# PORTAL
## An Expert System Shell

by

KENNETH IVAN GAMBLE
B.Sc., University of Victoria, 1983

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the Department
of
Computer Science

ACCEPTED
FACULTY OF GRADUATE STUDIES
[redacted]
DEAN
DATE 23 oct 1987

We accept this thesis as conforming
to the required standard

[redacted]
Dr. E. J. H Chang
[redacted]
Dr. G. Shoja
[redacted]
Dr. C. Morgan
[redacted]
Dr. D. Protti



University of Victoria

Supervisor: Professor Ernest J. H. Chang

## ABSTRACT

PORTAL is an expert system shell developed to provide an environment that facilitates the direct use of a computer by a domain expert programming an expert system. Particular reference is made to the expert system LIVER, an expert system for the interpretation of medical laboratory tests for liver dysfunction. We discuss the knowledge representation and inference mechanism developed for this system as well as the programming tools created to facilitate the expert's direct use of the system for entering, executing and testing the knowledge base as it is being developed.

Examiners:

[redacted]

Dr. E. J. H Chang

[redacted]

Dr. G. Shoja

[redacted]

Dr. C. Morgan

[redacted]

Dr. D. Protti

# Table Of Contents

## List of Tables



## List of Figures

# Acknowledgements

I would like to thank three people for their help and support in the development of this thesis. The first person is my wife, Donna Ellsay, without whose patience and support this work would not have been possible. Secondly I would like to thank Dr. Michael McNeely who provided the impetus and financial support for this work. Dr. McNeely's enthusiasm and hard work were a constant source of inspiration. Finally, I would like to thank Dr. Ernest Chang who has provided, through this project and others, opportunities for doing interesting work with interesting people in interesting environments.

I would also like to thank the Alberta Research Council for its support, through resources and time, that enabled the completion of this thesis.

*Machines that lack knowledge seem doomed to perform intellectually trivial tasks. Those that embody knowledge and apply it skillfully seem capable of equaling or surpassing the best performance of human experts. [Hayes-Roth,et al.1983]*

# Chapter 1

# Expert Systems

Expert systems are making a significant impact in computer applications [Hayes-Roth et al. 1983; Buchanan and Shortliffe 1985; Barr and Feigenbaum 1981]. Expert systems have become an increasingly popular programming tool for the collection and integration of human expertise into computer programs. Expert system technology has moved from basic research in universities to a mature commercial technology, as evidenced by the proliferation of commercial expert system tools.

There have been many promising results and extensive research and commercial development of these products over the last few years, including development of tools for the creation of expert system programs. These tools are sometimes referred to as expert system ***shells.*** This thesis is a discussion of some aspects of the development and implementation of such a shell, PORTAL [Gamble and Chang] 1984, and the implementation of a particular expert system application, LIVER [Chang, McNeely, and Gamble 1984a]. The main focus of the thesis will be on the direct use of the expert system shell by the human expert and the tools, techniques, and representations developed to facilitate this process.

## 1.0 Computerization of Human Problem-Solving

Expert system technology is the result of an effort to computerize human problem-solving methods. This effort started in the 1960's with attempts to create systems that used general problem-solving techniques, typified by the system GPS (**G**eneral **P**roblem **S**olver) [Newell and Simon 1972].

The aim of systems like GPS was to enumerate and implement general human problem solving methods. The term "general" implies methods that are independent of specific knowledge about particular problems. Two typical techniques developed through this work were ***means-end analysis***[1] and ***heuristic search.***

There are two problematic assumptions with the techniques proposed by such systems [Sell 1985]. The first is that we can actually know and understand most human problem-solving methods. The lack of solutions to two current open problems, the understanding of how the human mind works and the creation of an intelligent artifact [Sell 1985], supports belief in the difficulty of this task. While humans are able to solve problems, the exact mechanisms for many of the methods they use are still unclear.

The second assumption, the applicability of general problem solving techniques in all domains, creates a more serious problem. Attempts to increase the generality of problem solving techniques leads to a decrease in problem representation capabilities. In GPS for example each generalization

---

[1]Means-end analysis is a method by which the steps needed to solve a problem are generated. Given a ***state description*** and a ***goal*** a comparison is made to extract a ***difference*** between the state and the goal. This difference is calculated by a function designed specifically for the problem domain. An **operator** is selected from a difference table that will reduce this difference. If the operator cannot be directly applied to the initial state then subgoals are created and the process continues recursively. This results in a solution given by a sequence of operators representing the solution steps from the initial state to the final goal. [Barr & Feigenbaum 1981, Newell & Simon 1972].

caused a deterioration in the kinds of differences that could be detected [Barr and Feigenbaum 1981]. GPS was successful in solving a number of puzzles such as the Towers of Hanoi and the Missionary and Cannibals problem, as well as some more difficult problems like theorem proving in predicate calculus, but as more generality was added to the system the number of solvable problems decreased [Barr and Feigenbaum 1981].

The lack of success in implementing general problem-solving methods led to a "paradigm shift" [Goldstein and Papert 1977] in Artificial Intelligence (AI) research. Through observations of human experts solving problems in their area of expertise, called a ***domain***, it was recognized that their problem-solving skills arose not from applying general problem solving methods but from applying the particular knowledge they had of their domain [Feigenbaum 1977]. The direction of research shifted from generalized problem solving to domain specific knowledge-based programs, exemplified by expert systems.

The first knowledge-based program was DENDRAL [Feigenbaum et al. 1977; Mitchell 1978; Lindsey et al. 1980], a program that hypothesizes the acyclic molecular structure of a compound from spectroscopic analysis. Although DENDRAL's design did not explicitly contain an expert system, the actual implementation used expert rules to plan and evaluate the molecular structures generated by DENDRAL [Feigenbaum 1977]. DENDRAL was and continues to be a very successful program [Barr and Feigenbaum 1981]. It surpasses all humans at its task [Hayes-Roth et al. 1983], it is cited in numerous papers on molecular compounds, and it was the first to show the power of applying domain specific knowledge to problem solving [Feigenbaum 1977].

Two other seminal expert systems are MYCIN [Shortliffe 1976] and PROSPECTOR [Duda et al. 1979]. MYCIN introduced the concepts of ***rule-based*** knowledge representation with ***backward chaining*** and an explicit representation of ***uncertainty***. PROSPECTOR combined the use of rules with ***models*** of the domain. These concepts will be discussed in Chapter 2.

## 1.1 Characteristics of Human Expertise

What characteristics of human experts gives them greater facility in solving domain problems than non-experts? First, experts have more domain knowledge than non-experts [Feigenbaum 1977] from their greater experience and training. Secondly, there is a rich quality and integration to expert knowledge [Hayes-Roth et al. 1983]. Expert knowledge contains specialized methods and structures that are critically important in efficient utilization of their knowledge [Holland et al. 1986].

Thirdly, experts recognize significant patterns in domain problems. They have the ability to make inferential leaps [ Hayes-Roth et al. 1983], allowing them to quickly generate solutions to problems, especially typical domain problems. Given data for a particular problem experts can quickly recall solution schema that represent typical responses to similar data patterns. With this schema of the problem in mind they then fill in the details of the solution with case specific data [Chi et al. 1981]. Experts use domain-specific patterns to solve most problems and fall back on general domain methods and first principles when they lack direct experiential knowledge [Holland et al. 1986].

Finally, experts are able to explain their reasoning. Some researchers believe that this is a fundamental characteristic of human expertise and use it

as a measure against which expert systems are compared [Feigenbaum 1977; Hayes-Roth et al. 1983].

## 1.2 Problems Requiring Expert System Solution

Not all domains are appropriate for implementation using expert system techniques. Only those characterized by certain types of human expertise are good candidates.

One fundamental characteristic of a good candidate domain is that application of expertise is perceived as requiring skill [Feigenbaum 1977], particularly mental or cognitive skill, and that the skill manifests itself in quick and high quality solutions [Hayes-Roth et al. 1983]. A caveat is that the skill must not be easily acquired. Most good domains represent highly specialized fields of knowledge. Typically, it takes a number of years of study, practise and experience for a person to become recognized as an expert in a domain [Feigenbaum 1977; Waterman 1985]. Scene recognition or speech understanding are not good expert system domains because, although both are specialized and difficult to implement in a computer, they are general human skills easily performed by most children.

Another characteristic of good domains is robustness [Waterman 1985]. Robust expertise provides solutions that are generally valid and consistent across the whole domain.

Domains that are good for expert system development are usually also characterized by the absence of a complete model that explains all aspects of the domain, or any well-defined methods for solving all domain problems. Without a complete model it is experience in the utilization of domain knowledge that facilitates expert behaviour. Solutions to problems in such

domains utilize the particular knowledge that experts acquire through experience, the rules of thumb and significant patterns called ***heuristics*** [Feigenbaum 1977]. Conversely, domains which contain algorithmic methods that always guarantee correct solutions are not good expert system domains [Waterman 1985]. For example, sorting lists would not be a good expert system domain as there are well defined algorithms to implement this problem. Expert system technology should be applied if there is no other good computational technique; it should be the "method of last resort" [Waterman 1985].

The application that motivated the creation of PORTAL is an example of a good expert system domain. This initial application was the interpretation of medical laboratory tests for liver dysfunction. This domain has no algorithmic method for deriving a particular liver dysfunction from test results. Rather, interpretive skill is acquired through experience in the recognition of salient patterns of abnormal and normal test results. These patterns or heuristics are well suited to expert system technology.

## 1.3 Knowledge Engineering

Once a domain is selected as appropriate for implementation as an expert system the expertise must be transferred from the human expert to a computer program. This process of encoding human expertise in an expert system is called ***knowledge engineering*** [Feigenbaum 1977].

Knowledge engineering is the transformation of the expert's cognitive model of problem solving in the domain into the knowledge models available in expert system technology. Hayes-Roth [Hayes-Roth 1984] has defined four major task in this process.

| Knowledge Processing Tasks | Engineering Activities | Engineering Products |
|---|---|---|
| Mining | Knowledge Acquisition | Concepts and Rules |
| Molding | Knowledge System Design | Framework and Knowledge Representation |
| Assembling | Knowledge Programming | Knowledge Base<br>Inference Engine |
| Refining | Knowledge Refinement | Revised Concepts and Rules |

Table 1.1
Knowledge Engineering Tasks [Hayes-Roth 1984]

Traditional knowledge engineering requires a ***knowledge engineer*** [Feigenbaum 1977], a specialist in programming expert systems. The knowledge engineer provides the mapping of domain concepts into expert system program structures. He knows expert system syntax and guides the construction of the domain representation. The aim of a knowledge engineer is to program the expert's domain knowledge into a computer program so that it will exhibit domain expertise at a level equal to or better than the human.

Knowledge acquisition is one of the most difficult aspects of expert system development. Feigenbaum has called it the "bottleneck" in expert system development [Feigenbaum et al. 1983]. Knowledge acquisition requires that the knowledge engineer learn as much as possible about the expert's domain. He must become well versed in domain terminology and methodology. He must then match this knowledge to an appropriate expert system knowledge representation, develop the model of the domain within the representation and then program it into a computer. Finally he must direct

the testing and refinement of the knowledge base to ensure that it accurately reflects the expert's heuristics.

An alternative approach, which motivates this thesis, is to allow the domain expert to directly interact with a computer to create an expert system. A problem, however, is that the activity performed by a knowledge engineer is an important part of the expert system construction process. To remove the human knowledge engineer from this process will require strong supporting techniques.

A number of strategies can be employed to assist the domain expert. One is to provide a knowledge elicitation system. This strategy requires tools for interviewing the expert, extracting domain knowledge and checking for inconsistencies. The acquired knowledge is then compiled into an appropriate expert system representation, usually in the form of rules. Some systems that use this strategy are ETS [Boose 1984], TEIRESIAS [Davis and Lenat 1982] and MORE [Kahn et al. 1985].

A second strategy is to provide an expert system representation and inference technique, simple enough to be easily mastered by the domain expert, yet sufficiently powerful to accurately represent the domain heuristics. This strategy requires tools to assist the expert in correctly entering and testing his knowledge. This is the model adopted in PORTAL, the work described in this thesis.

To be both powerful and simple a representation scheme must be robust, able to handle the complexities of the expert's domain, yet not so complex as to be beyond the scope of the expert to learn in a reasonable amount of time. Rule-based representation provides such a scheme. It is both simple and flexible, and is usable in many expert domains, especially those of classification

and diagnosis [Buchanan et al. 1984]. Rule-based systems provide a representation that is easily understood by domain experts. It also provides a model similar to actual methods used by experts, such as the methods represented in statements about what do to in specific situations and statements identifying significant patterns of data [Barr and Feigenbaum 1981; Holland et al. 1986].

As with most simplifications there is a tradeoff, in this case with representational power. A well-trained knowledge engineer has at his disposal many knowledge representations and techniques. It can take a number of years of experience to become fluent in all the nuances of expert system technology. Domain experts usually do not have the inclination or the time to completely learn this new technology. However, if some restrictions are placed on the types of domain that may be implemented, many domain experts can quickly learn to use rule-based systems to represent their domain heuristics. PORTAL uses such a rule-based system to allow an expert to simply express his heuristics, particularly in classification and diagnostic domains.

A critical component of an expert directed system is a set of tools for specifying the knowledge and testing its validity. These tools should provide an environment that allows the expert to create and edit the knowledge representation as well as execute and analyse the developing knowledge base.

## 1.4 PORTAL

PORTAL [Gamble and Chang 85] is an expert system shell developed in conjunction with Drs. Chang and McNeely. It contains a suite of tools for creating, developing, executing and testing a rule-based knowledge representa-

tion and has been used by domain experts to directly create expert systems. PORTAL was developed to allow a domain expert to directly encode the heuristics for interpreting medical laboratory tests for liver dysfunction. This area of expertise provides a good example of the type of domain well suited to direct encoding by a domain expert.

The following chapters will examine PORTAL in more detail. Chapter 2 will give an overview of the concepts and terms in expert systems and knowledge acquisition that apply to the development of PORTAL. Chapter 3 will present a brief description of similar systems and a discussion of PORTAL's history. Chapters 4 and 5 will give a detailed description of PORTAL's knowledge representation and development tools. Chapter 6 will look at experiences with using PORTAL and will discuss some results and future work.

# Chapter 2
# Expert System Architecture

PORTAL is a shell for building forward chaining rule-based expert systems that use uncertain data and qualified knowledge. This chapter will review the basic components of rule-based systems. It will also examine some mechanisms that support reasoning with uncertainty, with particular reference to certainty factors as used in MYCIN [Shortliffe and Buchanan 1975].

The main question in developing an expert system is not what knowledge is to be encoded but how the knowledge is to be encoded [Barr and Feigenbaum 1982]. It is assumed a priori that a body of knowledge for the domain exists. In this chapter we will look at the problem of how to identify and subsequently represent, retrieve and use this knowledge in a computer program.

Before discussing the components of rule-based expert systems two basic concepts need to be addressed: the use of symbols and the predicate calculus. These concepts provide some of the theoretical foundation on which expert system technology is based.

## 2.0 Symbol Processing

The use of ***symbols*** to represent domain knowledge is fundamental to expert systems. For a computer to solve problems in the real world there must be a mechanism for representing the world in a program. Symbols provide such a mechanism. Symbols are physical patterns, usually represented as a string of characters, **A, My-car, Table, 27.367**. Symbols can be grouped together to form ***symbol structures***, which are composed of physically related symbols [Newell and Simon 1976]. Symbol structures are usually represented in

artificial intelligence research as a list of symbols or other symbol structures contained within parentheses: **(Colour My-car Blue)**, **(Times 4.6 5.3)**, **(Daughters-of Ken Jennifer Katie)**, **(Bird (Name Tweety) (Colour Yellow) (Nemesis Sylvester))**.

The use of lists to represent symbol structures led to the creation of list processing languages such as LISP (**LISt Processing**). These languages are one of the significant contributions of Artificial Intelligence (**AI**) to computer science. They provide new ways of viewing and using computers.

> *LISP has jokingly been called "the most intelligent way to misuse a computer." I think that description is a great compliment because it transmits the full flavour of liberation: it has assisted a number of our most gifted fellow humans in thinking previously impossible thoughts [Dijkstra, 1972]*

The LISP language gives programmers the ability to dynamically create and manipulate symbol structures as well as providing general programming capabilities.

## 2.1 The Predicate Calculus

The ability to create symbols and symbol structures in a computer is accompanied by the need for methods of utilizing them. One such method is the use of reasoning. Simon [1983] calls the reasoning metaphor "the idea that problem solving consists of applying reason to problem situations" [*ibid.* p. 9]. Reasoning is implemented in logic languages such as the predicate calculus, portions of which form part of the theoretical base of rule-based systems.

Statements about the world can be composed using predicate calculus symbol structures. The basic predicate calculus structure is the ***proposition*** or ***atomic formula.***, which consists of a ***predicate symbol*** and some ***terms***. The

predicate symbol is the first element in a symbol structure list and the remaining elements are the terms ie. **(Predicate-Symbol Term1 Term2 ... TermN)**. Terms provide names for domain objects. Predicate symbols provide ways of expressing relationships between objects, or qualities or attributes of objects, e.g. **(Sex Male)**, **(Age 33)**, **(Sister-of Ken Yvonne)**.

In predicate calculus, propositions can be connected together using ***connectives*** to create a ***formula.*** Three connectives are "∧" (and), "∨" (or), and "⊃" (implication). Another symbol, "~" (not), is sometimes called a connective. However, it does not "connect" two formulae, but rather negates a single formula. In list notation these connectives become **And, Or, If,** and **Not.** Connectives allow the expression of relationships among propositions. For example the implication formula:

```
(If (Or (Sex Male)
        (Greater-Than Age 50)
        (Less-Than Age 10))
    (Pregnancy Unlikely))
```

can be interpreted as: if sex is male or age is greater than 50 or less than 10 then it is unlikely that the person is pregnant.

Such implication formulae are important constructs in expert system technology, as they are the underlying structure for rules in rule-based systems. The two terms in an implication have been given special names, the first term is called the ***antecedent,*** the second term the ***consequent.***

The predicate calculus permits the use of variables in place of symbols in formulae. Variables allow the expression of generalized statements and can be used in the place of terms or predicates in formulae. The use of variables in the place of terms is called ***first-order*** predicate calculus, in the place of predi-

cates is called ***second-order*** predicate calculus. Most expert systems are based on first-order predicate calculus.

Variables support the use of ***quantifiers***. Two of these are ***universal*** and ***existential*** quantification. The symbol "∀" (**All**) is used to denote universal quantification and "∃" (**Exists**) existential quantification. The universal quantifier states that every assignment of a symbol or symbol structure to the variable named by the quantifier satisfies the formula. It allows expression of universal statements, e.g. (**All** (x) (**And** (**Man** x) (**Mortal** x))). Existential quantifiers are similar to universal quantifiers, except that the interpretation of the formula is that there exists some assignment to the variable that will make the statement true, but not necessarily for all assignments.

All formulae built out of terms and atomic formula using connectives and quantifiers are called ***well-formed formulas*** (wff). Wffs can be combined to create more complex wffs by using connectives and quantifiers, along with specific axioms that are a part of predicate calculus.

To apply predicate calculus to a particular domain, an interpretation is given for each wff by assigning a correspondence between each term in the language and an object in the domain, and between each predicate in the language and a relationship in the domain. One strength of predicate calculus comes from the well-understood interpretations that allow creation of complex statements. Another strength lies in the set of standard manipulations that preserve these interpretations, such as DeMorgan's law and the use of parentheses to indicate variable scoping [Stefik et al. 1983].

In expert systems the collection of formulae about a particular domain is called a ***knowledge base***. Given a knowledge base, predicate calculus provides methods for reasoning with the formulae, called ***proofs***, which use

mechanisms for manipulating wffs called ***rules of inference***. The rule of inference used as the basis for rule-based systems is ***modus ponens***,

$$\mathbf{A}, \mathbf{A} \supset \mathbf{B} \Rightarrow \mathbf{B}$$

where $\Rightarrow$ means ***yields***. Modus ponens states that if **A** is known and it is known that **A** implies **B** then **B** can be added to the set of formulae. Modus ponens permits the expression of heuristic knowledge. The implication rule, $\mathbf{A} \supset \mathbf{B}$, can be viewed as the statement: if pattern A exists in the domain then conclude B. Thus, given facts for a particular problem expressed as **A**, the facts and implications can be combined to generate new facts: in this case, **B**.

There are other rules of inference such as ***reductio ad absurbum*** and ***universal specialization***. Reductio ad absurbum is used as the basis for the resolution mechanism [Robinson 1965] in Prolog [Clocksin and Mellish 1984], but discussion of these rules is beyond the scope of this thesis.

## 2. 2 Expert System Components

Expert systems contain two types of components, those for representing domain knowledge and those for utilizing it. Knowledge representation components include techniques for representing the heuristics used to reason in a domain, and techniques for representing the data for particular domain problems. Utilization components provide techniques for applying the domain heuristics to data for a particular problem as well as interface facilities.

While every expert system contains at least one representation technique and method for applying the encoded knowledge to domain problems, there are almost as many variations in the capabilities and style of interface facilities as there are expert systems. In this chapter we will be discussing rule-based representation and application techniques only.

Discussion of interface facilities will be limited to the particular techniques used in PORTAL and will be covered in the chapters on the PORTAL implementation.

### 2.2.1 Rule-Based Systems

Rule-based expert systems provide a specific mechanism, the rule, for representing the heuristics used by experts. Current rule-based systems are a generalization of a formal reasoning method called ***production systems*** [Post 1943]. Rule-based systems are composed of three basic components: a set of rules; a database of facts, sometimes called ***working memory,;*** and an interpreter for the rules, called the ***inference engine*** [Davis and King 1984].

Rules are composed of two parts: a ***left hand side*** (LHS) and a ***right hand side*** (RHS). A rule can be viewed as a predicate calculus implication where the LHS is the antecedent and the RHS the consequent. The LHS and RHS of rules are also called the IF and THEN parts of a rule. The general IF/THEN form of a rule is:

```
IF
   <antecedents>
THEN
   <consequents>
```

The LHS and RHS of a rule are composed of formulae. In general the LHS formulae represent significant patterns of data in the domain and RHS formulae represent the conclusions that can be drawn or the meaning of the data. This example shows a rule from the LIVER system developed using PORTAL:

IF
    Sex is male or
    Patient Age is less than 10 years or
    Patient Age is greater than 50 years
THEN
    Pregnancy is unlikely

The aim in developing a rule-based expert system is to capture all such heuristic rules used by the domain expert.

In a classical rule-based system the data for a given problem are represented as propositions in working memory. These propositions represent statements such as: **Sex is female** and **Age is 32**.

Classical rule-based systems do not provide a clear separation of the objects in the domain from the heuristics used to reason about the objects. There is not necessarily any definition of domain objects outside of their implicit definition in a rule or working memory element. For example, the sex and age statements given above are only implicitly connected to a particular person. This approach is based on the assumption that the system is dealing with one patient at a time. The interrelationship of objects and their attributes comes as a side effect of the statements in the rules and working memory.

This lack of a separation of objects and heuristics does not cause a problem in domains where there are simple objects or where there is a single global object. The analysis of medical laboratory tests is such a domain and was the initial expert system built in PORTAL. There is a single implicit domain object, the patient, and all facts in working memory and heuristics in the rules are assumed to be statements about a single patient.

In domains that require more complex objects or deal with more than one object of the same type a more explicit representation of objects should be used. Frame-based systems [Minsky 1975] provide a method for explicitly de-

scribing domain objects and the relationships among the attributes of an object by using a frame to represent the object and assigning slots to each significant attribute of the object. Frame-based systems also provide techniques for stating default values for attributes and for creating representations of hierarchical relationships.

PORTAL provides a mechanism for using symbols to represent domain objects. These symbols are called ***entities***. PORTAL allows the expert to specify what type of data the entity represents, for example number or a set of values. It also has a set of built-in attributes for each data type, such as the units attribute of a numeric entity.

### 2.2.2 Inference Engine

Once the domain knowledge has been represented in rules it must be applied to domain problems. This is done by the inference engine. In rule-based systems there are two types of inference engines: ***forward-chaining*** or ***data-driven*** and ***backward-chaining*** or ***goal-directed***.

#### 2.2.2.1 Forward-Chaining

Forward-chaining inference requires all of the data for a problem to be present in working memory. The basic inference step is to find all the rules whose LHS elements match elements in working memory and then to select and execute one of them. Matching a rule involves comparing symbols, called ***literals***, in LHS formulae with elements in working memory. This comparison may involve predicate relationships between the LHS and working memory elements that must be true for the elements to match. For rules that contain variables, matching can also involve instantiation of the rules by binding the variables with values from working memory elements.

A rule is selected for execution from the set of rules whose complete LHS matches elements in working memory. Execution of a rule, called *firing* a rule, involves invoking actions represented in the RHS of the selected rules. The usual actions modify working memory by adding or deleting working memory elements. Some systems, such as OPS5 [Forgy 1981], allow execution of arbitrary procedures as RHS actions.

Firing a rule usually causes some modification in working memory, enabling new rules to become fireable on subsequent cycles. This process of rule selection and execution is called the ***recognize-act*** cycle [McDermott and Forgy 1978]. The cycle ends when some termination criteria are met. Some of these criteria will be discussed later in this chapter.

The recognition portion of the recognize-act cycle can be subdivided into the ***match*** and ***conflict resolution*** activities [McDermott and Forgy 1978]. In general, for a given rule-base there is more than one rule that matches elements in working memory. This set of fireable rules is called the ***conflict set***. Selection of the best rule to fire from the conflict set is called conflict resolution.

Forgy and McDermott [1978] discuss some techniques for selecting rules from a conflict set when using forward-chaining. The techniques divide into four classes of rule selection based on: properties of the rules, properties of the data, arbitrary techniques and hybrid techniques. In general most systems use hybrid techniques but it is elucidating to look at all four classes.

The first rule property based rule selection technique is ***rule ordering***. A trivial example of rule ordering is to select from the conflict set the first rule entered into the system. Other techniques for rule ordering include: ordering rules within groups of rules with similar formula, and explicit creation of a

rule precedence network (DENDRAL). PORTAL uses a form of rule ordering based on the certainty factor of the RHS elements.

The second technique for ordering based on properties of the rules is called ***special case ordering*** [McDermott and Forgy 1978] or ***generality ordering*** [Davis and King 1984]. With this technique the most ***specific*** rule is selected to be fired. The more specific rule of two rules is the one which has, as a subset, all the LHS formula of the more general rule, and which may have constants where the more general rule has variables.

Techniques that select rules from the conflict set based on properties of working memory elements usually involve selection based on ***recency*** [McDermott and Forgy 1978; Davis and King 1984] Recency ordering requires that each element in working memory has a ***time stamp*** indicating the order of insertion of the elements. Given two rules that are fireable the one whose matching working memory elements are the most recent is the one fired.

Another data based rule selection technique uses ***distinction*** as a criteria [McDermott and Forgy 1978]. To utilize distinction criteria the inference mechanism must keep track of the facts that were used to select previously fired rules. One example using distinction criteria is: if a fireable rule has no facts in common with the last rule fired then select that rule to be fired. Another distinction technique is: if the rule matches some facts that have never been used with the rule then prefer that rule.

The third category of rule selection techniques is called ***arbitrary*** techniques and can contain any technique desired by the system designer. One example of an arbitrary technique is to select a fireable rule at random.

Few systems use only one of these techniques; generally they use some combination of the above. Such combinations are called ***hybrid*** techniques.

Quite often a single technique does not select a unique rule to fire,. However, an ordered combination of techniques, including the arbitrary technique of selecting a rule at random, can produce a unique rule for any set of data.

The following is an example of forward chaining using "recency" and "distinction" as the conflict resolution technique. Rules with the most recent memory element are fired first. A rule will only be fired once for any set of working memory elements matching the LHS elements. The example rules are:

1. If A B then C
2. If B D then C
3. If D E then C
4. If C E then G

Where A, B, C, D, E, and G are possible working memory elements. The RHS actions in this example system are to add the RHS element to working memory if it is not currently there or to update the time stamp of the working memory element matching the RHS. We will represent working memory as a list of element/number pairs where the number represents the recency of the element, with a larger number indicating a more recent element. The sample data for this example is:

((B 3) (D 2) (E 1))

The termination criteria for this system will be that the conflict set is empty after the match portion of the recognize/act cycle. Table 2.1 gives an example of forward-chaining with this data.

In the match part of cycle 1 rules 2 and 3 are placed in the conflict set. Rule 2 is selected to be fired by the conflict resolution technique as it contains the most recent element, B. The RHS of rule 2 is fired to update working memory with C. C is given a time stamp of 4. On the next cycle rules 3 and 4

are selected as fireable. Rule 4 is selected to be fired as it contains the most recent working memory element, C. The execution of rule 4 causes G to be placed in working memory. In cycle 3 rule 3 is fired which updates the time stamp of working memory element G. In cycle 4 there are no further rules to be fired so the conflict set is empty and the inference terminates. From the initial data the conclusions G and C have been generated using forward-chaining.

| Cycle | Conflict Set Matches | Rule fired | Working memory |
|---|---|---|---|
| 0 | - | - | ((B 3) (D 2) (E 1)) |
| 1 | 2 (B 3) (D 2)<br>3 (D 2) (E 1) | 2 | ((C 4) (B 3) (D 2) (E 1)) |
| 2 | 3 (D 2) (E 1)<br>4 (C 4) (E 1) | 4 | ((G 5) (C 4) (B 3) (D 2) (E 1)) |
| 3 | 3 (D 2) (E 1) | 3 | ((G 6) (C 4) (B 3) (D 2) (E 1)) |
| 4 | - | - | ((G 6) (C 4) (B 3) (D 2) (E 1)) |

Table 2.1
Forward Chaining Execution Trace

Forward chaining provides a technique for extracting conclusions from a body of data by letting the data drive the selection of the rules. It is a good technique for domains that have no single type of conclusion and a small well defined set of data.

#### 2.2.2.2 Backward-Chaining

Backward-chaining does not require all the data for a problem to be initially present in working memory. The basic mechanism in backward-

chaining is to gather the data to prove a goal. There are three ways to prove a goal. The first is that the goal is already known as an element in working memory and is proven by being already known. The second is to find a rule that concludes the goal and try and prove each of the LHS elements of the rule. The third is to query the user for the existence of the goal. The best way to understand backward-chaining is through an example. We will use the same rules as in the forward-chaining example:

1. If A B then C
2. If D E then C
3. If B D then C
4. If C E then G

The rule numbering has been slightly changed to highlight an aspect of backward chaining . The known data for the problem will be the same as the forward-chaining example: B, D and E. The goal to be proven is element G.

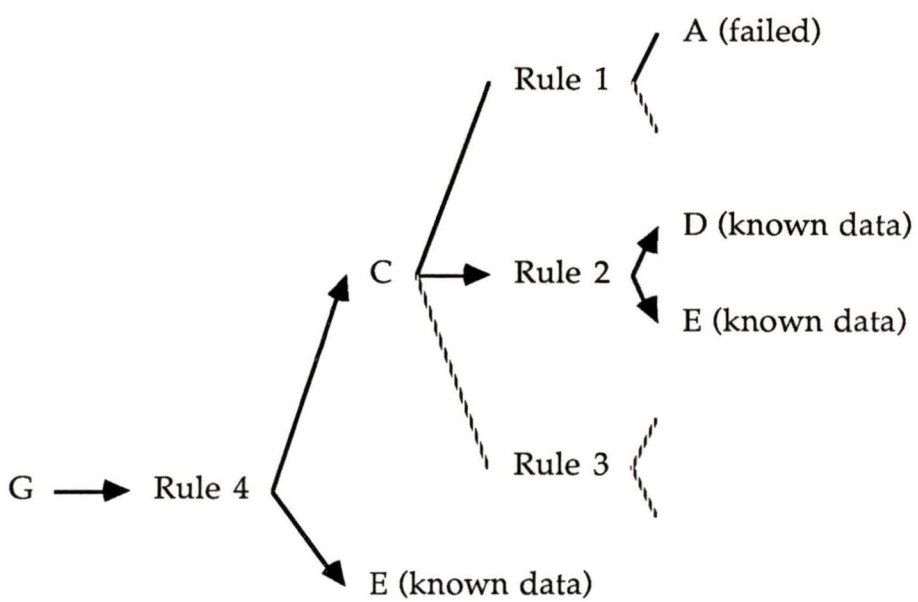


Figure 2.1
Backward Chaining Example

The following is a description of the backward-chaining tree given in Figure 2.1. The user selects goal G to be proven. G is neither in working memory nor known by the user so a rule must be used to prove it. Rule 4 is the only rule leading to G so it is selected. The LHS elements of rule 4 must now be proven. The first element is C and is chosen as a goal to be proven. C is not a known piece of data but there are three rules that lead to conclusion C. In this situation, some of the rule selection methods of forward-chaining conflict set resolution could be used to determine the particular rule to be used to prove C. Alternately, if there were existing elements in working memory then rules with partially matched LHS elements could be chosen first. In our case we will simply choose the first rule entered in the data base: rule 1. To prove C using rule 1 the LHS elements must be proven. Element A is chosen as the first goal to be proven. A is not in working memory or known to the user, and as there is no rule containing it on the RHS, it cannot be proven. Thus rule 1 cannot be used to prove C. Rule 2 is then chosen to prove C. To prove this rule B and E are required. Both are found to be true by querying the user and are placed in working memory. Rule 2 can now be fired and C placed in working memory. Now rule 4 is known to be true, the second element E was added to working memory as data for firing rule 2. Rule 4 is now fired and G is added to working memory. Thus G is proven and the backward-chaining ends.

#### 2.2.2.3 Uses of Forward and Backward Chaining

The type of domain and the style of problem solving both influence the choice of forward or backward chaining as the inference mechanism. The

main decision is whether or not the problem can be expressed as the solution to a goal. In MYCIN the main goal is to prove an infection, allowing efficient use of backward-chaining. In other domains such as the interpretation of laboratory tests there may be many possible conclusions, and thus a forward chaining method is more appropriate. Another conclusion is the quantity of data for typical domain problems. If the data set is well defined and small then forward chaining can be used. If the data set is potentially large and diffuse then a goal directed search would be appropriate. The interpretation of laboratory tests is a good example of a good forward chaining domain. The data set is well defined and quite often available in computer readable form.

### 2.2.3 Reasoning with Uncertainty

Two assumptions in the above discussion on inference mechanisms are: that all the data is of the same quality and that the significance of each heuristic expressed in a rule is the same. However, the real world contains varying degrees of data and heuristic quality. Techniques have been developed to qualify knowledge base statements. These techniques are able to make explicit statements about the degree of uncertainty in the conclusions derived by the system.

There are a number of mathematically based uncertainty models, which use a single-valued metric for the degree of belief of each statement in the system. This section discusses three techniques: the Bayes posterior probability method, possibility theory, and certainty factors as used in MYCIN. PORTAL uses the certainty factors but all three techniques will be discussed since aspects of the other two (probability theory and possibility theory) are used to develop PORTAL's certainty factor mechanism.

In systems that use Bayesian posterior probability, rules are defined not explicitly but implicitly. These implicit rules consist all the possible combinations of evidence that might lead to a given conclusion. Most combinations will not be likely to occur and so their probability will be zero. For a given set of data, the best conclusion is the one with the highest probability Several problems are associated with the use of the Bayesian method.

One of these is that conclusions must be independent: the probabilities of all possible conclusions for a given set of data must sum to 1.0. Empirical studies of case examples can provide some insight into these values but there is usually insufficient data to provide all of them [Adams 1984]. The alternative is to ask experts to provide educated guesses, but this has been found to be a difficult problem for experts to enunciate [Weiss and Kulikowski 1984].

Another fundamental problem with the Bayesian method is the need for prior probabilities. Even if the likelihood of a given conclusion for a given set of data can be derived the Bayes method requires the probability of the conclusion without any data. This value is not always available especially in domains that are large, or where conclusions are usually given from sample cases which are not independent of the evidence.

Finally, unless the domain is small and in a sense uninteresting as an expert system domain, the assumption of evidential independence is most likely inaccurate. This requires the use of joint probabilities for dependant evidence. This, in most domains, is a potentially large amount of data and whether or not it is obtainable at all is an open question.

Another numeric uncertainty technique is fuzzy set theory [Zadeh 1965] or possibility theory [Zadeh 1978]. The basic assumption in possibility theory is

that a predicate is a statement about a symbol's membership in the set represented by the predicate. Zadeh suggests that there are functions which return values in the range [0,1] that can express the degree to which an element is in a given set or the ***vagueness*** [Zadeh 1978] of the relationship between the set and its elements. A standard example is the set of positive integers S with elements s, and the fuzzy subset F, of small positive integers. The membership function $\mu F(s)$, the degree to which s is in the set F, could be expressed the following way: $\mu F(1) = 1$, $\mu F(2) = 1$, $\mu F(3) = 0.95$, ... , $\mu F(20) = .015$. In general any function on elements in a set that returns a value in the range [0,1] would do. Now, given a variable X that can take on values in S, a possibility distribution is created for "X is in F" such that the possibility of X=s is equal to $\mu F(s)$.

Possibility theory provides ways of combining expressions involving conjunctions, disjunctions and negations of such formula. Thus given that Poss{X=s} is the possibility that X is equal to s the following relationships exist:

Poss{X=s **or** X=p} = max[Poss{X=s}, Poss{X=p}]
Poss{X=s **and** X=p} = min[Poss{X=s}, Poss{X=p}]
Poss{X≠s} = 1 - Poss{X=s}

The power of possibility theory lies in its ability to express uncertainty due to vagueness. However, it still has the same problem of data derivation that plagues the probabilistic methods. It would appear as difficult to objectively generate a membership function as it would be to create prior probabilities [Buchanan and Duda 1982].

Certainty factors were an attempt to overcome some of the problems in Bayesian systems. The underlying theory for Certainty Factors (CFs), however, has been shown to be equivalent in part to the Bayesian model with the assumption of independence. A full discussion of this equivalence can be

found in Adams [Adams 1985]. The seminal article on CFs is Shortliffe and Buchanan's article [Shortliffe and Buchanan 1975].

An interpretation of the CF model using possibility theory has been developed by Aikens, and Buchanan and Duda[Aikens 1980; Buchanan and Duda 1982]. This allows a logical combination of evidence using CFs and is the model used in PORTAL.

In using CFs the expert tries to express his belief in how much a piece of evidence or a pattern of evidence e, tends to add or subtract weight from the probability of a conclusion C. The degree to which the evidence supports C is called the ***measure of belief*** MB[C,e], while the degree to which it does not support C is called the ***measure of disbelief*** MD[C,e]. Both these values are in the range [0,1]. For any piece of evidence there are values for these measures of belief.

When viewed from a probablistic point of view the values can be seen as the amount by which P(C | e) moves away from the prior probability of C. As the measure of disbelief decreases the probability moves towards 1, and conversely as the measure of belief decreases the probability in ¬C increases. The measures can be combined to indicate the total movement of probability weight. This value is called the Certainty Factor (CF) and is written:

$$CF[C,e] = MB[C,e] - MD[C,e]$$

The CF values range from -1 to 1. Thus the CF of an assertion A written CF(A) is the degree to which the probability of A must be adjusted from its prior probability. The CF can be seen as the shift of the probability weight for a conclusion given the evidence and knowledge in working memory and the rules.

A CF of -1 means that the evidence does not support an assertion, that the assertion is false and its probability is 0. A CF of 1 means that the evidence completely supports the assertion, that the assertion is true and that its probability is 1. A CF of 0 means that there is no evidence or that there is contradictory evidence for a assertion, that the assertion is unknown or that there is no change in its probability.

Every statement in the knowledge base would have an associated CF. Facts given to working memory by the user are given a positive CF, usually 1. When a rule is entered into the system the expert provides a CF that reflects his belief in how much the certainty of the conclusion in the RHS of the rule should be adjusted given the existence of the evidence in the LHS of the rule.

Let CA be the current CF for assertion A, i.e., the certainty of A prior to evaluating a rule. Let CEx be the certainty given by the expert for the rule, $B \supset A$, ie., the expert believes that the probability of A should be adjusted proportionally to CEx if the evidence B is in working memory. Assume that the evidence B is in working memory. Let CR be the certainty returned by the rule when is has been matched with data B. Then CR is

$$CR = CF(B) \cdot CEx$$

That is the certainty of the heuristic represented by the rule $B \supset A$ is proportional to the evidence supporting the conclusion. The accumulated certainty of A after evaluating the rule, called $CF_{accum}$, is a function of the previous certainty of A, CA and the certainty returned from evaluating the rule, CR, given by the formula:

$$CF_{accum}(CA, CR) = \begin{cases} CA + (1 - CA) \cdot CR \\ \quad \text{if } CA \text{ and } CR >= 0 \\ \dfrac{(CA + CR)}{(1 - \min(|CR|, |CA|))} \\ \quad \text{if } CA \cdot CR < 0 \\ -CF_{accum}(-CA, -CR) \\ \quad \text{if } CA \text{ and } CR < 0 \end{cases}$$

Possibility theory suggests a mechanism for combining certainty factors within rules. As mentioned above the certainty returned by a rule or implication is:

$CEx \cdot CF(LHS)$

where CEx is the value given by the expert and CF(LHS) is a function of the following definitions.

For propositions in the LHS the CF is a function of the veracity of the predicate and terms. If the predicate is true in relation to the terms then the CF returned is the minimum CF of the terms. If the predicate is false then the CF returned is zero minus the minimum of the terms. For connectives **and**, **or** and **not** the values returned are:

**and**$(CF_1 \ldots CF_i) = \min(CF_1 \ldots CF_i)$
**or**$(CF_1 \ldots CF_i) = \max(CF_1 \ldots CF_i)$
**not**$(CF) = -CF$

where $CF_i$ is the CF of the ith term.

As with Bayesian probabilities and possibility theory, objective derivation of values used as certainty factors, especially for rule values, is open to question. However, the empirical success of systems using CFs in mimicking expert behaviour suggests that uncertainty management using CFs still has some utility in spite of theoretical objections [Adams 1985].

There are other uncertainty management mechanisms that use non-numeric models of uncertainty. Rule selection criteria based on data recency can be viewed as a form of uncertainty management in that newer data is seen to have more certainty or applicability than older data. Thus rules that utilize new data are fired first as they are more applicable.

Certain symbolic uncertainty methods use formal models, two of which are Doyle's truth maintenance system TMS [Doyle 1979] and Cohen's reasoning with endorsements [Cohen 1985]

### 2.2.4 Advantages of Rule-based Systems

While there has been criticism of rule-based models for modeling only a subset of the cognitive processes used in expert decision making [Barr and Feigenbaum 1981], they still account for many of the processes used by experts and allow the representation of heuristics in a program.

Barr and Feigenbaum [1981] present some advantages in using rule-based systems. The first is ***modularity***. The knowledge encoded in rules are independent nuggets of knowledge. They can be added, removed or changed independently. Modification of a rule does not have a direct affect on any other rules, although it does have an effect on the total behaviour of the system. Modularity also allows incremental development of an expert system, an advantage that will be discussed later under knowledge acquisition.

A second advantage to rule-based systems is ***uniformity*** of representation. This provides easy recognition of the intent of a rule, although rule interaction can be rather opaque without special tools for viewing

The third advantage is ***naturalness*** which implies that rules model some of the real processes used by experts. Expert domains based on significant

significant patterns can be easily implemented using rule-based systems. However, reasoning from first principles or underlying models is not easily represented with this method, although recent research in the area of induction has suggested that this may be achievable [Holland et al. 1986].

## 2.3 Knowledge Engineering

Now that we have looked at the underlying concepts of rule-based expert systems we will look at the process of creating an expert system. This process is called ***knowledge engineering*** [Feigenbaum 1977]. Knowledge engineering is not a simple process. In general, knowledge engineering involves transformation of a body of knowledge, used to solve problems in a particular domain and mostly contained in the mind of a domain expert, into a computer program such that the program is as good as, or better than the expert at solving domain problems.

There are four modes for engineering an expert system [Buchanan and Shortliffe 1985]. The first mode is called ***handcrafting***. In this mode a knowledge engineer sits down with the domain expert, performs knowledge elicitation and then handcrafts a domain specific program that reflects the expertise. The program may or may not use any expert system techniques. The main benefit of this type of system is that it can become very powerful in the particular domain. A usual problem with such systems is the lack of generality in the implementation. This can create difficulties with extensibility and modifiablity particularly if no effort was made to separate the domain knowledge from the inference mechanism. The knowledge becomes explicitly represented or "hard coded" in the system and is difficult to change.

The second mode of knowledge engineering is similar to the first except the knowledge engineer uses an expert system shell to implement the program. The shell provides explicit use of expert system techniques which enforces the separation of the domain knowledge from the inference mechanism through utilization of knowledge representation and inference models. Expert system shells provide modularity and transparency and assist in the modification of the system as the knowledge base is developed.

This shell-based mode, like handcrafting, has the problem of transferring a body of expertise through a person who is not an expert in the problem domain (the knowledge engineer). This introduces the potential for mis-translation and mis-representation of domain knowledge.

The third mode of knowledge engineering is direct interaction by the domain expert with an expert system shell. This requires a computer conversant expert but at the same time, if care is taken in the development of the interface to the shell and a clear knowledge representation technique is used, some of the problems of transferring knowledge via the knowledge engineer are avoided. A second advantage to this mode is that the knowledge structures are directly developed by the domain expert so that the terms and relationships expressed are his. This creates a closer identification with the developing system. It also provides the expert with a clearer understanding of the formal representation of his problem-solving knowledge in the program. Gaines pointed out in his discussion of KSS0 [Gaines 1987] (a knowledge elicitation tool used to automatically generate rules in an expert system shell) that personal interaction by the expert in the definition of elements and

constructs[1] creates the impression that the generated rules are a direct expression of the expert's knowledge. The generated rules contain the actual terms and relations he uses when reasoning in his domain.

The fourth method of knowledge engineering is through induction techniques. This is a large area of research and a full review is beyond the scope of this thesis. The essence of the concept is that, given examples of problems in the domain and how they are solved, the system induces a knowledge base [Buchanan and Mitchell 1978, Hayes-Roth and McDermott 1977]. Some of the questions raised in this style of knowledge acquisition include, how to express examples, how to generate abstractions, how to learn from examples, and how to learn from metaphors.

[1]Elements and constructs are KSS0 structures that are used to help elicit the expert's knowledge.

# Chapter 3

# Description of PORTAL

## 3.0 Related Work

When the PORTAL project was initiated in 1983 the majority of work in expert systems was occurring in university research laboratories. The major knowledge acquisition tools in existence at that time were the direct outgrowth of attempts to generalize working expert systems. The main knowledge acquisition tools were: KAS [Reboh 1981] (Knowledge Acquisition System) based on the PROSPECTOR expert system [Duda 1979], EMYCIN [van Melle 1981] based on MYCIN [Shortliffe 1976] and EXPERT [Weiss et al. 1978] a model-based system used to develop a number of medical expert systems.

KAS grew out of PROSPECTOR. It used PROSPECTOR's opportunistic reasoning mechanism, semantic nets (for problem data and rules) and a taxonomy of domain objects to allow inheritance and default assumptions.

PROSPECTOR used a semantic network as its underlying knowledge representation structure and KAS provided a network editor. It also had a network syntax checker that contained knowledge about typical network problems such as cycles and disconnected sub-networks. In addition, KAS had a facility for recording unfinished work as well as a method for providing defaults for such uncompleted work. For example KAS provided default user queries for rules that were terminal nodes that had no user specified query. As an expert system tool KAS was a knowledge entry facilitator. It provided help with the syntax and structure of knowledge objects but not for eliciting or forming appropriate knowledge structures.

EMYCIN was a domain independent version of MYCIN. It provided backward chaining through a production-rule/object-attribute-value representation that used certainty factors. EMYCIN contained MYCIN's WHY and HOW functions to query the state of the knowledge base. It also used the Abbreviated Representation Language (ARL), an English-like language that could be translated into MYCIN's internal form. The ARL parser provided syntax and legal attribute value checking.

EMYCIN also checked for rule contradiction, i.e., the same antecedents leading to contradicting conclusions, and for rule subsumption. EMYCIN provided runtime trace and debugging facilities along with a library mechanism for storage and execution of test cases.

EXPERT provided a special purpose language for describing models of the problem domain which were compiled into hypotheses, findings and decision rules. It used a standard text editor for knowledge input. EXPERT had no further tools and assumed that the model building language was simple enough to hide the complexities of the representation from the expert, and that the compiler would catch any potential errors during compilation of the model.

### 3.1 Description of PORTAL

PORTAL, like many other expert system shells, grew out of the development of a particular expert system designed for the interpretation of medical laboratory tests. This application motivated a number of design decisions that shaped PORTAL's knowledge representation, inference mechanism and interface.

The domain expert, Dr. M. D. McNeely, chief pathologist of a large medical laboratory, was sufficently intrigued by the project to want to work

directly on developing the expert system. PORTAL's knowledge base editor and the tools for entering data, running the system and viewing the results were all designed to allow incremental development of an expert system by a domain expert.

Dr. McNeely's long term goal is to provide interpretation services for all aspects of the laboratory, but it was felt that the initial project should be a prototype implemented in one area of laboratory interpretation. The determination of liver dysfunction from lab tests was selected. This domain was chosen because although it is relatively small, there is no simple algorithmic method for determining liver dysfunction from lab tests. The interpretation methods in this domain require the recognition of salient patterns of test results and clinical data, developed through years of experience with numerous cases. The resulting expert system was called **L**aboratory **I**nvestigation and **V**erification through **E**xpert **R**easoning (LIVER).

Certain characteristics of this domain led to a number of decisions about the form of the inference engine was to take and the type of representation to be adopted. The first decision was to use a forward chaining rule-based system. The interpretation of laboratory test results involves recognizing salient patterns of abnormal and normal test values. These patterns are extremely well suited to representation in a rule-based system. A typical LIVER rule is rule 267, one of the rules for recognition of Gilbert's Syndrome.

RULE 267
  IF
    TOTAL BILIRUBIN is between 1.2 and 3 mg/dl and
    INDIRECT BILIRUBIN is greater than 1 mg/ml and
    SGOT is less than 25 IU and
    ALKALINE PHOSPHATASE is less than 95 IU and

PROTEIN ELECTROPHORESIS is NORMAL and
WHITE BLOOD COUNT is between 5000 and 8000 /cu and
ERYTHROCYTE SEDIMENTATION RATE is less than 30 mm/hr
THEN
GILBERTS SYNDROME (0.85)

Dr. McNeely's reasoning methodology involved both the recognition of significant patterns which would lead directly to diagnosis of particular diseases and the recognition of patterns indicative of general conditions that, in combination with other disease specific tests, would lead to diagnosis of particular diseases. An example of a general condition is cholestatic condition, expressed in rules such as:

Rule 360
If
Total Bilirubin is greater than 1.3 mg/dl and
SGOT is less than 40 IU and
Alkaline Phosphatase is greater than 85 IU and
Uric Acid is less than 8 mg/dl
Then
Cholestatic Condition (0.8)

This condition could then be used with other tests to lead to particular diseases through rules such as:

Rule 55
If
Pregnancy is likely and
Cholestatic Condition is likely
Then
Cholestasis of Pregnancy (0.8)

Rule 557
If
Polymorphonuclear Neutrophils is greater than 70% and
White Blood Count is greater than 10000 /cu and
Cholestatic Condition is likely
Then
Cholecystitis (0.8)

Conditions are examples of heuristic classifications that result from experience in interpretation. They are informal categories that the expert uses to group test results without having to commit to a particular disease.

Identification of salient patterns in a set of data is easily implemented by forward chaining. Given data values that satisfies each of the tests in the LHS of rule 360, the rule can be fired. This would result in the conclusion of cholestatic condition being added to working memory. If the test results of rule 557 or the clinical data of rule 55 were also available, then the conclusions of cholecystitis or cholestasis of pregnancy would be made.

Definitive diagnosis of liver dysfunction requires more information than that of laboratory tests and the sparse clinical data available at a medical laboratory. There can be, and usually is, more than one possible diagnosis for a given set of lab data. The solution to a problem in the interpretation of lab tests is not necessarily the identification of a particular disease but more often a list of the most likely diseases. This criterion led to the decision fire multiple rules during each recognize/act cycle, rather than a single rule. This allowed PORTAL to simultaneously advance all possible interpretations for a given set of data. This technique was further extended by adding certainty factors that

tors that created an ordering on the conclusions generated for a given set of data. This allowed PORTAL to effectively deal with incomplete and uncertain data and to express uncertainty in the generated conclusions. This combination of certainty factors with multiple rule firings per recognize/act cycle resulted in the creation of a unique inference mechanism. A full explanation of PORTAL's inference mechanism is given in chapter 4.

Another aspect of the domain that influenced the design of PORTAL was the nature of the data. Pathologists reason about test results and clinical data for a single person at a time. The facts and conclusions generated by the expert system are therefore simple attributes of individuals. Thus the representation scheme used in PORTAL required a simple technique for describing these attributes. Statements like:

Total Bilirubin is 1.6 mg/dl (1.0)
Protein Electrophoresis is normal (1.0)
Hepatitis B (1.0)

are typical examples of medical laboratory data and conclusions We used an unstructured representation of domain objects called ***entities***. Each object is given a name that is used in working memory statements to represent domain data, and in the rules to represent references to the domain object. A fuller discussion of entities occurs in Chapter 4.

The above quasi-English examples of data and rules are exactly how PORTAL displays knowledge structures to the user. PORTAL's internal representation consists of Lisp structures such as:

```
(267 (&AND (BETWEEN* (VAL (QUOTE TOBI)) (QUOTE (1.2 3)))
           (GREATER* (VAL (QUOTE INBI)) 1)
           (LESS* (VAL (QUOTE SGO)) 25)
           (LESS* (VAL (QUOTE ALPH)) 95)
           (EQUAL* (VAL (QUOTE PREL)) (QUOTE NORMAL))
           (BETWEEN* (VAL (QUOTE WBCW)) (QUOTE (5000 8000)))
           (LESS* (VAL (QUOTE ESRE)) 30))
  ((GISY 0.85)))
```

for the Gilbert's syndrome rule 267 and

```
((TOBI 1.0 1.6 ((PREMISE)))
 (PREL 1.0 NORMAL ((PREMISE)))
 (HEA 1.0 ((PREMISE))))
```

for the data set shown above. PORTAL's interface translates these Lisp structures into English-like forms. These translations are more easily understood by the user than the underlying Lisp structures. The user also interacts with the system using the English form which PORTAL dynamically translates into its internal form.

### 3.2 Implementation of PORTAL

PORTAL was implemented in Franz LISP on a Vax 11/780 running Berkeley UNIX 4.1bsd. PORTAL has since been ported to CommonLisp on a μVax running VMS, the environment at the medical laboratory. Lisp was chosen as the implementation language for two reasons: it had the necessary symbolic processing capabilities with which to implement a forward-chaining expert system, and it also had the terminal control capabilities with which a user interface could be built.

Lisp was chosen over Prolog because the inference mechanism was to be forward chaining and because of the user interface capabilities of Lisp If the expert system had been a backward chaining system then PROLOG's logic-based resolution mechanism would have been ideal. However the system we

required was forward chaining and we desired finer control of the inference mechanism than that available through PROLOG.

Theoretical aspects of the inference engine, particularly the conflict resolution mechanism and the consistency mechanism, were the result of discussions with Dr. Chang. The next test strategy was also strongly influenced by discussions with both Drs. Chang and McNeely. All other aspects of PORTAL are the work of the author, especially the knowledge representation techniques, the rule and entity editor and the implementation of PORTAL in Lisp.

# Chapter 4

# Knowledge Representation and Inference Mechanism

In this chapter we will look at PORTAL's use of these representation and inference techniques, first with a discussion of PORTAL's knowledge representation and then a discussion of its inference implementation.

PORTAL's knowledge representation contains two data types, ***entities*** and ***rules***. Entities are used to define objects in the domain, the things about which the expert reasons, and rules are used to define relationships between entities. These relationships are an expression of the heuristics used by the expert to reason in the domain. PORTAL uses forward chaining inference with uncertainty as its main reasoning technique. Forward chaining inference provides a method for deriving conclusions from any given set of data. Uncertainty allows an expression of belief in the conclusions given the expert's encoded knowledge, as well as allowing the system to deal with partial and incomplete knowledge.

## 4.0 Entities

Entities are the basic unit of knowledge in PORTAL. They are similar to constants in predicate calculus. They are place and value holders for data and concepts in the expert system's knowledge representation. Every object about which the expert reasons must be defined as a entity. In LIVER each laboratory test result, each item of clinical data and each disease type has an

associated entity. An entity has two major uses: first it is used to represent its associated real world object in the database of facts, called the ***fact-base*** in PORTAL; second, an entity is used as a term in rule predicates.

In PORTAL there are three types of entities: ***state***, ***numeric*** and ***enumerated***. These types were created as being sufficient to represent all the of data used in the LIVER system.

State entities represent states or conditions in a domain. State entities have no value of their own. They simply represent the existence of some condition in the domain. They are the closest entity type to a classical predicate calculus constant. In LIVER, state entities represent the existence of a disease or condition, e.g., Hepatitis A, Hepatocellular Condition.

Numeric entities represent domain objects that have a numeric value. In the LIVER system these include laboratory test results as well as numeric clinical data such as a patient's age. Enumerated entities represent data that have a non-numeric value, such as a patient's sex or race.

Each entity has a set of attributes that further define the object. These are called ***properties***. The particular properties for an entity are defined by its type. All entities have a ***name*** property which is the only property of a state entity. The value of an entity's name property is an alphanumeric string chosen by the expert, that uniquely identifies the object. Entity names that are proper subsets of another are considered to be distinct. This allows entities to represent both generic and specific concepts. For example, Hepatitis, Hepatitis B, Hepatitis A, Non A Non B Hepatitis all represent different entities. This is purely a semantic tool and there is as yet no structural relationship among such entities.

An entity is implemented in PORTAL as a LISP symbol that is used in the internal representation of rules and facts. The actual LISP symbol is automatically generated by PORTAL from the name given by the user when the entity is defined. For example, the LISP symbol for Gilbert's Syndrome is "GISY". All references to an entity in the user interface use the name property when referring to the entity while all internal references use the generated LISP symbol. As far as the user is concerned the name for the entity is the symbol for the domain object.

Numeric entities have two other properties that further define the data they represent. The first is the ***units*** property which allows the expert to specify units for the values the entity represents. The Total Bilirubin entity has a value of "mg/dl" for its units property. The second numeric property is a ***normal range*** property. This property allows the expert to specify a range of values that the entity normally would have. In the LIVER system, this property represents the range of values that a numeric entity can have if a patient is not sick or if a laboratory test does not reflect the current problem. This allows the expert to use rule predicates based on whether or not an entity's value is within a normal range, which is a common statement in laboratory test analysis. The normal range for Total Bilirubin for example is 0 to 1.2 mg/dl.

An enumerated entity has another property in addition to its name property: the ***values*** property. This property contains a list of values which are the unordered specification of the possible values the entity can have. For example the Protein Electrophoresis entity's value property contains the values "normal" and "abnormal" which would be the possible values for the entity.

The enumeration of the appropriate entities for a given application is more of an art than a science. The expert must explicitly specify all domain objects about which he reasons. Some of these are standard domain objects, such as the Total Bilirubin or White Blood Count in LIVER, while others are internal representations that are part of the expert's internalization of experience. In LIVER, these are represented by entities such as Hepatocellular Condition and Cholestatic Condition

Two heuristics for creating entities became evident during the development of LIVER. Firstly, the expert should enumerate as many domain entities as possible prior to rule creation. Secondly, a standard nomenclature should be used. If the entity naming process is *adhoc* and entities are created as needed, redundant and duplicate entities will be created. This happens particularly if the system is being developed over a long period of time. In the LIVER system, a problem developed in the naming of cancer-related conclusions. "Metastasis", "carcinoma" and "cancer" are among the many names of cancer. These homonyms resulted in the use of multiple entities for the same object. This problem arises when the heuristics used to conclude a particular diagnosis are distributed over a number of entities having different names. The effect is to weaken the overall belief in the entity, even if all the requisite data are present.

## 4.1 Classes

PORTAL entities are grouped into three classes. This is a reflection of the expert's model of laboratory test analysis. Entities in the first class, called ***test*** or ***initial,*** represent data given to the system as initial values for the prob-

lem being considered. In LIVER, these are laboratory test results and clinical data. The second class of entities, ***intermediate*** or ***condition***, represent inferences from groups of initial data that are typical of a class of diseases. The third class, ***disease*** or ***final***, are entities that represent conclusions the expert can make.

These three classes reflect the structure of reason by Dr. McNeely (Figure 4.1). Given test data for a particular problem, represented as entities in the initial class, the expert uses heuristics that lead from the general test data to groups or syndromes of liver disease, called conditions. Of course, there exist values of general tests which directly support specific diseases as conclusions. Intermediate conditions are represented as state entities in the intermediate class. Once intermediate conditions have been identified, the expert uses heuristics that combine intermediate conclusions with disease specific tests to support the selection and ordering of further test in order to make diagnoses represented in the final or disease class.

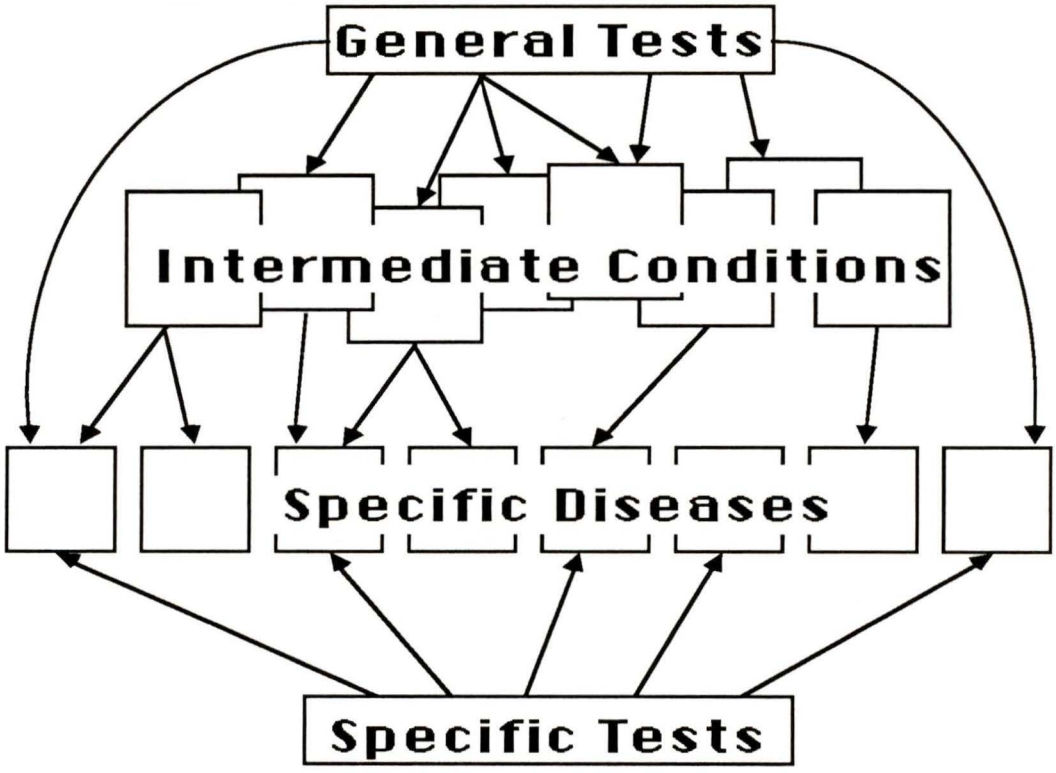


Figure 4.1
Diagnostic Strategy

## 4.2 Facts

The data that comprise a particular problem and its solution are called *facts* in PORTAL. Each fact for a problem is represented by its corresponding entity, with a certainty factor, and a value if the entity is numeric or enumerated. Facts given to the system by the user are called ***premises*** and are always assumed to be true (a certainty of 1.0). Facts generated through firing rules using the inference mechanism are called ***deductions***. Each fact maintains a list of the rules used to generate it, or that the fact is a premise. This list is called the fact's ***justification***. The certainty of a deduction is a function of the

certainty of the justifications and their data. The collection of all the premises and deductions for a given problem is called the fact-base.

An entity is instantiated as a fact in the fact-base if there is an associated real value in the problem being considered, and that information is given to PORTAL. For example, if the test result for Total Bilirubin was available it would be entered in the fact-base as:

Total Bilirubin is 1.6 mg/dl (1.0)

This states that the value of Total Bilirubin is 1.6 mg/dl, with a certainty of 1.0

An entity can also be instantiated if a rule that concludes the entity has been fired. For example given the data:

Total Bilirubin is 11 mg/dl (1.0)
Patient Age is 55 (1.0)
Sex is male (1.0)

and the rule:

Rule 115
If
Patient Age is greater than 50 and
Total Bilirubin is greater than 10 and
Sex is Male
Then
Hepatic Cancer (0.6)

The entity representing Hepatic Cancer would become instantiated in the fact-base:

Hepatic Cancer (0.6)
Total Bilirubin is 11 mg/dl (1.0)
Patient Age is 55 (1.0)
Sex is male (1.0)

and its justification would be Rule 115.

The existence of a disease or intermediate condition is represented by a state entity and a certainty value in the fact-base as in the Hepatic Cancer example above. Problem data such as laboratory test results and clinical data are

represented by numeric or enumerated entities along with the corresponding data values and a certainty factor as are Total Bilirubin and Sex above.

### 4.3 Rules

PORTAL uses rules to represent the expert's heuristics. The rule-based system methodology allows the expert to name patterns of significant data within his domain and use these patterns to generate conclusions. These conclusions can, in turn, be data for other rules.

The LHS of a PORTAL rule consists of predicates on entities in the domain joined together by connectives. The RHS of a PORTAL rule contains a list of entities with certainty factors and for numeric and enumerated entities, values, that can be asserted if the rule fires.

#### 4.3.1 Predicates

Predicates in PORTAL allow the expert to express his heuristics as predicates on the existence and/or value of an entity in the fact-base. The interpretation of these predicates is a function of the certainty factor of the entity. Let CF(A) be the certainty factor for fact A in the fact-base. In general if the predicate evaluates to true it returns CF(A). It returns zero minus CF(A) if the predicate relationship is false, and zero if the entity is not in the fact-base.

Each entity type has its own set of predicates. State entity predicates are the unary existence predicates ***likely*** and ***unlikely***.

Hepatocellular Condition is likely

These predicates are functions of the certainty of the state entity. There is only one enumerated entity predicate, the binary predicate ***equal***, which returns the certainty factor of the entity if the current value of the entity is equal to the second term of the predicate, ie.,

Patient Sex is Male

The English translation of the equal predicate used in the user interface is the word "is".

Numeric entities have the most predicates. There are the usual binary predicates ***greater than, less than*** and ***equal*** as well as the tertiary predicate ***between,*** which is true if the value of the entity is between the second and the third terms. The final numeric predicate is the unary predicate ***normal,*** which is true if the current value of the entity is within the normal range given by the expert when creating the entity. Typical examples are:

Total Bilirubin is greater than 10 mg/dl
Alkaline Phosphatase is normal
SGOT is between 200 and 500 IU

The English versions of predicates can appear the same over different types of entities, eg.,

Alkaline Phosphatase is normal
Protein Electrophoresis is normal

where the "is normal" with the Alkaline Phosphatase predicate is the numeric predicate "normal" and the "is" with the Protein Electrophoresis predicate is the enumerated "equal" predicate. PORTAL uses the type of the entity to generate the appropriate internal representation. eg.,

```
(NORMAL* (VAL (QUOTE ALPH)) (NORMVALS (QUOTE ALPH)))
(EQUAL (VAL (QUOTE PREL)) (QUOTE NORMAL))
```

### 4.3.2 Connectives and Negation

PORTAL uses the n-ary connectives ***and*** and ***or*** , and allows nesting to any depth (see the grammar in Appendix A). For example:

Rule 72
  If
    Hepatocellular Inflammatory Condition is likely and

Cholestatic Condition is likely and
Alkaline Phosphatase is greater than 160 IU and
SGOT is between 25 and 200 IU or
SGGT is between 25 and 200 IU
Then
Hepatocellular Condition due to Cholestasis (1.0)
Cholestatic Condition due to Hepatic Inflammation (-1.0)

Negation is also available with the unary connective ***not***. The expert, however, found that the predicates were sufficient to express all his heuristics thus no rule in LIVER uses the ***not*** connective.

### 4.3.3 RHS Formulae

RHS statements in PORTAL instantiate entities as facts or adjust the certainty value of known facts. If the entity being concluded in the RHS of a rule is not in the fact-base then the entity is added to the fact-base with a certainty that is equal to the certainty from firing the rule. If the fact already exists in the fact-base then the certainty of the fact is adjusted as a function of the current certainty of the entity and the certainty of the rule. For example given the facts:

Total Bilirubin is 11 mg/dl (1.0)
Patient Age is 55 (1.0)
Sex is male (1.0)

and the rule:

Rule 115
If
Patient Age is greater than 50 and
Total Bilirubin is greater than 10 and
Sex is Male
Then
Hepatic Cancer (0.6)

The Hepatic Cancer entity would be added to the fact-base with a certainty factor of 0.6

Hepatic Cancer (0.6)
Total Bilirubin is 11 mg/dl (1.0)

Patient Age is 55 (1.0)
Sex is male (1.0)

If the fact-base already contained

Hepatic Cancer (0.4)

Then Hepatic Cancer's certainty factor would be updated to .76[1]. The use of certainty factors allows a single expression of the normal addition, deletion and modification of entities to the fact-base used in other rule-based systems [Brownston et al. 1985].

## 4.4 Inference mechanism

PORTAL's inference mechanism is forward chaining with certainty factors. Before a discussion of the actual algorithm used by the inference mechanism a discussion of PORTAL's use of certainty factors is appropriate.

### 4.4.1 Certainty Factors

To allow the expert to reason using uncertain and partial knowledge, PORTAL uses a variation on the certainty factor mechanism proposed by Shortliffe in the MYCIN system [Shortliffe & Buchanan 1975, Shortliffe 1976].

All entities, whether or not they have been instantiated, have a certainty factor. This certainty factor is called the ***current certainty*** of the entity. For entity **A** it is written CF(**A**). The current certainty of an entity can be explicit or implicit. The current certainty of entities in the fact-base and rules is explicit. The explicit current certainty for an instantiated entity, or fact, can be generated in two ways. An entity that is input by the user as an initial fact has an explicit current certainty of 1.0. This assumes that the data entered by the user is true. Secondly, an explicit certainty for an entity can be

---

[1]Using the CFAccum algorithm from Chapter 2, 0.4 +(0.6 * (1 - 0.4)) = .76

generated through the execution of rules by the inference engine. All uninstantiated entities have an implicit current certainty of zero, there is no knowledge about their certainty as they are not in the fact-base. Instantiated entities can have an explicit certainty of zero which implies that there is conflicting evidence about the belief in the entity.

Each rule in the expert system represents a heuristic that the expert believes is important in making an expert decision. Each heuristic, however, does not necessarily express the same strength of belief or certainty in an entity. Certainty factors allow the expert to express the strength of his belief in the effect of his heuristic on the entity being concluded. For each entity in the RHS of a rule the expert assigns a certainty factor which would be a positive value if the LHS pattern supports the entity and a negative value otherwise.

Shown below are a number of rules that conclude Gilbert's Syndrome. They each have different certainties which reflect the expert's belief in the contribution of the LHS patterns, e.g.,

Rule 718
  If
    Total Bilirubin is greater than 1.5 mg/dl and
    Unconjugate Hyperbilirubinemia is likely
  Then
    Gilbert's Syndrome (0.6)

Rule 267
  If
    Total Bilirubin is between 1.2 and 3 mg/dl and
    Indirect Bilirubin is greater than 1 mg/dl and
    SGOT is less than 25 IU and
    Alkaline Phosphatase is less than 95 IU and
    Protein Electrophoresis is normal and
    White Blood Count is between 5000 and 8000 /CU and
    Erythrocyte Sedimentation Rate is less than 30 mm/hr
  Then
    Gilbert's Syndrome (0.85)

Certainty factors also allow the expert to express rules that reflect belief in conclusions based on partial or incomplete knowledge. For any rule that includes a number of entities in the LHS the expert can write other rules to express his strength of belief in significant subsets of these entities. The above rules for Gilbert's Syndrome exemplify this. Unconjugated Hyperbilirubinemia is a function of the ratio of indirect to direct bilirubin. Thus rule 718 encodes a significant subset of the heuristic pattern in rule 267. The inference mechanism takes care of selecting the most specific rule for a given set of data. This mechanism will be explained later in this chapter.

#### 4.4.1.1 Predicate, Connective and Negation Certainty Factors

The certainty of a predicate on an entity is the current certainty of the entity if the predicate is true, except for the unlikely state entity predicate which is zero minus the certainty of the entity. For example, given the predicate:

Patient Age is greater than 33

and the fact:

Patient Age is 34 (1.0)

the certainty of the predicate is CF(Patient Age is greater than 33) = CF(Patient Age ) = 1.0. If the relationship expressed by the predicate is not true, ie.,

Patient Age is 21

then the certainty returned by the predicate is CF(Patient Age is greater than 33) = 0 - CF(Patient Age ) = -1.0.

The method for combining certainty factors of predicate terms in a connective statement is a simple function on the certainty of the terms. Let $t_1$ ... $t_n$ be the terms of a connective, then the certainty of the connective is:

$$CF((\textbf{and } t_1 \ldots t_n)) = \min(CF(t_1), \ldots, CF(t_n))$$
$$CF((\textbf{or } t_1 \ldots t_n)) = \max(CF(t_1), \ldots, CF(t_n))$$

Intuitively, if **A** and **B** and **C** implies **D** then the certainty of **D** must be a function of the least likely of **A**,**B** and **C**. Conversely, if **A** or **B** or **C** implies **D** then the certainty of **D** is a function of the most likely of the three.

Negation of a formula is the same as evaluation of a predicate where the predicate relation is not true. If c be any formula, it follows that

$$CF(\textbf{not}(c)) = 0 - CF(c)$$

Intuitively, the certainty of the negation of a predicate should apply the same weight to a conclusion as the certainty of the predicate itself. In other words, the negation of a formula simply changes the sign of the certainty returned by the formula. The weight of the certainty remains the same.

#### 4.4.1.2 Rule Certainty

The certainty of an entity concluded by a rule represents the degree to which the expert believes the heuristic expressed by the LHS of the rule will affect the belief in the entity being concluded in the RHS. The new certainty of the entity being concluded by a rule is a function of the certainty factor in the rule, the certainty returned from the evaluation of the LHS of the rule and the prior certainty of the entity being concluded. For example given the rule:

Rule 72
If
Hepatocellular Inflammatory Condition is likely and
Cholestatic Condition is likely and
Alkaline Phosphatase is greater than 160 IU and
SGOT is between 25 and 200 IU or
SGGT is between 25 and 200 IU
Then
Hepatocellular Condition due to Cholestasis (1.0)
Cholestatic Condition due to Hepatic Inflammation (-1.0)

and the facts:

Hepatocellular Inflammatory Condition (0.6)
Cholestatic Condition (0.7)
Alkaline Phosphatase is 200 IU (1.0)
SGOT is 150 IU (1.0)

The fact-base would be updated to include the facts:

Hepatocellular Condition due to Cholestasis (0.6)
Cholestatic Condition due to Hepatic Inflammation (-0.6)

This follows the description of certainty factors as outlined in Chapter 2

### 4.4.2 PORTAL'S inference algorithm

To run a problem against an expert system created in PORTAL the user must input a set of data that defines the current knowledge about the problem. This data forms the initial fact-base. In the LIVER system the initial data consists of laboratory test results and any clinical data available such as the patient's age and sex. Each piece of data entered is assumed to be absolutely true, given a certainty of 1.0 and a justification that it is a ***premise***.

#### 4.4.2.1 Rule Usage

PORTAL applies rules to facts in the fact-base in a forward-chaining manner using a recognize/act cycle. The recognize part of the cycle is divided in two, conflict set generation and conflict set resolution. The act part of the cycle fires the selected rule(s) updating the fact-base as described above.

#### 4.4.2.2 Conflict Set generation

The conflict set is all the rules whose LHS evaluates to a certainty greater than **0.2**[1] based on the current facts in the fact-base. During the first pass through the recognize/act cycle, rules are selected based on the initial data and all the rules in the knowledge base. In subsequent cycles rules are selected based on the initial data and the facts generated on previous cycles. The rule set from which rules are selected in subsequent cycle includes only the rules not placed in the conflict set in previous cycles.

#### 4.4.2.3 Conflict Set Resolution

Classification problems implemented using PORTAL require an ordered set of possible conclusions as the final solution. In PORTAL's interpretation of medical laboratory tests each possible diagnosis is generated together with its certainty factor. PORTAL's conflict resolution supports all possible conclusions by allowing rules for each possible conclusion to be selected and fired on each recognize/act cycle. Conflict set resolution in PORTAL, unlike classical conflict set resolution, can, and usually does, fire more than one rule during each cycle.

PORTAL allows the expert to have more than one conclusion per rule, but the conflict resolution mechanism requires partitioning of the conflict set by conclusions. Thus a rule with $n$ RHS conclusions is really a shorthand for $n$ single conclusion rules. If a rule has $n$ conclusions then $n$ temporary rules are created during the recognize/act cycle, each containing the LHS of the rule and one of the $n$ conclusions. For example, the Gilbert's Syndrome rule 267 actually contains another conclusion for Hemolytic Disease with a certainty of

---

[1]This is an arbitrary value used in the MYCIN system as the least amount of certainty needed to elicit belief in a statement.

.85. Thus rule 267 is really two rules, the one we have seen previously and the rule:

Rule 267
If
Total Bilirubin is between 1.2 and 3 mg/dl and
Indirect Bilirubin is greater than 1 mg/dl and
SGOT is less than 25 IU and
Alkaline Phosphatase is less than 95 IU and
Protein Electrophoresis is normal and
White Blood Count is between 5000 and 8000 /CU and
Erythrocyte Sedimentation Rate is less than 30 mm/hr
Then
Hemolytic Disease (0.85)

To resolve the elements of the conflict set the conflict resolution algorithm initially groups rules that lead to the same RHS conclusion. Thus all the fireable rules that lead to Gilbert's Syndrome would be in the same initial group. PORTAL then partitions each of these groups by the entities in the LHS of the rules in each group. If two rules that lead to the same conclusion have at least one entity in common in their respective LHSs they are placed in the same partition. For Gilbert's Syndrome, the rules:

Rule 718
If
Total Bilirubin is greater than 1.5 mg/dl and
Unconjugate Hyperbilirubinemia is likely
Then
Gilbert's Syndrome (0.6)

Rule 267
If
Total Bilirubin is between 1.2 and 3 mg/dl and
Indirect Bilirubin is greater than 1 mg/dl and
SGOT is less than 25 IU and
Alkaline Phosphatase is less than 95 IU and
Protein Electrophoresis is normal and
White Blood Count is between 5000 and 8000 /CU and
Erythrocyte Sedimentation Rate is less than 30 mm/hr
Then
Gilbert's Syndrome (0.85)

would be in the same group if they were both fireable because they both contain the entity Total Bilirubin. However, the Gilbert's Syndrome rule:

Rule 353
If
Nicotinic Acid Challenge Test is negative
Then
Gilbert's Syndrome (-0.6)

would be in a separate partition.

Finally, PORTAL splits these partitions into groups with rules that conclude with a positive certainty and rules that conclude with a negative certainty. The inference engine then selects the rule in each partition that would contribute the highest absolute certainty and it is fired.

This partitioning mechanism takes into account the interaction of rules with similar LHSs under the assumption that, of the rules leading to the same conclusion with similar entities in the LHS, the most significant should be selected for firing. Intuitively this algorithm says that for all the heuristics applicable for a conclusion, pick ones that have the strongest effect on the certainty. This encourages the expert to create rules that represent significant partial knowledge, without the concern that they will combine with the more complete rules to falsely promote a particular conclusion. The partial rule will simply be ignored in favour of the more comprehensive one by the conflict resolution strategy.

#### 4.4.2.4 Act

The act portion of PORTAL's recognize/act cycle simply updates the fact-base to reflect the conclusions from the resolved conflict set. It uses the certainty factor update mechanism $CF_{accum}$ defined in Chapter 2 using the mechanism shown earlier in this chapter. If a conclusion is already

instantiated in the fact-base, the certainty of the conclusion is adjusted using $CF_{accum}$ to reflect the certainty generated by firing the rule. If the conclusion is not in the fact-base then it is added with a certainty factor equal to that returned by evaluating the rule.

#### 4.4.2.5 Iteration and Termination

The recognize/act cycle continues until there are no further rules to fire. At the end of each recognize/act cycle all rules in the conflict set are removed from consideration for subsequent cycles. This guarantees termination of the inference mechanism as the number of rules always decreases.

### 4.4.3 PORTAL's Conclusions

Conclusions generated by PORTAL are an ordered list of all possible explanations of the initial data. These conclusions are based on the expert's heuristics as represented in the knowledge base. In LIVER this is a list of the likely and unlikely liver diseases. The ability to generate all possible conclusions is an important capability of PORTAL. In many expert systems only the best conclusion is produced. However in many domains there may be multiple causes or explanations for a given set of data.

In addition, problems in the LIVER domain do not usually contain enough information to permit a definitive conclusion. However there are diseases for which definitive tests exist, so that if if the expert has created rules that reflect these heuristics (rules that conclude with a certainty of 1.0), the inference mechanism will be able to conclude a disease with absolute certainty. This occurs in LIVER where there are disease-specific tests such as the "Hepatitis A Antibody IGM" test which, if positive, indicates absolutely that the patient has Hepatitis A:

Rule 18
If
Hepatitis A Antibody IGM is positive
Then
Hepatitis A (1.0)

## 4.5 Next-Test

The ***next-test*** [Chang, McNeely and Gamble 1984b] paradigm provides an extension of the pattern recognition capabilities of PORTAL by allowing the user to query the system after deductions have been made on a set of initial data for the most significant piece of missing data. In the LIVER system this is an important capability as it allows the system to suggest possible tests to conduct given the current knowledge of the patient and his possible diseases.

The next-test mechanism evolved over a number of stages, starting with a brute force enumeration of the possibilities, to a more subtle graph search technique. These are described below.

### 4.5.1 Count of LHS occurrences

The first next-test technique implemented simply counted the number of occurrences of entities not currently in the fact-base and in the LHS of unfired rules leading to the possible diseases. This technique worked on the assumption that experts would write rules that reflected their methods, and if a test entity occurred in a large number of rules then it was probably an important piece of data to collect. This heuristic was effective but did not work in the case where a definitive test was available. In medical practice general tests are requested first in most cases and thus tend to be pervasive in the expert's heuristics as they are most often used as the basis for interpretations. These would show up fairly frequently in this next-test

method. Disease specific tests, however, are only used to test a hypothesis or confirm a particular disease. However, they occur in fewer rules compared to general tests, and therefore would compete poorly in this next-test strategy. In conclusion, this next-test mechanism was good at finding missing general data but poor at selecting definitive disease specific tests.

### 4.5.2 Mintest and Quickmintest

***Mintest*** and ***quickmintest*** were the next generation of next test techniques. Mintest and Quickmintest both work on the principle of minimizing disbelief in the system. This is calculated by selecting the current best conclusions (certainties greater than 0.6)[1] and looking at the unfired rules leading to these conclusions. A list of entities that occur in the LHS of these rules but not in the fact-base is created. For each such entity a value is generated. The entity is temporarily instantiated to that value and the inference mechanism is run with the current data plus the temporary entity. Mintest and Quickmintest return a list of entities ordered by their effect on the total certainty in the system.

The difference between Mintest and Quickmintest is in the number of values instantiated for each entity. Mintest instantiates the entity for each predicate in which the entity occurs within the LHS of unfired rules. Quickmintest selects one value that is representative of the values necessary to make the predicates in which the entity occurs true. This condition was not always achievable but worked empirically. These techniques selected the test that would provide the most information in decreasing the disbelief in the

---

[1]This value was selected after discussion with the domain expert. It represented the lowest value that he would accept for a conclusion that would be "in the running" as a possible disease.

significant conclusions of the system. They worked on the principle that the current "best" conclusions were likely good conclusions and that the next best piece of data would be one that lowered the disbelief in these conclusions

### 4.5.3 Addproof and Lowerproof

The ***addproof*** and ***lowerproof*** commands provide an entity specific technique of selecting the next best test and are part of a larger proposed methodology. In both cases the user selects the entity of which he wishes to increase or lower the certainty. The system then looks at all the unfired rules leading to the entity and tries to find the one that will cause a rule fire. If there is more than one possibility it will select an entity that raises or lowers the certainty of the entity the most. This methodology reflects an aspect of diagnostic strategies used by doctors, that of trying to rule in or rule out a hypothesis.

# Chapter 5

# PORTAL
# Development Environment

The PORTAL development environment consists of a knowledge representation editor, and a number of tools with which to examine, execute and test an expert system as it is being developed.

PORTAL uses a standard teletype interface to allow use of most common terminals and terminal emulators. The interface uses full screen display with command and prompted input. The system can be used over low baud rate lines while still providing a dynamic full screen interface.

## 5.0 Knowledge Representation Editor

PORTAL's knowledge representation editor uses knowledge of the structure of rules and entities to assist the expert in creating rules and entities. He is guided by the use of prompts, full screen display and online help facilities.

### 5.0.1 Entity Editor

Entities can be created at any time in the knowledge representation editor simply by typing in the command **entity**. When creating an entity, the user is asked for the name of the entity. This is the prose phrase that will be used in the interface, e.g., "Hepatitis B" or "Hepatocellular Condition". The phrase is checked against existing entities for uniqueness.

Once the entity is named the system then prompts the expert for the type (numeric, enumerated or state), the appropriate properties for the type of the entity, and the entity's class. The user then exits the entity editor and returns to the rule editor.

### 5.0.2 Rule Editor

Our aim with the PORTAL rule editor was to provide the user with an editor that was easy to use yet able to create arbitrarily complex rules, especially being able to deal with arbitrary nested LHS clauses.

When the user enters the rule editor all rules in the rule-base are available for editing. The user can either edit a specific rule or select subsets of the rules to allow him to deal with rules with common entities. The user can select rule subsets by specifying an entity of interest. This entity can be in either the LHS or RHS. For example, in LIVER the expert can select all the rules that lead to Neonatal Hepatitis and only those rules would then be available for editing. The user can then select a further subset of these rules, e.g., all rules with the test Total Bilirubin in the LHS. This would result in the set of all rules that conclude Neonatal Hepatitis with Total Bilirubin in the LHS.

This capability serves two purposes. Firstly, the expert can work on groups of rules that have similar conclusions, thus focusing his work and allowing him to see only the rules of greatest interest. Secondly, it allows the expert to create rules about an entity at any time but still be able to access and deal with all rules containing the same entity as a unit.

PORTAL's rule editor is called a ***structure-directed*** editor. By "structure-directed" we mean that the interface to the editor focuses the expert on the particular parts of the rule he is working on. If the expert wants to edit

the LHS of a rule he simply selects a rule, and the LHS of that rule is displayed. If he wants to edit a subclause of the LHS, there are commands available for selecting subclauses. These subclauses can be selected using either entity names in the subclause or the major connective of the subclause. Similarly, creation and deletion of parts of the LHS are specified by indicating which subclause is to be created or deleted.

The use of certainty factors causes the RHS formulae to be very uniform. There is only one RHS statement that that is to indicate the certainty with which the entity is to be concluded. The creation of RHS entities are done through the use of prompts in the rule editor. The expert would enter the entity involved, the value it will have, if any, and the certainty factor for the entity. For example, if a rule is to conclude Hepatocellular Condition with a certainty of 0.8, then the user would simply be prompted to enter these values.

## 5.1 Execution and Evaluation Tools

PORTAL provides a number of tools for executing partially completed expert systems and evaluating the resulting conclusions during development. These are also the tools that would be used to run the completed system on real problems. These tools allow the user to enter problem data, evaluate the data by applying the heuristic rules with the inference engine, and print and analyse the resulting conclusions. These commands are described below.

### 5.1.1 Remember and Forget

The ***remember*** and ***forget*** commands provide the interface for entry and deletion of case data. Remember takes an entity and an optional value, and creates an entry in the fact-base for the entity. Remember sets the certainty

of the fact to 1.0 and makes its justification "premise". Remember can also be used to modify the value of a fact in the fact-base simple by re-remembering the fact with a new value.

Forget provides a facility for selectively deleting entries in the fact-base. Entries can be deleted by name, class, premise or conclusion. In addition the entire fact-base can also be erased.

### 5.1.3 Deduce

The ***deduce*** command invokes the forward-chaining inference mechanism. Using the inference mechanism described earlier, the deduce command first removes all non-premise facts from the fact-base and then runs the recognize/act cycle using the rules in the rule-base and the premises. At the end of each cycle the system displays the generated conclusions that have certainty factors greater than 0.8 . The recognize/act cycle continues until the inference mechanism terminates.

At this point the user can use the following commands to display and query the fact and rule-base to display the facts deduced and the rules that were selected, fired and ignored.

### 5.1.2 Print

The ***print*** command is available in both the editor and execution environment. This command provides a number of facilities for displaying different aspects of the knowledge representation, both as it is being developed and as it is being used. Print allows the user to display some or all of the rules and entities and allows the user to view the fact-base by entity class or by premises and deductions.

### 5.1.4 Leadsto

The ***leadsto*** command allows the expert to examine, outside of the rule editor, all the rules that lead to a given conclusion. In addition if there is any data in the fact-base then the rules displayed will indicate whether or not they are fireable. This allows the expert to see which rules are selected as fireable for a given set of data. It also provides, with the justify command, a complete display of the fired and fireable rules at the conclusion of a deduction.

### 5.1.5 Justify

The ***justify*** command allows the user to query the derivation path of a particular conclusion. **PORTAL** will indicate whether the conclusion is a premise or a deduction. If the fact is a deduction then the rules actually fired in generating the conclusion are displayed. The justify command answers the question "why is this fact in the fact-base?"

### 5.1.6 Consistency

A fundamental problem facing an expert creating an expert system is whether the rules he creates are consistent. The classical logic definition of consistency works on the assumption of a closed world. That is, for a set of formulas $\Gamma$ there exists some formula $A$ in the universe of discourse such that not $\Gamma \vdash A$. Intuitively a consistent set of formulae should not allow you to conclude everything and conversely an inconsistent set of formulas can be used to conclude anything.

One way of maintaining consistency is to create consistent formulae only; ie. if a set of formulas is currently consistent then the addition of a new rule (formula) should not make the new set inconsistent. To facilitate this style of consistency PORTAL has a ***consistency*** command which allows the

There were also subtle differences in the actual terminology especially where the biopsy report was a prose description of the findings rather than a particular disease. This prose description had to be interpreted in the LIVER's terminology. This lead to the creation of a mapping from the test case descriptions to LIVER's disease entities. The problem with this mapping was that there was not always a one to one mapping. This resulted in difficulties in comparing conclusions.

Secondly, the case data only included a standard battery of tests, and did not include any of the disease specific tests available to LIVER. This meant that the precision of the test diagnosis was less than if the more specific tests had been available.

The main problem that came to light from this test was interpretation of the results. Given a standard battery of tests, the conclusions generated would include the correct result but could also include other similar interpretations. As was indicated earlier, LIVER's interpretation of laboratory tests provides an ordering on the most likely conclusions. Thus analysis of LIVER's performance with the test data would have to take into account the particular set of data used and the possibility that the particular data could be interpreted in other ways. The question would not be whether LIVER would be able to provide the same answer as a biopsy but whether an expert given the same data would have provided an answer similar to LIVER's. In other words, it would take an expert to judge the quality of the generated answers. This level of comparison was not attainable simply from a comparison with the biopsy result.

## 6.1 Major Contributions

Besides providing an environment in which an expert can develop rule-based expert systems, a significant contribution of this work is the conflict resolution mechanism of the inference engine. The use of rule partitions and certainty factors in conflict resolution to simultaneously put forward multiple conclusions is a method unique to PORTAL. While this method would not work for all expert system domains it does provide a mechanism for domains that perform diagnostic and pattern recognition tasks based on uncertain and incomplete knowledge.

Empirically, PORTAL has been successful in that it has allowed an expert to directly create his own expert system, as evidenced by LIVER as well as another expert system for the interpretation of anaemia-related diseases. PORTAL has also been distributed to a number of universities and research facilities.

Another benefit of the development of LIVER was the work on the next-test strategies [Chang, McNeely and Gamble 1984b]. These strategies could, with further development, provide a significant increase in the utility of the system especially in the laboratory test domain. Medical laboratory tests can be very expensive, especially ones that are disease specific. With the use of the next-test strategy the focus of the laboratory investigation can be more clearly directed and result in a more cost-effective utilization of laboratory testing.

## 6.2 Extensions

There are a number of extensions to the PORTAL system that are being considered, but with each extension one has to consider the balance between

simplicity and power of representation. The thrust of this work is to provide a model that is sufficiently powerful to perform the task at hand while being simple enough to be accessible to the domain expert.

### 6.2.1 Representation

One of the areas in which it is possible to extend the capability of PORTAL is in the structures for representing knowledge. It has been pointed out that PORTAL's entity representation was chosen because it was simple. One extension that may provide a more general and powerful representation and still maintain some of the simplicity of entities is to provide a frame-based hierarchical system.

#### 6.2.1.1 Frames and Hierarchies

The use of frames does not necessarily imply a hierarchical structure, and vice-versa. Either of these features could add a number of capabilities to the system. A hierarchical structuring of entities would allow the system to capture some of the semantic information seen in the system of entity names. For example, the **Hepatitis** entity could represent the hepatitis disease group and **Hepatitis B, Hepatitis A,** would then implicitly represent the specific diseases.

The use of frame-like structures would allow the creation of more complex data objects that could represent objects in the domain, like a person, with a number of attributes or slots such as name, age, sex further defining the person. Of course, if the system is not reasoning about more than one patient at a time such a structure is just syntactic sugar.

The problem with these extensions is that of maintaining an environment within which it is easy for the expert to directly create and manipulate

these structures. Either the new representation must either be limited or a good interface to the frame representation must be created.

#### 6.2.1.2 Justification /TMS

Another extension to the PORTAL representation could be the addition of a symbolic truth maintenance system (TMS). The current justification list for each fact in the fact-base along with additional structures could be used to allow the retraction of premises and the subsequent modification of the facts based on the retracted information.

#### 6.2.1.3 Rule Extensions

There can be two fundamental extensions to the functionality of rules in PORTAL. However they increase the complexity of the representation and correspondingly the complexity of the tools and interface.

The LHS of PORTAL rules could be extended to include user specified predicates. Since the LHS of a rule is evaluated with the LISP eval function, implementation of arbitrary LHS predicates would be trivial for an experienced LISP programmer. Predicates would simply return a certainty factor based on the degree of belief in the predicate given a set of facts. There would also be some modification of the conflict resolution mechanism to allow selection of rules with such predicates. However, the problem is not the ability to create arbitrary predicates but one of providing a language and a set of tools with which a non-LISP programmer or domain expert could create such predicates. The whole point of this system is the use of a simple yet strong paradigm. Creating yet another language to specify predicates is moving the level of complexity towards that of systems used by a knowledge engineer.

A second extension of PORTAL's rule functionality is the inclusion of arbitrary RHS actions. While this is a simple implementation problem it would cause a major change in the types of rules the expert creates. The current method of specifying multiple RHS entity conclusions is really a short hand for individual rules for each RHS entity. An extension to allow actions would mean that individual rules would have to be created for each current RHS conclusion. The addition of RHS actions, like arbitrary LHS predicates would also necessitate the creation of an action specification language and a set of tools for writing the procedures, again making the system more difficult for the domain expert to use.

#### 6.2.1.4 Rule Sets

Currently all the rules in the system are in one global rule-base. The extension to multiple rule sets would provide the ability to have cooperating experts or topic specific rule subsets of the domain.

### 6.2.2 Inference mechanism

The main extension to the inference engine would be generalization of the next test paradigm to allow full backward chaining. The current addproof command is really a restricted form of backward chaining and the data structures are already in place for the implementation.

Another extension would be to look at implementing some aspects of the work on induction by Holland, Holyoak, Nisbett and Thagard [Holland et al. 1986]. This is a logical extension of PORTAL's multiple rule firing mechanism that provides a more formal basis for this particular aspect of PORTAL's conflict resolution strategy.

### 6.2.3 Dynamic Consistency Checking

Another fairly trivial extension would be to partially automate the use of the consistency check by having the system suggest such a check after every editing session. A future research topic would be the development of a mechanism to support semantic consistency checking, which would require a domain specific understanding of inconsistency within rule interactions

### 6.2.4 Delivery Environment

The current delivery environment is exactly the same as the knowledge acquisition environment. A separate, less powerful environment would be appropriate for a delivery system. The current data entry mechanism using "remember" and "forget" commands is a bit cumbersome and a check-list or form-filling interface for data input may be more suitable. The current display facility using the print command could also be extended in the delivery environment to provide a clearer and more automatic display of the important and salient conclusions of the system.

## 6.3 Conclusions

Expert system technology provides representation and procedural mechanisms for encoding certain specialized human expertise. There are two basic problems to be overcome in transforming the expert's cognitive model of his domain into a expert system model. The first problem is to acquire the expert's knowledge and the second is to program this knowledge in a computer.

PORTAL provides an environment in which the domain expert can directly program his expertise into an expert system. This lessens the former problem of knowledge acquisition but compounds the latter one of knowledge representation. To facilitate direct programming by the domain expert,

PORTAL provides a simplified expert system language consisting of two simple elements in its knowledge representation: entities and rules. These structures provide a model for implementing certain types of expert systems, particularly those useful in classification and diagnosis.

PORTAL provides an interface to allow the expert to directly program his expertise into the knowledge base through the use of a structure directed editor and tools for viewing the knowledge base. It also provides tools to execute, test and refine the expert system as it is being developed.

PORTAL's essential strength lies in its ability to be quickly comprehended by an expert by providing simple representation techniques and by providing a knowledge acquisition and representation environment which allows him to directly transfer his expertise into an expert system.

## BIBLIOGRAPHY

Adams, B. J. (1985). Probablistic Reasoning and Certainty Factors, In Buchanan, B. G. & Shortliffe, H. E. (eds). *Rule-Based Expert Systems*, Reading, Mass: Addison-Wesley Publishing Company, 263-271.

Aikins, J. S. (1980). Prototypes and Production Rules: A Knowledge Representation for Computer Consultation. *Ph.D Dissertation, Satnford Univ. Computer Science Department.* STAN-CS-80-814.

Barr, A. B., & Feigenbaum, E. A. (eds.) (1981). *The handbook of artificial intelligence, Vol 1.* Los Altos, CA.: William Kaufman, Inc.

Barr, A. B., & Feigenbaum, E. A. (eds.) (1982). *The handbook of artificial intelligence, Vol 2.* Los Altos, CA.: William Kaufman, Inc.

Boose, J. (1984). Personal construct theory and the transfer of human expertise. In *Proceedings of the National Conference on Artificial Intelligence.* Austin Texas.

Brackman, R. J. (1979) On the Epistemological Status of Semantic Networks, In N. V. Findler (ed). *Associative Networks*, Orlando, Florida: Academic Press Inc.

Brownston, L., Farrell, R., Kant, E. and Martin N. (1985).*Programming Expert Systems in OPS5. An Introduction to Rule-Based Programming.* Addison-Wesley Publishing Company, Inc., Reading, Mass. 1985.

Buchanan, B. G. & Duda, R. O. (1982) Principles of Rule-Based Expert Systems. Fairchild Technical Report No. 626, FLAIR Technical Report No.9 and Stanford Heuristic Programming Project Report No HPP-82-14.

Buchanan, B. G. and Mitchell, T. (1978) Model-directed learning of production rules. In eds D. Waterman and F. Hayes-Roth, *Pattern-Directed Inference Systems*, NewYork: Academic Press, pp. 297-312

Buchanan, B. G. and Shortliffe, H. E. (eds) (1985). *Rule-Based Expert Systems*, Reading, Mass: Addison-Wesley Publishing Company.

Chang, E. J. H., McNeely M. D., Gamble K. I., "An Expert System For Liver Function Tests", *Proceedings JCIT*, 1984a.

Chang, E. J. H., McNeely M. D., Gamble K. I., "Strategies for choosing the Next Test in an Expert System", *Proceedings AAMSI.,*. 1984b.

Chang, E. J. H., McNeely M. D., Gamble K. I., "Validation of LIVER" Presented at IA-BIOMED86, Montpellier, France.

Clocksin, W. F., & Mellish, C. S. (1984). *Programming in Prolog,* Berlin: Springer-Verlag.

Cohen, P. R. (1985). *Heuristic Reasoning with Uncertainty,* London: Pitman Publishing Limited.

Davis, R. and King, J. J. (1977). "An overview of production systems", In E.Elcock and D Michie (eds.), *Machine Intelligence 8.* Ellis Horwood Chicester, England.

Davis, R. & King, J. J. (1986). The origin of rule-based systems in AI, In Buchanan, B. G. & Shortliffe, E. H. (eds.) *Rule-based Expert Systems.* Reading, MASS: Addison-Wesley Publishing Company, 20-54.

Davis, R. and Lenat D.B. (1982)*Knowledge-Based Systems in Artificial Intelligence,* McGraw-Hill, New York.

Davis, R. (1986) "Application of meta-level knowledge to construction, maintenance and use of large knowledge bases", Department of Computer Science, Stanford University, HPP-76-7, 1976.

Davis, R., & Fox J. (1985) "Expert Systems Part 1" *Conference Tutorial Program,* Ninth International Joint Conference on Artificial Intelligence, UCLA.

Dijkstra, E. W. (1972). The humble programmer, *Communications of the ACM* **15**, No. 10, 859.

Doyle, J. (1979). A truth maintenance system. *Artificial Intelligence* **12**. 231-272.

Duda, R., Gaschnig, J., & Hart, P. E. (1979). Model design in the PROSPECTOR consultant system for minearl exploration. In D. Michie (ed.), *Expert systems in the micro-electorinc age.* Edinburgh University Press, 153-167.

Duda, R. O., "A computer-based consultant for mineral exploration" Final Report SRI project 6415, SRI International, September 1979.

Feigenbaum, E. A. (1977). The art of artificial intelligence: Themes and case studies of knowledge engineering. In *Proceedings of the Fifth International Conference on Artificial Intelligence.* Pittsburg, PA: Computer Science Department, Carnegie-Mellon University.

Feigenbaum, E. A., Buchanan, B. G., & Lederberg, J (1971) On Generality and Problem Solving: a Case Study Using the DENDRAL Program", *Machine Intelligence 6*, Edinburgh Universtiy Press.

Feigenbaum, E. A. and McCorduck, P. (1983).*The Fifth Generation, Artificial Intelligence and Japan's Computer Challenge to the World*, Addison-Welsley Publishing Company, Massachusetts.

Forgy, C. L. (1981). *OPS5 User's Manual*, Department of Computer Science, Carnegie-Mellon University.

Gamble, K. I. and Chang E. J. H., *PORTAL Reference Manual*, Laboratory for Computer Enhanced Cognition, Department of Computer Science, University of Victoria, LCEC 0001-85, 1985.

Goldstein, I, & Papert, S. (1977) "Artificial Intelligence, Language , and the Study of Knowledge" *Cognitive Science*, Vol. 1, No. 1.

Greiner, R., & Lenat, D. (1980). A representation language language. In *AAAI* 1, pp.165-169.

Hayes-Roth F. (1984). The Knowledge Based Expert System: A Tutorial, *Computer*, Vol. 17, No 9.

Hayes-Roth F., and McDermott, J. (1977). Knowledge acquisition from structural descriptions. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence* (Cambridge, MA), pp. 356-362.

Hayes-Roth F., Waterman, D. A., & Lenat, D. B. (eds) (1983). *Building Expert Systems*, Reading, MA: Addison-Wesley Publishing Company, Inc.

Holland, J. H., Holyoak, K .J., Nisbett, R. E. & Thagard, P. R. (1986). *INDUCTION Processes of Inference, Learning and Discovery.* Cambridge, MA: The MIT Press.

Kahn, G. Nowlan, S., and McDermott, J. (1985) MORE: An Intelligent Knowledge Acquisition Tool. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence.* University of California at Los Angeles.

Kleene, S. C. (1967). *Mathematical Logic.* New York: John Wiley and Sons.

Kunz, J. C., Kehler, T. P., & Williams, M. D. (1984). Applications development using a hybrid AI development system, *The AI Magazine*, vol 5, no 3, Fall.

Knowledge Craft (1986) *The Knowledge Craft Reference Manual*, Pittsburg, PA: Carnegie Group Inc.

Larkin, J. H., McDermott, J., Simon, D., and Simon, H. A. (1980). Expert and novice performance in solving physics problems. *Science*, **208**, 1335-1342.

LeFaivre. R.A. (1977) FUZZY reference manual. Computer Science Department: Rutgers University.

McDermott, J. & Forgy, C. (1978). Production System Conflict Resolution Strategies. In, Waterman, D. A. & Hayes-Roth, (eds), *Pattern Directed Inference Systems.* New York: Academic Press, 1978.

Michie, D. (1974). *On Machine Intelligence.* New York: John Wiley and Sons.

Minksy, M. (1975). A framework for representing knowledge. In P. Winston (ed.), *The Psychology of Computer Vision*, McGraw-Hill.

Newell, A. & Simon, H. A. (1972). *Human Problem Solving*, Englewood Cliffs, NJ: Prentice-Hall.

Newell, A. & Simon, H. A. (1976). Computer science as empirical enquiry: Symbols and search. (the ACM Turing Lecture.) *Communications of the ACM* **19**, no. 3, 113-126.

Post, E. (1943). "Formal reductions of the general combinatorial problem.",*American Journal of Mathematics*, Volume 65.

Reboh, R. (1981). "Knowledge engineering techniques and tools for expert systems", PhD Dissertation, Linkoping University, No. 71, Linkoping.

Robinson, J. A. (1965). A Machine-Oriented Logic Based on the Resolution Principal, *Journal of the ACM* **12**, No. 1, 23-41.

Sacerdoti, E. D. (1977) *The structure of plans and behavior*. Holland: Elsevier-North

Schank, R. C. (1972). Conceptual Dependancy: A theory of natural language understanding. *Cognitive Psychology* **3**, 552-631.

Shortliffe, E. H. & Buchanan, B. G. (1975). "A Model of Inexact Reasoning in Medicine", *Mathematical Biosciences 23*.

Shortliffe, E. H. (1976). *Computer-based Medical Consultations: MYCIN*, American Elsevier Publishing Co., New York.

Simon, H. A. (1983). Search and Reasoning in Problem Solving, in *Search and Heuristics*, Pearl, J. (ed), Amsterdam: North-Holland, 7-30.

Stefik, M., Aikins, J., Balzer, R., Benoit, J., Birnbaum, L., Hayes-Roth, F. & Sacerdoti, E. (1983). Basic Concepts for Building Expert Systems, in Hayes-Roth F., Waterman, D. A., & Lenat, D. B., (eds), *Building Expert Systems*, Reading, MA: Addison-Wesley Publishing Company, Inc.

Thomason, R. H. (1970). *Symbolic Logic: An Introduction*. London: Collier-Macmillian Limited.

van Melle, W. (1981). *System aids in constructing consultation programs*. Ann Arbor: Michigan: UMI Research Press.

Waterman, D. A. and Hayes-Roth, F. (eds.) (1979). *Pattern-directed Inference Systems*, New York: Academic Press.

Weiss, S., Kulikowski, C., Amarel, S., Safir, A. (1978). "A model-based method for computer-aided medical decision-making", *Artificial Intelligence*, Volume 11.

Weiss, S. M. & Kulikowski, C. A. (1984). *A Practical Guide to Designing Expert Systems.* Totowa, NJ: Rowman & Allanheld.

Williams, C. (1984). Art: the advanced reasoning tool. *Inference Corp. Report,* Inference Corp., Los Angeles CA.

Winston, P. H. (1979).*Artificial Intelligence,* Reading, MA: MIT Press.

# APPENDIX A

## Representation Grammar

### A.1 Rule-base

RULE_LIST := nil | RULE RULE_LIST

RULE := RULE_NUMBER IF THEN

IR := CLAUSE

THEN := THEN_ELEMENT | THEN_ELEMENT THEN

CLAUSE := PREDICATE_PHRASE
| UNARY_CONNECTIVE CLAUSE
| BOOLEAN_CONNECTIVE CLAUSE CLAUSE
| N-ARY_CONNECTIVE CLAUSE CLAUSE
CLAUSE_LIST
| ARITHMETIC_CONNECTIVE VARIABLE_PHRASE
VARIABLE_PHRASE

CLAUSE_LIST := nil | CLAUSE CLAUSE_LIST

PREDICATE_PHRASE := STATE_PREDICATE_PHRASE
| VALUE_PREDICATE_PHRASE
| BOOLEAN_PREDICATE_PHRASE

STATE_PREDICATE_PHRASE := STATE_PREDICATE
VARIABLE_PHRASE

VALUE_PREDICATE_PHRASE :=
PREDICATE VARIABLE_PHRASE
[ VALUE | VALUE-PAIR ]

BOOLEAN_PREDICATE_PHRASE := BOOLEAN_PREDICATE
VARIABLE_PHRASE BOOLEAN_VALUE

STATE_PREDICATE := likely | unlikely

VALUE_PREDICATE := greater* | less* | equal* | between*

BOOLEAN_PREDICATE := equal*

UNARY_CONNECTIVE := &not

BOOLEAN_CONNECTIVE := &and | &or

ARITHMETIC_CONNECTIVE := /

VARIABLE_PHRASE := val ENTITY

THEN_ELEMENT := STATE_ELEMENT | VALUE_ELEMENT

STATE_ELEMENT := STATE_ENTITY CERTAINTY_FACTOR

VALUE_ELEMENT := VALUE_ENTITY CERTAINTY_FACTOR
ENTITY_VALUE

**A.2 Entity Base**

ENTITY_LIST := nil | ENTITY ENTITY_LIST

ENTITY := STATE_ENTITY | VALUE_ENTITY

STATE_ENTITY := ENTITY_NAME [ CONTEXT ]

VALUE_ENTITY := NUMERIC_ENTITY |
ENUMERATED_ENTITY

NUMERIC_ENTITY := ENTITY_NAME [ UNITS ]
[NORMAL_RANGE]
[ CONTEXT ]

ENUMERATED_ENTITY := ENTITY_NAME
[ENUMERATED_VALUES ]
[ CONTEXT ]

ENTITY_NAME := <word phrase>

UNITS := <word phrase>

ENUMERATED_VALUES := nil | ENUMERATED_VALUE
ENUMERATED_VALUES

NORMAL_RANGE := LOW_VALUE HIGH_VALUE

**A.3 Fact-base**

FACT_LIST := nil | FACT FACT_LIST

FACT := STATE_FACT | VALUE_FACT

STATE_FACT := [ STATE_ENTITY | STATE_ENTITY CONTEXT ]
CERTAINTY_FACTOR nil

VALUE_FACT :=
[ VALUE_ENTITY | VALUE_ENTITY CONTEXT]
CERTAINTY_FACTOR VALUE

VALUE := NUMERIC_VALUE | ENUMERATED_VALUE

NUMERIC_VALUE := <number>

ENUMERATED_VALUE := <word>

# VITA

Surname: GAMBLE Given Names: Kenneth Ivan

Place of Birth: Valleyfield Quebec Date of Birth: June 27 1952

Educational Institutions Attended, with Dates of Entering and Leaving:

CARIBOO COLLEGE, KAMLOOPS B.C. 1970 to 1972

UNIVERSITY OF VICTORIA, B.C. 1980 to 1987

Degrees, Diplomas, Etc., Awarded, with Dates and Names of Institutions:

B.Sc (with distinction) 1983 University of Victoria, Victoria

Publications:


Chang, E. J. H., McNeely M. D., Gamble K. I., et al, "The Laboratory For Computer Enhanced Cognition", 1984, Presented at the CIPS Conference, 1984.

Chang, E. J. H., McNeely M. D., Gamble K. I., "An Expert System For Liver Function Tests", *Proceedings JCIT*, 1984a.

Chang, E. J. H., McNeely M. D., Gamble K. I., "Strategies for choosing the Next Test in an Expert System", *Proceedings AAMSI.,*. 1984b.

Chang, E. J. H., McNeely M. D., Gamble K. I., "Validation of LIVER" Presented at IA-BIOMED86, Montpellier, France.

Gamble, K. I. and Chang E. J. H., *PORTAL Reference Manual*, Laboratory for Computer Enhanced Cognition, Department of Computer Science, University of Victoria, LCEC 0001-85, 1985.

PARTIAL COPYWRITE LICENSE

I hereby grant the right to lend my thesis (dissertation if appropriate) (the title of which is shown below) to users of the University of Victoria Library, and to make single copies only for such users or in response to a request from the Library of any other university, or similar institution, on its behalf or for one of its users. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Title of Thesis/Dissertation

PORTAL - An Expert System Shell

Author [redacted]
(Signature)

KEN GAMBLE
(Name in block letters)

Sept 22 1987
(Date)