
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'B -ct) (= 'D -ft))
   -->  (fact direction D 0)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-BE
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'B -ct) (= 'E -ft))
   -->  (fact direction E 0)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-BF
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'B -ct) (= 'F -ft))
   -->  (fact direction F 290)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-CA
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'C -ct) (= 'A -ft))
   -->  (fact direction A 180)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-CB
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
test (and (= 'C -ct) (= 'B -ft))
-->  (fact direction B 210)
     (remove 1 3)
     (fact goal (context is found-direction)))
```

```
(p direction-CD
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'C -ct) (= 'D -ft))
   -->  (fact direction D 310)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-CE
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'C -ct) (= 'E -ft))
   -->  (fact direction E 290)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-CF
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'C -ct) (= 'F -ft))
   -->  (fact direction F 270)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-DA
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'D -ct) (= 'A -ft))
   -->  (fact direction A 140)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-DB
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'D -ct) (= 'B -ft))
   -->  (fact direction B 180)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-DC
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
test (and (= 'D -ct) (= 'C -ft))
-->  (fact direction C 130)
        (remove 1 3)
        (fact goal (context is found-direction)))
```

```
(p direction-DE
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'D -ct) (= 'E -ft))
   -->  (fact direction E 180)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-DF
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'D -ct) (= 'F -ft))
   -->  (fact direction F 200)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-EA
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'E -ct) (= 'A -ft))
   -->  (fact direction A 130)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-EB
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'E -ct) (= 'B -ft))
   -->  (fact direction B 180)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-EC
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'E -ct) (= 'C -ft))
   -->  (fact direction C 110)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-ED
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
test (and (= 'E -ct) (= 'D -ft))
-->  (fact direction D 0)
        (remove 1 3)
        (fact goal (context is found-direction)))
```

```
(p direction-EF
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'E -ct) (= 'F -ft))
   -->  (fact direction F 250)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-FA
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'F -ct) (= 'A -ft))
   -->  (fact direction A 110)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-FB
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'F -ct) (= 'B -ft))
   -->  (fact direction B 110)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-FC
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'F -ct) (= 'C -ft))
   -->  (fact direction C 90)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-FD
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
   test (and (= 'F -ct) (= 'D -ft))
   -->  (fact direction D 20)
        (remove 1 3)
        (fact goal (context is found-direction)))

(p direction-FE
        (goal (context is find-direction))
        (current-town is -ct)
        (far-town is -ft)
test (and (= 'F -ct) (= 'E -ft))
-->  (fact direction E 70)
        (remove 1 3)
        (fact goal (context is found-direction)))
```

```
;The facilities of the YAPS rule language do not allow R3, as
;described in chapter four, to be implemented as a single rule.

(p R3-A
        (goal (context is find-direction-roads))
        (current-town con-to -t)
   test (= 'A -t)
   -->  (fact far-town is A)
        (remove 1 2)
        (fact goal (context is find-direction)))

(p R3-B
        (goal (context is find-direction-roads))
        (current-town con-to -t)
   test (= 'B -t)
   -->  (fact far-town is B)
        (remove 1 2)
        (fact goal (context is find-direction)))

(p R3-C
        (goal (context is find-direction-roads))
        (current-town con-to -t)
   test (= 'C -t)
   -->  (fact far-town is C)
        (remove 1 2)
        (fact goal (context is find-direction)))

(p R3-D
        (goal (context is find-direction-roads))
        (current-town con-to -t)
   test (= 'D -t)
   -->  (fact far-town is D)
        (remove 1 2)
        (fact goal (context is find-direction)))

(p R3-E
        (goal (context is find-direction-roads))
        (current-town con-to -t)
   test (= 'E -t)
   -->  (fact far-town is E)
        (remove 1 2)
        (fact goal (context is find-direction)))

(p R3-F
        (goal (context is find-direction-roads))
        (current-town con-to -t)
   test (= 'F -t)
   -->  (fact far-town is F)
        (remove 1 2)
        (fact goal (context is find-direction)))

(p R4-more-roads
        (goal (context is found-direction))
        (current-town con-to -t)
    --> (remove 1)
        (fact goal (context is find-direction-roads)))
```

```
(p R4-no-more-roads
        (goal (context is found-direction))
    --> (remove 1)
        (fact goal (context is find-destination-direction)))

(p R4-found-destination-direction
        (goal (context is found-direction))
        (direction -td -d)
        (destination is -td)
    --> (remove 1)
        (fact goal (context is find-best-road)))
```

```
(p R5
        (goal (context is find-destination-direction))
        (destination is -t)
   test (= 'D -t)
   -->  (fact far-town is D)
        (remove 1)
        (fact goal (context is find-direction)))

;The facilities of the YAPS rule language do not allow R6, as
;described in chapter four, to be implemented as a single rule.
```

```
(p R6-1
        (goal (context is find-best-road))
        (direction -t1 -d1)
        (direction -t2 -d2)
        (direction -td -dd)
        (destination is -td)
   test (and (<= (leastangle(modulus (- -dd -d1)))
                 (leastangle(modulus (- -dd -d2))))
         (<> -t1 -t2)
         (<> -t1 -td)
         (<> -t2 -td))
    --> (remove 1 3)
        (fact goal (context is find-best-road)))

(p R6-2
        (goal (context is find-best-road))
        (direction -t1 -d1)
        (direction -t2 -d2)
        (direction -td -dd)
        (destination is -td)
   test (and (> (leastangle(modulus (- -dd -d1)))
                (leastangle(modulus (- -dd -d2))))
         (<> -t1 -t2)
         (<> -t1 -td)
         (<> -t2 -td))
    --> (remove 1 2)
        (fact goal (context is find-best-road)))

(p R6-3
        (goal (context is find-best-road))
    --> (remove 1)
        (fact goal (context is add-road-route)))
```

```
(p R6-add-route-moreroads
        (goal (context is add-road-route))
        (current-town is -ct)
        (direction -t -d)
        (direction -td -dd)
        (destination is -td)
   test (<> -t -td)
    --> (remove 1 2 3 4)
        (fact road-on-route start -ct)
        (fact road-on-route end -t)
        (fact current-town is -t)
        (fact goal (context is check-route-complete)))

;For the last road in a route there is only one direction
;fact in the data base.

(p R6-add-route-lastroad
        (goal (context is add-road-route))
        (current-town is -ct)
        (direction -td -dd)
        (destination is -td)
    --> (remove 1 2 3)
        (fact road-on-route start -ct)
        (fact road-on-route end -td)
        (fact current-town is -td)
        (fact goal (context is check-route-complete)))

(p R7-complete
        (goal (context is check-route-complete))
        (current-town is -ct)
        (destination is -td)
   test (= -ct -td)
    --> (remove 1)
        (fact goal (context is route-complete)))

(p R8-not-complete
        (goal (context is check-route-complete))
        (current-town is -ct)
        (destination is -td)
   test (<> -ct -td)
    --> (remove 1)
        (fact goal (context is find-roads)))

(setq data '(
         (goal (context is find-roads))
         (current-town is A)
         (destination is D)
         ))

(def modulus
     (lambda (x)
             (cond ((>= x 0) x)
                   (t  (- 0 x))
             )))
```

```
(def leastangle
     (lambda (x)
             (cond ((<= x 180) x)
                   (t  (- 360 x))
             )))
```

The following figure shows the flow of control from 'context' to 'context'. Initially the system is in the 'find-roads' 'context'.

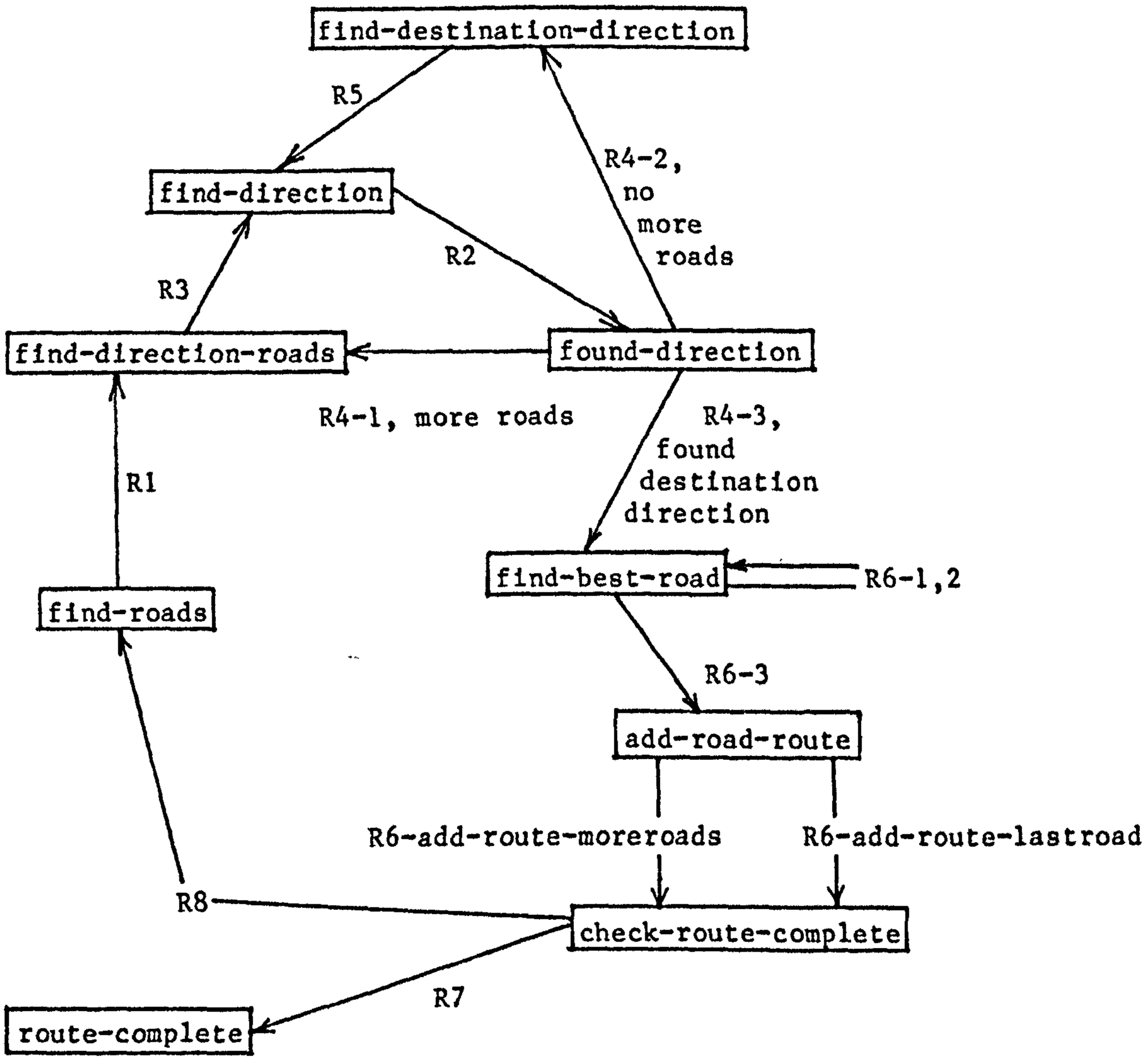


fig C.1

The flow of control in NAVIGATE-2

The following is an example of the operation of NAVIGATE-2 as it finds a route from A to D.

```
(loadfacts data)

  Adding fact    : 1. (goal (context is find-roads))
  Adding fact    : 2. (current-town is A)
Running rule con-to-A
        Facts used:
                1. (goal (context is find-roads))
                2. (current-town is A)
  Adding fact    : 3. (current-town con-to B)
  Removing fact  : 1. (goal (context is find-roads))
```

```
  Adding fact    : 4. (goal (context is find-direction-roads))

Running rule R3-B
        Facts used:
                4. (goal (context is find-direction-roads))
                3. (current-town con-to B)
  Adding fact    : 5. (far-town is B)
  Removing fact  : 4. (goal (context is find-direction-roads))
  Removing fact  : 3. (current-town con-to B)
  Adding fact    : 6. (goal (context is find-direction))

Running rule direction-AB
        Facts used:
                6. (goal (context is find-direction))
                2. (current-town is A)
                5. (far-town is B)
  Adding fact    : 7. (direction B 300)
  Removing fact  : 6. (goal (context is find-direction))
  Removing fact  : 5. (far-town is B)
  Adding fact    : 8. (goal (context is found-direction))

Running rule R4-no-more-roads
        Facts used:
                8. (goal (context is found-direction))
  Removing fact  : 8. (goal (context is found-direction))
  Adding fact    : 9. (goal (context is find-destination-direction))

No rules currently in the conflict set
  Adding fact    : 10. (destination is D)

Running rule R5
        Facts used:
                9. (goal (context is find-destination-direction))
                10. (destination is D)
  Adding fact    : 11. (far-town is D)
  Removing fact  : 9. (goal (context is find-destination-direction))
  Adding fact    : 12. (goal (context is find-direction))

Running rule direction-AD
        Facts used:
                12. (goal (context is find-direction))
                2. (current-town is A)
                11. (far-town is D)
  Adding fact    : 13. (direction D 330)
  Removing fact  : 12. (goal (context is find-direction))
  Removing fact  : 11. (far-town is D)
  Adding fact    : 14. (goal (context is found-direction))

Running rule R4-found-destination-direction
        Facts used:
                14. (goal (context is found-direction))
                13. (direction D 330)
                10. (destination is D)
  Removing fact  : 14. (goal (context is found-direction))
  Adding fact    : 15. (goal (context is find-best-road))

Running rule R6-3
        Facts used:
                15. (goal (context is find-best-road))
  Removing fact  : 15. (goal (context is find-best-road))
```

```
  Adding fact    : 16. (goal (context is add-road-route))

Running rule R6-add-route-moreroads
        Facts used:
                16. (goal (context is add-road-route))
                2. (current-town is A)
                7. (direction B 300)
                13. (direction D 330)
                10. (destination is D)
  Removing fact  : 16. (goal (context is add-road-route))
  Removing fact  : 2. (current-town is A)
  Removing fact  : 7. (direction B 300)
  Removing fact  : 13. (direction D 330)
  Adding fact    : 17. (road-on-route start A)
  Adding fact    : 18. (road-on-route end B)
  Adding fact    : 19. (current-town is B)
  Adding fact    : 20. (goal (context is check-route-complete))

Running rule R8-not-complete
        Facts used:
                20. (goal (context is check-route-complete))
                19. (current-town is B)
                10. (destination is D)
  Removing fact  : 20. (goal (context is check-route-complete))
  Adding fact    : 21. (goal (context is find-roads))

Running rule con-to-B
        Facts used:
                21. (goal (context is find-roads))
                19. (current-town is B)
  Adding fact    : 22. (current-town con-to A)
  Adding fact    : 23. (current-town con-to C)
  Adding fact    : 24. (current-town con-to F)
  Removing fact  : 21. (goal (context is find-roads))
  Adding fact    : 25. (goal (context is find-direction-roads))

Running rule R3-F
        Facts used:
                25. (goal (context is find-direction-roads))
                24. (current-town con-to F)
  Adding fact    : 26. (far-town is F)
  Removing fact  : 25. (goal (context is find-direction-roads))
  Removing fact  : 24. (current-town con-to F)
  Adding fact    : 27. (goal (context is find-direction))

Running rule direction-BF
        Facts used:
                27. (goal (context is find-direction))
                19. (current-town is B)
                26. (far-town is F)
  Adding fact    : 28. (direction F 290)
  Removing fact  : 27. (goal (context is find-direction))
  Removing fact  : 26. (far-town is F)
  Adding fact    : 29. (goal (context is found-direction))

Running rule R4-more-roads
        Facts used:
                29. (goal (context is found-direction))
                23. (current-town con-to C)
```

```
  Removing fact  : 29. (goal (context is found-direction))
  Adding fact    : 30. (goal (context is find-direction-roads))

Running rule R3-C
        Facts used:
                30. (goal (context is find-direction-roads))
                23. (current-town con-to C)
  Adding fact    : 31. (far-town is C)
  Removing fact  : 30. (goal (context is find-direction-roads))
  Removing fact  : 23. (current-town con-to C)
  Adding fact    : 32. (goal (context is find-direction))

Running rule direction-BC
        Facts used:
                32. (goal (context is find-direction))
                19. (current-town is B)
                31. (far-town is C)
  Adding fact    : 33. (direction C 60)
  Removing fact  : 32. (goal (context is find-direction))
  Removing fact  : 31. (far-town is C)
  Adding fact    : 34. (goal (context is found-direction))

Running rule R4-more-roads
        Facts used:
                34. (goal (context is found-direction))
                22. (current-town con-to A)
  Removing fact  : 34. (goal (context is found-direction))
  Adding fact    : 35. (goal (context is find-direction-roads))

Running rule R3-A
        Facts used:
                35. (goal (context is find-direction-roads))
                22. (current-town con-to A)
  Adding fact    : 36. (far-town is A)
  Removing fact  : 35. (goal (context is find-direction-roads))
  Removing fact  : 22. (current-town con-to A)
  Adding fact    : 37. (goal (context is find-direction))

Running rule direction-BA
        Facts used:
                37. (goal (context is find-direction))
                19. (current-town is B)
                36. (far-town is A)
  Adding fact    : 38. (direction A 120)
  Removing fact  : 37. (goal (context is find-direction))
  Removing fact  : 36. (far-town is A)
  Adding fact    : 39. (goal (context is found-direction))

Running rule R4-no-more-roads
        Facts used:
                39. (goal (context is found-direction))
  Removing fact  : 39. (goal (context is found-direction))
  Adding fact    : 40. (goal (context is find-destination-direction))

Running rule R5
        Facts used:
                40. (goal (context is find-destination-direction))
                10. (destination is D)
  Adding fact    : 41. (far-town is D)
```

```
  Removing fact  : 40. (goal (context is find-destination-direction))
  Adding fact    : 42. (goal (context is find-direction))

Running rule direction-BD
        Facts used:
                42. (goal (context is find-direction))
                19. (current-town is B)
                41. (far-town is D)
  Adding fact    : 43. (direction D 0)
  Removing fact  : 42. (goal (context is find-direction))
  Removing fact  : 41. (far-town is D)
  Adding fact    : 44. (goal (context is found-direction))

Running rule R4-found-destination-direction
        Facts used:
                44. (goal (context is found-direction))
                43. (direction D 0)
                10. (destination is D)
  Removing fact  : 44. (goal (context is found-direction))
  Adding fact    : 45. (goal (context is find-best-road))

Running rule R6-1
        Facts used:
                45. (goal (context is find-best-road))
                33. (direction C 60)
                38. (direction A 120)
                43. (direction D 0)
                10. (destination is D)
  Removing fact  : 45. (goal (context is find-best-road))
  Removing fact  : 38. (direction A 120)
  Adding fact    : 46. (goal (context is find-best-road))

Running rule R6-1
        Facts used:
                46. (goal (context is find-best-road))
                33. (direction C 60)
                28. (direction F 290)
                43. (direction D 0)
                10. (destination is D)
  Removing fact  : 46. (goal (context is find-best-road))
  Removing fact  : 28. (direction F 290)
  Adding fact    : 47. (goal (context is find-best-road))

Running rule R6-3
        Facts used:
                47. (goal (context is find-best-road))
  Removing fact  : 47. (goal (context is find-best-road))
  Adding fact    : 48. (goal (context is add-road-route))

Running rule R6-add-route-moreroads
        Facts used:
                48. (goal (context is add-road-route))
                19. (current-town is B)
                33. (direction C 60)
                43. (direction D 0)
                10. (destination is D)
  Removing fact  : 48. (goal (context is add-road-route))
  Removing fact  : 19. (current-town is B)
  Removing fact  : 33. (direction C 60)
```

```
  Removing fact  : 43. (direction D 0)
  Adding fact    : 49. (road-on-route start B)
  Adding fact    : 50. (road-on-route end C)
  Adding fact    : 51. (current-town is C)
  Adding fact    : 52. (goal (context is check-route-complete))

Running rule R8-not-complete
        Facts used:
                52. (goal (context is check-route-complete))
                51. (current-town is C)
                10. (destination is D)
  Removing fact  : 52. (goal (context is check-route-complete))
  Adding fact    : 53. (goal (context is find-roads))

Running rule con-to-C
        Facts used:
                53. (goal (context is find-roads))
                51. (current-town is C)
  Adding fact    : 54. (current-town con-to D)
  Removing fact  : 53. (goal (context is find-roads))
  Adding fact    : 55. (goal (context is find-direction-roads))

Running rule R3-D
        Facts used:
                55. (goal (context is find-direction-roads))
                54. (current-town con-to D)
  Adding fact    : 56. (far-town is D)
  Removing fact  : 55. (goal (context is find-direction-roads))
  Removing fact  : 54. (current-town con-to D)
  Adding fact    : 57. (goal (context is find-direction))

Running rule direction-CD
        Facts used:
                57. (goal (context is find-direction))
                51. (current-town is C)
                56. (far-town is D)
  Adding fact    : 58. (direction D 310)
  Removing fact  : 57. (goal (context is find-direction))
  Removing fact  : 56. (far-town is D)
  Adding fact    : 59. (goal (context is found-direction))

Running rule R4-found-destination-direction
        Facts used:
                59. (goal (context is found-direction))
                58. (direction D 310)
                10. (destination is D)
  Removing fact  : 59. (goal (context is found-direction))
  Adding fact    : 60. (goal (context is find-best-road))

Running rule R6-3
        Facts used:
                60. (goal (context is find-best-road))
  Removing fact  : 60. (goal (context is find-best-road))
  Adding fact    : 61. (goal (context is add-road-route))

Running rule R6-add-route-lastroad
        Facts used:
                61. (goal (context is add-road-route))
                51. (current-town is C)
```

```
                58. (direction D 310)
                10. (destination is D)
  Removing fact  : 61. (goal (context is add-road-route))
  Removing fact  : 51. (current-town is C)
  Removing fact  : 58. (direction D 310)
  Adding fact    : 62. (road-on-route start C)
  Adding fact    : 63. (road-on-route end D)
  Adding fact    : 64. (current-town is D)
  Adding fact    : 65. (goal (context is check-route-complete))

Running rule R7-complete
        Facts used:
                65. (goal (context is check-route-complete))
                64. (current-town is D)
                10. (destination is D)
  Removing fact  : 65. (goal (context is check-route-complete))
  Adding fact    : 66. (goal (context is route-complete))

No rules currently in the conflict set
nil

Facts in db
Cycle   Fact
10.     (destination is D)
17.     (road-on-route start A)
18.     (road-on-route end B)
49.     (road-on-route start B)
50.     (road-on-route end C)
62.     (road-on-route start C)
63.     (road-on-route end D)
64.     (current-town is D)
66.     (goal (context is route-complete))
nil
```

APPENDIX D

MICRO-ORGANISMS-1

MICRO-ORGANISMS-1 is written in the KES expert system building shell, a product of Software Architecture and Engineering Inc. KES is based on a system developed by Reggia[49] for the convenient construction of knowledge-based support systems. In the production system component of KES, facts are represented as attribute-value pairs and the production rules are backward chained.

```
attributes:

\ Name, type and values of all attributes.

    identity of organism (mlt):
                        pseudomonas,
                        klebsiella.

    stain of organism (sgl):
                        gram neg,
                        gram pos.

    morphology of organism (sgl):
                        rod,
                        coccus.

    pathogeny of organism (mlt):
                        high,
                        moderate,
                        low.

    injury of patient (mlt):
                        burns,
                        broken bones,
                        none.

    site of culture (sgl):
                        throat,
                        skin
%

rules:

 R1
    if stain of organism = gram neg,
    &  morphology of organism = rod,
    &  injury of patient = burns
    then
       identity of organism = pseudomonas <0.6>.
```

---

[49] Reggia, James A., "Knowledge-Based Decision Support Systems: Development Through KMS", TR-1121, University of Maryland, Comp. Sci. Dept., Maryland, U.S.A., 1981.

```
R2
   if stain of organism = gram neg,
   &  morphology of organism = rod,
   &  site of culture = throat
   then
      identity of organism = klebsiella <0.5>.

R3
   if identity of organism = pseudomonas
   then
      pathogeny of organism = high <0.7>.

 R4
    if identity of organism = klebsiella
    then
       pathogeny of organism = high <0.4>.

 R5
    if stain of organism = gram neg,
    &  morphology of organism = rod
    then
       pathogeny of organism = high <0.8>
%

actions:

    obtain pathogeny of organism.

    if status (pathogeny of organism) = known
    then
       message " "
          "The pathogeny of the organism is "
           display values (pathogeny of organism)
    else
       message " "
          "The pathogeny of the organism is unknown."
%
```

APPENDIX E

MICRO-ORGANISMS-2

MICRO-ORGANISMS-2 is written in the KES expert system building shell, a product of Software Architecture and Engineering Inc. In the production system component of KES, facts are represented as attribute-value pairs and the production rules are backward chained.

```
attributes:

\ Name, type and values of all attributes.

    identity of organism (mlt):
                        pseudomonas,
                        klebsiella.

    identity of category (mlt):
                        gram pos rods,
                        gram neg rods.

    stain of organism (sgl):
                        gram neg,
                        gram pos.

    morphology of organism (sgl):
                        rod,
                        coccus.

pathogeny of organism (mlt):
                        high,
                        moderate,
                        low.

pathogeny of category (mlt):
                        high,
                        moderate,
                        low.

aerobicity of organism (mlt):
                        aerobic,
                        anaerobic.

aerobicity of category (mlt):
                        aerobic,
                        anaerobic.

injury of patient (mlt):
                        burns,
                        broken bones,
                        none.

site of culture (sgl):
                        throat,
                        skin.
```

```
    inherit properties (sgl):
                        valid,
                        invalid
%

rules:

 R1'
     if identity of category = gram neg rods
     &  injury of patient = burns
    then
       identity of organism = pseudomonas <0.6>.

R2'
     if identity of category = gram pos rods
     &  site of culture = throat
    then
       identity of organism = klebsiella <0.5>.

R3
    if identity of organism = pseudomonas
    then
       pathogeny of organism = high <0.7>.

R4
    if identity of organism = klebsiella
    then
       pathogeny of organism = high <0.4>.

R6
    if stain of organism = gram neg,
    &  morphology of organism = rod,
    then
       identity of category = gram neg rods <1.0>.

R7
    if stain of organism = gram pos,
    &  morphology of organism = rod,
    then
       identity of category = gram pos rods <1.0>.

R8
     if identity of category = gram neg rods
    then
       pathogeny of category = high <0.8>.

R9
     if identity of category = gram pos rods
    then
       pathogeny of category = moderate <0.4>.
```

```
\ Due to the lack of facilities in the KES production rule
\ language, R10 as described in chapter four cannot be
\ implemented as a single rule.  The rules R10-1, R10-2, etc.
\ achieve the effect of the single rule R10.

 R10-1
     if status (identity of organism) = unknown
    then
        inherit properties = valid.

 R10-2
     if pathogeny of category = high,
     &  inherit properties = valid
    then
        pathogeny of category = high <1.0>.

 R10-3
     if pathogeny of category = moderate,
     &  inherit properties = valid
    then
        pathogeny of category = moderate <1.0>.

\ and other rules as necessary.

 R10-4
     if aerobicity of category = aerobic,
     &  inherit properties = valid
    then
        aerobicity of category = aerobic <1.0>.

 R10-5
     if aerobicity of category = anaerobic,
     &  inherit properties = valid
    then
        aerobicity of category = anaerobic <1.0>.

\ and other rules as necessary.

 R11
     if identity of organism = pseudomonas
     then
        aerobicity of organism = aerobic <0.8>.

 R12
     if identity of category = gram neg rods
     then
        aerobicity of category =  aerobic <0.6>
%
```

```
actions:

    obtain pathogeny of organism.

    if status (pathogeny of organism) = known
    then
       message " "
          "The pathogeny of the organism is "
           display values (pathogeny of organism)
    else
       message " "
          "The pathogeny of the organism is unknown."
%
```

The behaviour of MICRO-ORGANISMS-2 can best be appreciated from the following figure.

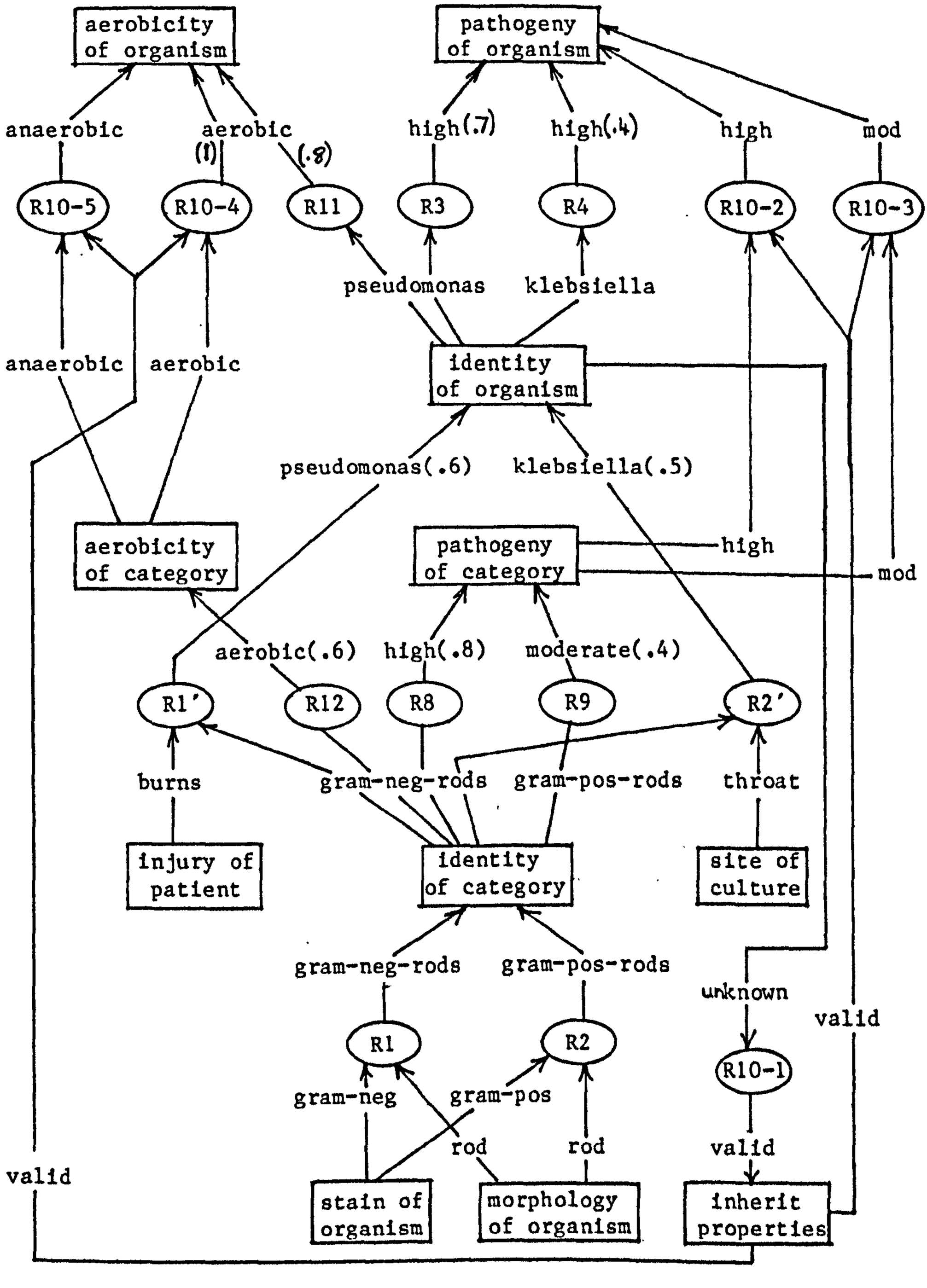


fig E.1

The rules of MICRO-ORGANISMS-2

APPENDIX F

NAVIGATE-3

NAVIGATE-3 is implemented in the ML[50] functional language. ML is a statically-scoped, strongly typed language. Functions can be passed as parameters, returned as values and form parts of data structures. This latter facility is very convenient for the implementation of frames with slots containing procedures. The strong typing system guarantees that no type errors can occur at run time.

```
{TYPES}

type townn = A|B|C|D|E|F;

        {The names of the six towns on the map}

type roadn = r1|r2|r3|r4|r5|r6|r7|r8;

        {The names of the eight roads on the map}

type town = mtown of (townn * int * int * roadn list *
                      (townn*int)list);

        {A town consists of the following:}
        {a) A list of roads that are connected to the town}
        {c) A list of directions (an angle from 0 to 360}
        {degrees to each of the other towns on the map}

type road = mroad of
    (roadn * int * townn * townn * int * townn * townn * int);

        {A road consists of the following:}
        {a) The name of the road}
        {b) The length of the road}
        {c) The two directions of travel along the road.}
        {Each direction is represented as a beginning town,}
        {an end town and the direction in degrees from the}
        {beginning town to the end town.}

type mapp = mmap of (town list * road list);

        {A mapp is a list of towns and a list of roads.}

type stage = mstage of (roadn*townn*townn);

        {The route from one town to another consists of a}
        {sequence of stages.  Each stage consists of a}
        {roadname.  As all roads can be travelled in two}
        {directions a stage also contains the name of the}
        {town at the beginning of the road and the name of}
        {the town at the end.}
```

---

[50] Cardelli, L., **ML under UNIX**, Bell Laboratories, Murray Hill, New Jersey, U.S.A., 1983.

```
{DATA REPRESENTING THE MAP}

val tA = mtown (A, 0,2,  [r1; r2], [(B, 45); (C, 20); (D, 346);
                 (E, 15); (F, 20)]);
val tB = mtown (B, 4,6,  [r1; r3], [(A, 225); (C, 0); (D, 300);
                 (E, 5); (F, 20)]);
val tC = mtown (C, 8,6,  [r3; r4; r5; r6], [(A, 200); (B, 180);
                 (D, 270); (E, 9); (F, 45)]);
val tD = mtown (D, 8,0,  [r2; r4; r7], [(A, 166); (B, 130);
                 (C, 90); (E, 49); (F, 80)]);
val tE = mtown (E, 14,7, [r5; r7; r8], [(A, 195); (B, 185);
                 (C, 189); (D, 229); (F, 150)]);
val tF = mtown (F, 11,9, [r6; r8], [(A, 200); (B, 200); (C, 229);
                 (D, 260); (E, 330)]);

val rd1 = mroad (r1, 5, A,B,  45, B,A, 225);
val rd2 = mroad (r2, 8, A,D, 346, D,A, 166);
val rd3 = mroad (r3, 4, B,C,   0, C,B, 180);
val rd4 = mroad (r4, 5, C,D, 270, D,C,  90);
val rd5 = mroad (r5, 6, C,E,   9, E,C, 189);
val rd6 = mroad (r6, 4, C,F,  45, F,C, 225);
val rd7 = mroad (r7, 9, D,E,  49, E,D, 229);
val rd8 = mroad (r8, 3, E,F, 150, F,E, 330);

val m = mmap ([tA; tB; tC; tD; tE; tF],
              [rd1; rd2; rd3; rd4; rd5; rd6; rd7; rd8]);

val townpairs = [(A,B);(A,C);(A,D);(A,E);(A,F);
                 (B,A);(B,C);(B,D);(B,E);(B,F);
                 (C,A);(C,B);(C,D);(C,E);(C,F);
                 (D,A);(D,B);(D,C);(D,E);(D,F);
                 (E,A);(E,B);(E,C);(E,D);(E,F);
                 (F,A);(F,B);(F,C);(F,D);(F,E)];

        {Test data, town1 never equals town2}

{MAP DESTRUCTORS}

        {map}

val townsmap (mmap(tns, rds)) = tns;
val roadsmap (mmap(tns, rds)) = rds;

        {town}

val townntown (mtown(tn, x,y, rl, dl)) = tn:townn;
val loctntown (mtown(tn, x,y, rl, dl)) = (x,y):int*int;
val roadstown (mtown(tn, x,y, rl, dl)) = rl:roadn list;
val dircstown (mtown(tn, x,y, rl, dl)) = dl:(townn*int) list;
```

```
        {road}

val roadnroad (mroad(rn, l, t1, t2, d1, t3, t4, d2)) = rn:roadn;
val lengthroad (mroad(rn, l, t1, t2, d1, t3, t4, d2)) = l:int;
val town1road (mroad(rn, l, t1, t2, d1, t3, t4, d2)) = t1:townn;
val town2road (mroad(rn, l, t1, t2, d1, t3, t4, d2)) = t2:townn;
val dir1road   (mroad(rn, l, t1, t2, d1, t3, t4, d2)) = d1:int;
val town3road (mroad(rn, l, t1, t2, d1, t3, t4, d2)) = t3:townn;
val town4road (mroad(rn, l, t1, t2, d1, t3, t4, d2)) = t4:townn;
val dir2road   (mroad(rn, l, t1, t2, d1, t3, t4, d2)) = d2:int;

        {stage}

val roadnway (mstage(n, t1, t2)) = n;
val t1way (mstage(n, t1, t2)) = t1;
val t2way (mstage(n, t1, t2)) = t2;

val findtown (t:townn):town = fdtn (t, townsmap m);

        {Given the name of a town findtown returns that town}

val rec fdtn (t:townn, l:town list):town =
          if null l then escape "notowns"
          else
          if t = townntown(hd l) then hd l
          else fdtn (t, tl l);

val findroad (r:roadn):road = fdrd (r, roadsmap m);

        {Analogous to findtown}

val rec fdrd (r:roadn, l:road list):road =
          if null l then escape "noroads"
          else
          if r = roadnroad(hd l) then hd l
          else fdrd (r, tl l);

val modulus (a:int):int = if a < 0 then 0-a else a;

val langle (a:int):int = if a <= 180 then a else 360-a;

val p22 (x,y) = y;

val rec member a = fun [].false |
                   (b::x).if a = b then true
                                   else member a x;

{member a l is true if a is present in the list l}
{and false otherwise}
```

```
val inc (x, y) = ic (x, y, []);

        {inc of two lists x and y returns a list of all the}
        {elements that belong to both lists, i.e. the}
        {incidence of x and y}

val rec ic (x,y,ans) = if null x then ans
                       else
                       if null y then ans
                       else
                       if member (hd x) y
                       then ic (tl x, y, (hd x)::ans)
                            else ic (tl x, y, ans);

val adj (t1:townn, t2:townn):bool =
        if t1 = t2 then escape "adjtownsequal"
        else
        if null (inc (roadstown (findtown t1),
                      roadstown (findtown t2))) then false
                                                else true;

        {Two towns are adjacent if they are linked by a}
        {single road, i.e. no intervening roads.}

{STAGE CONSTRUCTING FUNCTIONS}

val waytowns (t1:townn, t2:townn):stage =
             if t1 = t2 then escape "waytownsequal"
             else
                  mstage (hd (inc (roadstown (findtown t1),
                                   roadstown (findtown t2))),
                          t1, t2) ? escape "nowaytowns";


        {waytowns constructs a route stage from two}
        {adjacent towns.  The road part of the stage is an}
        {element of the roads joined to both towns.}

val wayroadtown = fun (t:townn).
                      fun (r:roadn).
                          {if road not joined to town error}
                          mstage (r, t,
                                  if town1road (findroad r) = t
                                  then town2road(findroad r)
                                  else town1road(findroad r)
                                  );

        {Given a townn wayroadtown returns a function.}
        {This function takes a roadn and then returns a}
        {route stage from the given town along the given}
        {road to the town at the other end of the road.}

val waysfrom (t:townn):stage list =
             map (wayroadtown t) (roadstown (findtown t));

        {Produces all the stages that start at a given town}
```

```
val sametown (t:townn, (tn:townn, d:int)):int =
                if t = tn then 0 else 1;


     {A function that compares a townn with a pair}
     {consisting of a townn and a direction (int from 0}
     {to 360).  Used as a metric in bestfit}

val bestfit (a, l, met) = fold (comp a met) l (hd l);

        {bestfit individually compares all the items in the}
        {list l to the item a and produces the element in l}
        {that is closest to a}

val comp = fun x.fun met.fun (a, b).
                        if met(x, a) < met(x, b) then a else b;

        {Two items a and b are compared to each other by}
        {evaluating each of them with the function met.}
        {met returns the 'distance' between two items.}

val tdirec (t1:townn, t2:townn):int =
                let val dc = dircstown (findtown t1)
                in
                  p22 (bestfit (t2, dc, sametown))
                end;

        {tdirec returns the direction in degrees from t1 to}
        {t2.  It does this by finding the first town on the}
        {map and then finds the direction to the second}
        {town in the first town's list of directions to}
        {other towns.}

val wdirec(way:stage):int = tdirec(t1way way, t2way way);

        {wdirec returns the direction of a given route stage.}

val devn (d:int, way:stage):int = langle(modulus (d - wdirec (way)));

        {d is a direction}

val mindevn (d:int, ways:stage list):stage = bestfit (d, ways, devn);

        {mindevn returns the stage in ways that whose}
        {direction is closest to the direction d.}

val bestway (t1:townn, t2:townn):stage =
                mindevn (tdirec (t1, t2), waysfrom t1);

        {t1 and t2 are not equal nor adjacent.  bestway}
        {returns the stage that begins at t1 and whose}
        {direction is closest to the overall direction from}
        {t1 to t2.}

val findroute(t1:townn, t2:townn):stage list = fr(t1, t2, []);

    {finds the list of stages that form the route from t1 to t2.}
```

```
val rec fr(t1:townn, t2:townn, route:stage list):stage list =
          if t1 = t2 then route
          else
          if adj(t1, t2) then fr(t2, t2, route @
                                          (waytowns(t1,t2)::nil))
          else fr (t3, t2, route @ (waytowns(t1,t3)::nil))
               where
                 val t3 = t2way(bestway(t1, t2))
               end;

        {there are three cases to consider}
        {a) If t1 = t2 then there is no need to find a}
        {route.}
        {b) If t1 and t2 are adjacent then the route}
        {consists of the stage from t1 to t2.}
        {c) Otherwise find the best single stage from the}
        {current town.  Add this stage to the route and}
        {make the town at the far end of this stage the new}
        {current town.}
```

APPENDIX G

MICRO-ORGANISMS-3

NAVIGATE-3 is implemented in the ML functional language.

```
module proptype
body
import listops;

          {An organism has various kind[s] of property.}
          {for example, identity iden, pathogeny path and so on.}

type kind = iden|path|stain|morph|growth|air;

          {Each kind of property has a name}

type name = a|b|c|d|x|y|o|u|
                                {names of iden}
            h | 1 |
                                {name[s} of pathh}
            pos|neg|
                                 {name[s] of stain}
            rod|coccus|
                                 {name[s] of morph}
            clumps|chains|
                                 {name[s] of growth}
            ae|anae;
                                {name[s] of air}

      {A property has a kind, name and certainty factor}

type prop = mprop of kind * name * int;

val propkind(mprop(a, b, c)) = a;
val propname(mprop(a, b, c)) = b;
val propint(mprop(a, b, c)) = c;

     {Retrieves the first property of kind k in the}
     {list of properties pl.}

val rec getpropkind(k:kind, pl:prop list):prop
  = if null pl then escape "getpropkind"
  else
    if propkind(hd pl) = k then hd pl
    else getpropkind(k, tl pl);

     {Retrieves the first property with name n in the}
     {list of properties pl.}

val rec getpropname(n:name, pl:prop list):prop
  = if null pl then escape "getpropname"
  else
    if propname(hd pl) = n then hd pl
    else getpropname(n, tl pl)
end;      {module proptype}
```

```
        {The properties of all the infectious organisms known}
        {to the system are kept in a taxonomy.  This taxonomy}
        {is searched, top-down, whenever the properties of an}
        {organism are required.  Any property can be used as}
        {the key for this search.}

module taxtype
body
import proptype;

        {A taxonomy of organisms consists either of a single}
        {organism or of a collection of taxonomies together with}
        {rules for identifying subtaxonomies and properties of}
        {that collection.}

type rec tax = mtaxleaf of (prop list)|
               mtaxnode of ((tax list)*
                            (prop list)*
                            (prop list * prop list)list
                           );

        {The following three functions extracts the various}
        {parts of a taxonomy}

val taxsoftax(mtaxnode(t, p, f)) = t|
    taxsoftax(mtaxleaf l) = escape "taxsoftax";

val propsoftax(mtaxnode(t, p, f)) = p|
    propsoftax(mtaxleaf l) = l;

val funoftax(mtaxnode(t, p, f)) = f|
    funoftax(mtaxleaf l) = escape "funoftax";

     {gettax retrieves the taxonomy in the list of}
     {taxonomies tl with the name n.}

val rec gettax(n:name, l:tax list):tax
  = if null l then escape "gettax"
  else
    let val p = propsoftax(hd l)

     {The name of a taxonomy is the value of the iden property}

    in if propname(getpropkind(iden, p)) = n
       then hd l
       else gettax(n, tl l)
    end;
```

```
     {`ftax' stands for "fuzzy" taxonomy i.e. a pair of taxonomy}
     {with certainty factor.}

type ftax = mftax of tax*int;

val taxofftax(mftax(a, b)) = a;

val intofftax(mftax(a, b)) = b;

val nosubtax(mftax(mtaxleaf _,_))=true|
    nosubtax(mftax(mtaxnode _,_))=false

end;     {module taxtype}

     {This module contains the functions that are}
     {used to identify an organism as a member of a}
     {subcategory of a given category.}

module identifysubtaxc
body
import taxtype;

     {the property p sat[isfies] the property q if it is}
     {of the same kind, has the same name and the certainty factor}
     {of p is at least as large as the certainty factor of q.}

val sat(p:prop, q:prop):bool
  = if (propkind p = propkind q) then
      (if (propname p = propname q) then (propint p >= propint q)
       else false)
  else false;

     {matchl returns true if p is sat[isfied] by a property}
     {in the list pl of properties}

val rec matchl(p:prop, pl:prop list):bool
  = if null pl then false
  else
    if sat(hd pl, p) then true
    else matchl(p, tl pl);

     {match returns true if all the properties in prem are}
     {sat[isfied] by a property in the list pl of properties}

val rec match(prem:prop list, pl:prop list):bool
  = if null prem then true
  else
    if matchl(hd prem, pl) then match(tl prem, pl)
    else false;

     {satisfy returns true if the properties x sat[isfy]}
     {the premise of the rule r.}

val satisfy(x:prop list) (r:prop list * prop list):bool

     {pl2 returns the premise of the rule r}

  = match(pl2 r, x);
```

```
      {Get the taxonomy in t that has the same name as p}
      {and let the certainty factor of the taxonomy be}
      {the certainty factor of p.}

val getsubtax(t:tax list) (p:prop):ftax
  = mftax(gettax(propname p, t), propint p);

      {fire the list of rules rl to produce the list of taxonomies tl}

val fire (rl:(prop list * prop list)list, tl:ftax):ftax list

      {p22 (hd r) returns the rhs of a rule and consists of the}
      {name of a subtaxonomy in tl}

  = (map (getsubtax (taxsoftax(taxofftax tl)))) (p22 (hd rl));

val processerror(r:(prop list * prop list)list)
  = escape "processerror";

      {infersubtax uses a list of properties pl to attempt}
      {to sat[isfy] the rules associated with the taxonomy t.}
      {Providing only one rule, satrule, is satisfied,}
      {this rule is fire[ed] to produce a list of taxonomies.}

val infersubtax(pl:prop list, t:ftax):ftax list

      {retrieve rules of taxonomy t}

  = let val rules = funoftax(taxofftax t)
in

     {find rules that satisfy the properties pl}

   let val satrule = filter(satisfy pl) rules
   in
     if null satrule then [t]
     else

    {if only one rule satisfy[ed] then fire that rule}

       (if length satrule = 1 then fire(satrule, t)
        else processerror satrule)
  end
end;

    {only identifysubtax of the organism described by the}
    {properties pl if the current taxonomy of the organism}
    {contains subtaxonomies.}

val identifysubtax(x:prop list, t:ftax):ftax list
  = if nosubtax t then [t]
  else infersubtax(x, t);

      {scale part of the certainty factor calculus
required by indentifysubtax.}
```

```
val scalel (x:int) (t:ftax):ftax
  = mftax(taxofftax t, mul(x, intofftax t));

val scale (x:int, t:ftax list):ftax list
  = map (scalel x) t;

     {the organism, described by the list of properties}
     {pl, is identified as a member of a subtaxonomy of t.}

val identifysubtaxc(pl:prop list)(t:ftax):ftax list
  = scale(intofftax t, identifysubtax(pl, t))

end     {module identifysubtaxc};

     {This module contains the functions that allows the program}
     {to manipulate uncertainties.  The implementation below is of}
     {a "dummy" calculus and is for illustration purposes only.}
     {However, this module contains the sort of functions necessary to}
     {implement a calculus such as that used in mycin.}

module fgetprop
body

import taxtype;

     {scombine accumulates the certainty factors of different}
     {contributions of evidence for a particular property.}
     {We use plus here but another function could be put in its place.}
     {For example, the "accumulation" of (a,_,3) and (a,_,4) is (a,_,7).}

val scombine(l:prop list):prop
  = if not(allsame((map propname) l)) then escape "scombine"

     {Only properties of the same kind are accumulated}

  else mprop(propkind(hd l),
             propname(hd l),
             fold plus ((map propint) l) 0
            );

     {nameis is true of n and p if n is the name of the property p}

val nameis (n:name) (p:prop):bool
  = propname p = n;

     {nsplit is used by namesplit which "sorts" properties into}
     {groups so that all the properties in a single group}
     {have the same name as required by scombine.}

val rec nsplit(l:prop list, a):(prop list)list
      = if null l then a else
    let val n = propname(hd l)
    in
      let val nl = filter(nameis n) l
      in nsplit((difference l nl), nl::a)
      end
    end;
```

```
val namesplit(l:prop list):(prop list)list
  = nsplit(l, []);

     {combine "accumulates" the certainty factors of properties}
     {but the properties need not all have the same name.}

val combine(l:prop list):prop list
  = if not(allsame(map propkind l)) then escape "combine"
    else (map scombine) (namesplit l);

     {retrieve the k property from the taxonomy t.}
     {In addition modify the certainty factor of k}
      {to take account of the certainty factor of t.}

val getpropc (k:kind) (t:ftax):prop
  = let val p = getpropkind(k, propsoftax(taxofftax t))

     {p is the value of the k property of the taxonomy t}

      and x = intofftax t

     {x is the certainty factor of the taxonomy}

in
    mprop(propkind p, propname p, mul((propint p), x))

     {mul is the certainty factor operator but some other}
     {function could be used instead.}

  end;

     {fgetprops retrieves the p properties of all the taxonomies}
     {in t and "accumulates" their certainty factors.}

val fgetprop(p:kind, t:ftax list):prop list
  = combine((map (getpropc p)) t)
end;      {module fgetprop}

     {main program}

import fgetprop identifysubtaxc;

     {eqftax tests if t and s are the same taxonomies.}

val eqftax(t:ftax, s:ftax):bool
  = let val p = propsoftax(taxofftax t)
     and q = propsoftax(taxofftax s)
in
  propname(getpropkind(iden, p))
  = propname(getpropkind(iden, q))      {name of p = name of q}
end;
```

```
     {eqftaxlist tests if t and s are the same list of taxonomies.}

val eqftaxlist(t:ftax list, s:ftax list):bool
  = if (length t = length s)
    then
       (fold preand (map eqftax (pairlists(t, s))) true)
    else false;

     {identify the immediate subcategory of t to which x belongs}

val identifysubtaxes(x:prop list, t:ftax list):ftax list
  = appendall (map (identifysubtaxc x) t);

     {identify the organism x as a member of taxonomy t}

val rec fidentify(x:prop list, t:ftax list):ftax list
  = let val subtaxes = identifysubtaxes(x, t)

     {subtaxes is the immediate subcategory of t to which x belongs}
  in
    if eqftaxlist(t, subtaxes)  then t

     {if at bottom of taxonomy then organism identified}

    else fidentify(x, subtaxes)

     {else identitfy x further in the taxonomy}

 end;

     {findprop finds the value of the p property of the organism.}
     {The unknown organism, a member of the taxonomy t,}
     {is described by its properties x}

val findprop(p:kind, x:prop list, t:ftax):prop list
    = fgetprop(p, fidentify(x, [t]));
```