1987

A Comparative study of four major knowledge representation techniques used in expert systems with an implementation in Prolog

JoAnn T. Hudgick

Rochester Institute of Technology
School of Computer Science and Technology

A Comparative Study of Four Major Knowledge Representation Techniques Used in Expert Systems with an Implementation in Prolog

by

JoAnn T. Hudgick

A thesis, submitted to
The Faculty of the School of Computer Science and Technology,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Approved by: _____

John A. Biles

_____

Andrew T. Kitchen

_____

Peter G. Anderson

September 20, 1987

Title of Thesis: A Comparative Study of Four Major Knowledge Representation Techniques Used in Expert Systems with an Implementation in Prolog

I, JoAnn T. Hudgick, hereby **grant permission** to the Wallace Memorial Library of RIT to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Date June 6, 1989

Abstract

Knowledge representation is a central issue in Artifical Intelligence (AI) research. In order to solve the diverse and complex problems encountered, one needs both a large amount of knowledge and some mechanism for the management and skillful utilization of that knowledge. The basic problem in knowledge representation is the development of an adequate formalism to represent that knowledge. In this thesis I will discuss four of the major techniques for representing knowledge in expert systems: first order logic, production rules, semantic networks, and frames. Using Prolog as the implementation language, I will demonstrate that all of the above mentioned representation techniques, when used in actual implementations, will be reduced to an equivalency - that being a set of Prolog facts and rules. Prolog limits us to a set of facts expressed as "predicate(argument1, argument2, ..., argumentn)" and "IF ... THEN" rules, thus eliminating many of the unique features which characterize the various representation techniques. Therefore, Prolog can be viewed as a representation technique itself.

Table of Contents

Chapter 1.

Statement of the Problem

There exists many different methods for representing knowledge in expert systems, each offering special or unique features. Certain methods support the representation of specific types of knowledge. Various languages have been developed to implement these representation techniques. Is it possible to use one language to implement various knowledge representation techniques? If so, how would the use of a language such as Prolog impact the practical use of different representation systems?

In this thesis I will examine four of the major knowledge representation methods: first order logic, production rules, semantic networks and frames and attempt to write expert systems in Prolog using these four methods. The structure of Prolog's facts, predicate(argument1,...,argumentn) and IF...THEN rules, establishes constraints on the knowledge representation formalism. The attempted implementations of these four techniques are all reduced to the same thing: a set of Prolog facts and rules. Thus, Prolog itself, can be viewed not only as an implementation language, but as a knowledge representation technique. Even though each of the representation techniques, in theory, represents knowledge in a specific way, the implementation language can dictate which technique is feasible.

In chapter 2 we will look at an overview of expert systems, discussing some of the basic design issues and how they differ from conventional computer programs. Chapter 3 is an introduction to knowledge representation in expert systems. We will discuss the organization of knowledge, the types of knowledge (declarative, procedural and control) and examine some of the major concerns in the selection of a knowledge representation technique. In chapter 4 we will focus on logic and knowledge representation. Here various logics are examined: propositional, predicate calculus, first-order predicate logic, fuzzy logic, default reasoning and certainty factors. We will talk about Prolog as an implementation language and as a knowledge representation technique. Next we will look at an expert system built in Prolog using first-order logic. Chapter 5 describes the components of a production system. In chapter 6 semantic networks are examined, and chapter 7 deals with frames. Also, in chapters 4 through 7 we

will compare and contrast the four knowledge representation methods in terms of structure, inference mechanisms, the knowledge base, expressive power and implementation in Prolog. When Prolog is used as the implementation language, we are confined by its structure and therefore loose many of the features otherwise available when using the various schemes. Appendix A describes the knowledge domain used for the expert system. Appendix B contains the Prolog implementationn, and appendices C, D and E illustrate how the knowledge base of appendix A might appear using languages suited for production rules, semantic networks and frames, respectively. Chapter 8 is the concluding chapter.

Chapter 2.

An Overview of Expert Systems

2.1 What is an Expert System?

Definitions of expert systems abound. The underlying theme is that they are information systems that achieve expertise in a particular field. We could debate extensively the issue of "expertise in a field", but for simplicity let us say that if a system does achieve expertise, it emulates the behavior of the human expert in that field. Hayes-Roth and Waterman present a good definition: "An expert system is a computer program that embodies the expertise of one or more experts in some domain and applies this knowledge to make useful inferences."[HAY 83].

Expert systems can give the illusion of expertise based on superficial knowledge, while they lack the insights that human experts acquire over a long period of time. It is unlikely that expert systems will ever be able to mimic the human qualities of intuition and common sense. Nonetheless, expert system designers are attempting to model as closely as possible human thought processes.

Expert systems must be able to solve complex problems at an expert level. They must be able to incorporate new knowledge easily, be easily understandable by both experts and non-experts in the particular field, and should be able to justify their solutions. They are a class of computer programs that can "advise, analyze, categorize, communicate, consult, design, diagnose, explain, explore, forecast, form concepts, identify, interpret, justify, manage, monitor, plan, present, retrieve, schedule, learn, test, and tutor" [MIC 85]. Expert systems are finding widespread application as advisers on subjects ranging from medicine to finance. As embedded systems, they also serve as the "brains" of robots, automatic weapons, and photocopiers.

An expert system derives its problem-solving power primarily from the quality and completeness of its knowledge base. Facts, rules of thumb (heuristics), models, and other general knowledge about solving a particular class of problems (a problem domain) are encoded and stored. In order to solve a

particular problem, the system uses facts about the problem provided by the user plus its domain knowledge and general problem-solving procedures to find and apply a specific solution.

A typical expert system consists of an inference engine, a knowledge base, and a workspace (see figure 2.1.1). The inference engine solves a problem by interpreting the domain knowledge contained in the knowledge base which contains facts and rules. Facts are short-term information that can change, for example, during the course of a consultation. Rules are longer-term information about how to generate new facts or hypotheses from what is presently known. "Marie is a patient" is an example of a fact, and "If the patient has a runny nose then prescribe an antihistamine" is an example of a rule. The workspace is an area of memory set aside for storing a description of the problem constructed by the system from facts supplied by the user, or inferred from the knowledge base during a consultation. A system may also include a natural language interface for communicating with a user, a reasoning explanation subsystem to justify the sequence of inferences, and a knowledge acquisition subsystem for expanding the knowledge base.

In contrast to conventional database systems, expert systems require a knowledge base with diverse kinds of knowledge: knowledge about objects, knowledge about processes, and hard-to-represent commonsense knowledge about goals, motivation, time, actions, etc.

This thesis addresses some of the important issues raised in attempts to represent this breadth of knowledge:

(1) How do we structure the explicit knowledge in a knowledge base?

(2) How do we encode rules for manipulating a knowledge base's explicit knowledge to infer knowledge contained implicitly within the knowledge base?

(3) How do we deal with incomplete knowledge?

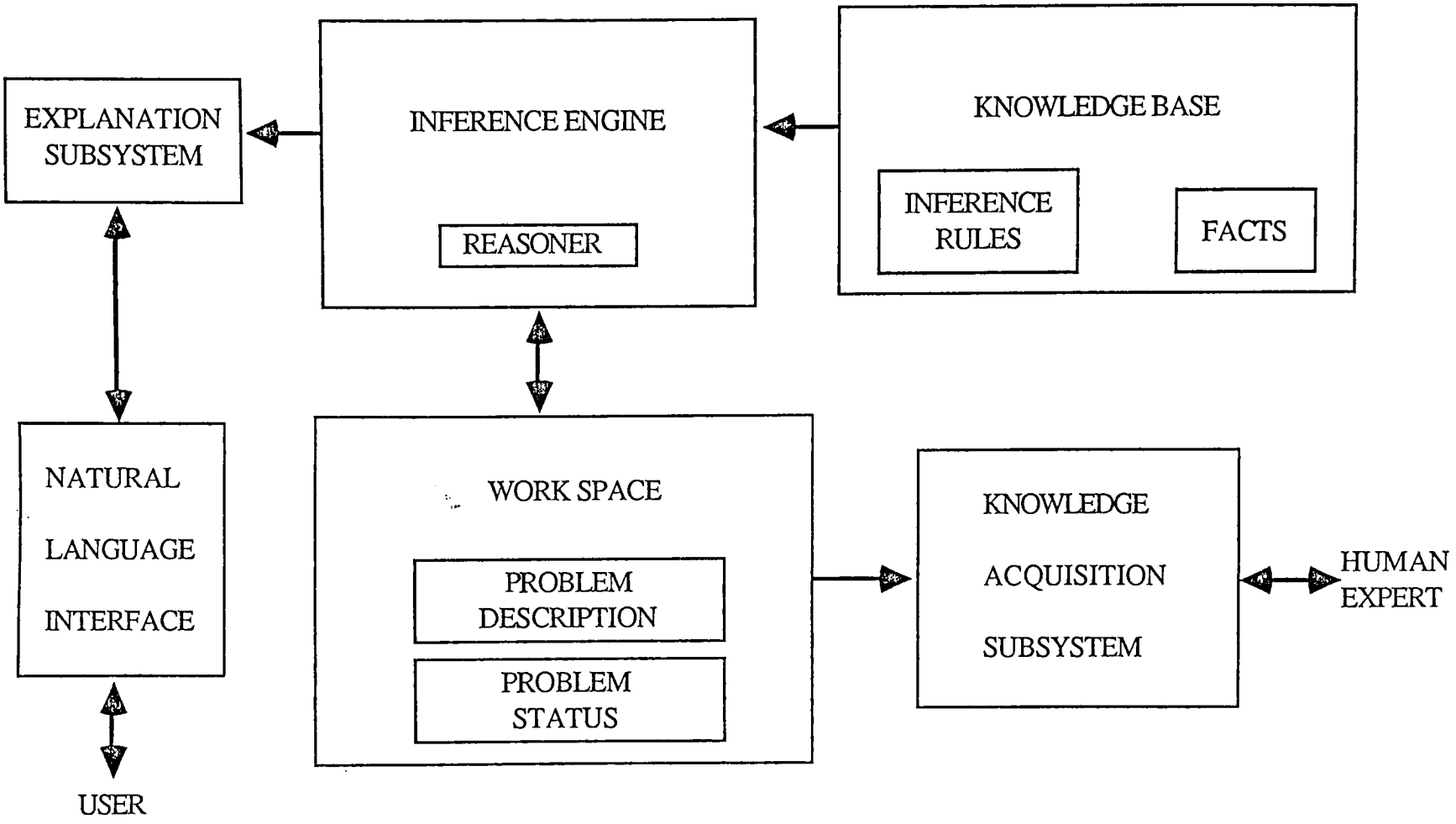


Figure 2.1.1. A Typical Expert System

Chapter 3.

Knowledge and its Representation

3.1 Knowledge and Expertise

Knowledge can be viewed as consisting of both facts and heuristics. The facts constitute a body of information generally agreed upon by experts in a field. The heuristics are rules of good judgement (rules of plausible reasoning, rules of good guessing) that characterize expert-level decision making.

In order for the expert system to reflect an expert's knowledge and ability, it may have to reflect the expert's memory and information processing, problem definition, heuristics and skills.

(1) Memory and Information Processing: Research has shown that experts do not have a larger memory capacity than novices. However, they are better at using their memory capacity then novices. First, experience allows experts to combine or "chunk" small bits of information into larger, meaningful units of knowledge. These chunks are easier to recall from memory. Chunking allows physicians to store and recall complicated procedures. Second, experienced experts are capable of "automatically" linking familiar inputs with appropriate responses stored in memory. An experienced physician quickly recognizes configurations or patterns of viruses. These inputs are automatically linked to a set of possible actions. Because these responses are automatic they save valuable time which would be wasted if deliberation were necessary.

(2) Problem Definition: Experts excel at "decomposing" problems that appear ill-defined to the rest of us. As an example, the difficult problem of building a house is easily broken down by an expert architect into smaller, more structured problems. First, possible constraints might be identified (i.e., size limitations, design, financial limitations). Then the architect considers that the house must have a floor, a roof and a structure. The structure consists of a frame, plumbing, electrical fixtures, etc. In this way, the problem becomes manageable and choices are narrowed to only a few. The expert's identification and classification of problem components is a crucial abil-

ity that must be captured in the expert system.

(3) Heuristics: Experts are able to articulate rules of thumb for solving certain classes of problems. These heuristics are often expressed in an if...then format. They must be identified and incorporated into the expert system. As an example: "if the patient has a head injury, check for a concussion".

(4) Skill: Skill can be thought of as a blending of thought and action in order to efficiently meet a particular objective. This means that a skill requires specialized knowledge combined with a motor response. Performing an appendectomy is a skill because it combines knowledge of the human anatomy with the physical actions to perform the operation. Experts are so practiced that much of what they do is automatic. Because a well- developed skill is automatic it becomes difficult for the expert to describe it in detail. Still, domain experts and knowledge engineers must articulate the components of a skill if it is to be performed by an expert system.

3.2 Organization of Knowledge in Expert Systems

Ordinary computer programs organize knowledge into two categories, data and program. In order for an expert system to "perform the task of an expert" it needs a large amount of knowledge. Therefore, these systems need to be organized to avoid confusion. Generally, knowledge is divided into three categories: declarative, procedural, and control.

Declarative knowledge (factual) represents a particular situation and is usually acquired through a dialogue with the domain expert. It can include facts about objects, about events, and about how they relate to each other. "Declarative knowledge can be retrieved and stored but cannot be immediately executed. To be effective, it must be interpreted by procedural knowledge." [BRO 85]. The way in which such information is represented is significant, as the structure of the representation contains information itself. (This will be discussed in subsequent chapters.)

Procedural knowledge is usually collected in advance from the domain specialist and forms the

core of a knowledge base. "Procedural knowledge can be immediately executed using declarative knowledge as data but may not be examined." [BRO 85]. This forms the reasoning part of the system in order to infer conclusions. Knowledge of how to do things, heuristic knowledge to find short-cut solutions, and non-factual (probablistic and default) knowledge are procedural in nature.

Both declarative and procedural knowledge are necessary in most knowledge domains; therefore, most systems employ a combination of both. The basic idea is to use these facts and rules to create a model that accurately represents something as well as the actions that can be performed by it and on it. Then by testing actions on the model, one can predict what is likely to happen in the real world.

Control knowledge is a computer program that "is used to direct the sequence in which problem-solving steps are carried out" [BRO 85]. A number of control strategies can be used, all with the same underlying objective, to achieve the goal by applying an appropriate sequence of operators to an initial situation. Briefly, we will examine the three most common strategies.

(1) In forward (data driven) chaining the system attempts to reason forward from facts to a solution. The sequence of behavior may be illustrated by the following: Ask a question or take what the user provides; make inferences from that data; ask another question, and so on until an hypothesis (the goal state) has been reached.

(2) Backward (goal driven) chaining is a method in which the system starts with an hypothesis (the goal state) then works backwards to find the evidence supporting the solution. This often entails formulating and testing intermediate hypotheses (subgoals).

(3) The rule value approach or sideways chaining assigns a value to each item of evidence and asks that question with the highest value first. As questions are answered their rule values are modified. The system calculates a rule value, usually applying some probability function, for each item of evidence in the knowledge base.

The specific implementation of a control strategy depends upon the representation of the knowledge it manipulates and on the purpose of the reasoning. Generally, forward chaining is used to deduce all that can be deduced from a set of facts. Backward chaining, on the other hand, is used to

verify or deny a conclusion. Data-driven reasoning will never terminate if the initial evidence does not lead to a conclusion. For this reason, most expert systems are primarily goal-driven. Some systems combine goal- and data-driven inferencing. First, data-driven inferencing is used to suggest a set of hypotheses based on initial data. The system then begins to consider each hypothesis in turn. Whenever a new item of evidence is discovered during backward inferencing, the system switches to forward inferencing to see if the new evidence would suggest another goal or a short-cut to achieving the current goal [KIN 84].

In an expert system, the knowledge about a problem (declarative or procedural) is segregated from the control knowledge and is represented in modular chunks. This makes knowledge based systems relatively easy to modify. In the succeeding chapters we will examine methods of representing declarative and procedural knowledge in expert systems and look at an implementation in Prolog.

3.3 Introduction to Representation

There are many ways in which knowledege can be represented in expert systems. The four most widely used are:

(1) first-order predicate logic

(2) production rules

(3) semantic networks

(4) frames or schemata

When building an expert system it is important that a representation formalism be chosen that is both "usable by the system and comprehensible to human beings" [MIC 82]. There is no prescribed method for accomplishing this; therefore, it is a source of controversy among knowledge engineers.

When discussing representation we must differentiate between facts and their representations.

Facts are truths in some domains. These are the things we want to represent. Representations of the facts are the things we will actually manipulate.

In order to get information into and out of the system there must be some sort of mapping (function or relation) from facts to representations and from representations to facts. A natural language interface may be necessary to facilitate this transfer of information, but in this case we will also need a mapping from the natural language sentence to the representation, and from the representation back to the sentence (see figure 3.3.1) [RIC 83].

For example, the fact

Washington was a U.S. president

can be represented as

USpresident(Washington).

The fact

All U.S. presidents must be native born citizens

can be represented as

USpresident(x) -> native-born(x).

Using deductive reasoning the representation of a new fact

native-born(Washington)

is generated. From this fact, using the appropriate mapping function, we could generate the sentence (fact)

Washington was a native born citizen.

This example and those to follow in this chapter illustrate the use of predicate logic as the

representation scheme. This will be discussed in depth in chapter 4.

An important question that needs to be answered, regardless of the representation formalism we choose, is "At what level of detail should the knowledge be represented?". In other words, "What should be the primitives?". Should we use a small number of low-level primitives or many that cover a range of levels? [DAV 82]. For example, suppose we were to look at the following fact:

Infotech employs Susan.

This could be represented as

employ(Infotech,Susan)

and the question "Does Infotech employ Susan?" could easily be answered. But, the question "Does Susan work at Infotech?" could not be answered directly from this one fact. We would need to add another fact to the knowledge base,

employ(x,y) -> work-at(y,x)

to infer the answer. Here we have added more primitives to cover a wider range of levels.

On the other hand, if we represented the same fact in terms of a small set of primitives such as,

```
pays(agent(Infotech),
     object(Susan),
     to-perform(work))
```

we have broken the term "employ" down into more primitive concepts of "pays" and "to-perform(work)". With this representation we are not directly able to answer either of the questions. But, "Who pays Susan to perform work?" could be answered directly.

The major advantage of representation in terms of a small set of primitives is that the rules of inference need be written only in terms of the primitives rather than in terms of the many ways the knowledge may originally have appeared.

The major drawbacks are that a great deal of work is required to convert each fact into its low-level primitives. This leads to another problem. In certain domains it may not be clear what the primitives should be or there may not be enough information in the high-level forms to convert them to their primitive forms. A great deal of generality and potential design optimality are lost when restricted to low-level descriptions. It appears that the lower the level we choose, the less inference is required, but the more inference is required to create the representation from natural language.

What, then, is the correct level of representation? This is to a large extent dependent upon the domain itself and to what use the knowledge is to be put. In general, knowledge in a natural language understanding system is represented as a small set of low level primitives, whereas in most expert systems knowledge is represented by a larger set of primitives covering a range of values. Whatever level we choose it is essential that we bear in mind that the power of an expert system is primarily a function of the quality and completeness of its knowledge base. In order to convey the intended meaning of a piece of knowledge it is important that we represent it appropriately. Thus, to convert natural language sentences into some other type of representation, we must first decide what facts the sentences represent then convert those facts into the chosen representation.

If an expert system is to emulate human intelligence, it must contain a great deal of declarative and procedural knowledge. A key question throughout the history of AI has been how to best represent this knowledge. At present there is no "best" technique. "There is no theory of knowledge representation. We don't know why somes schemes are good for certain tasks and others are not. But each scheme has been successfully used in a variety of systems that do exhibit intelligent behavior." [BAR 82].

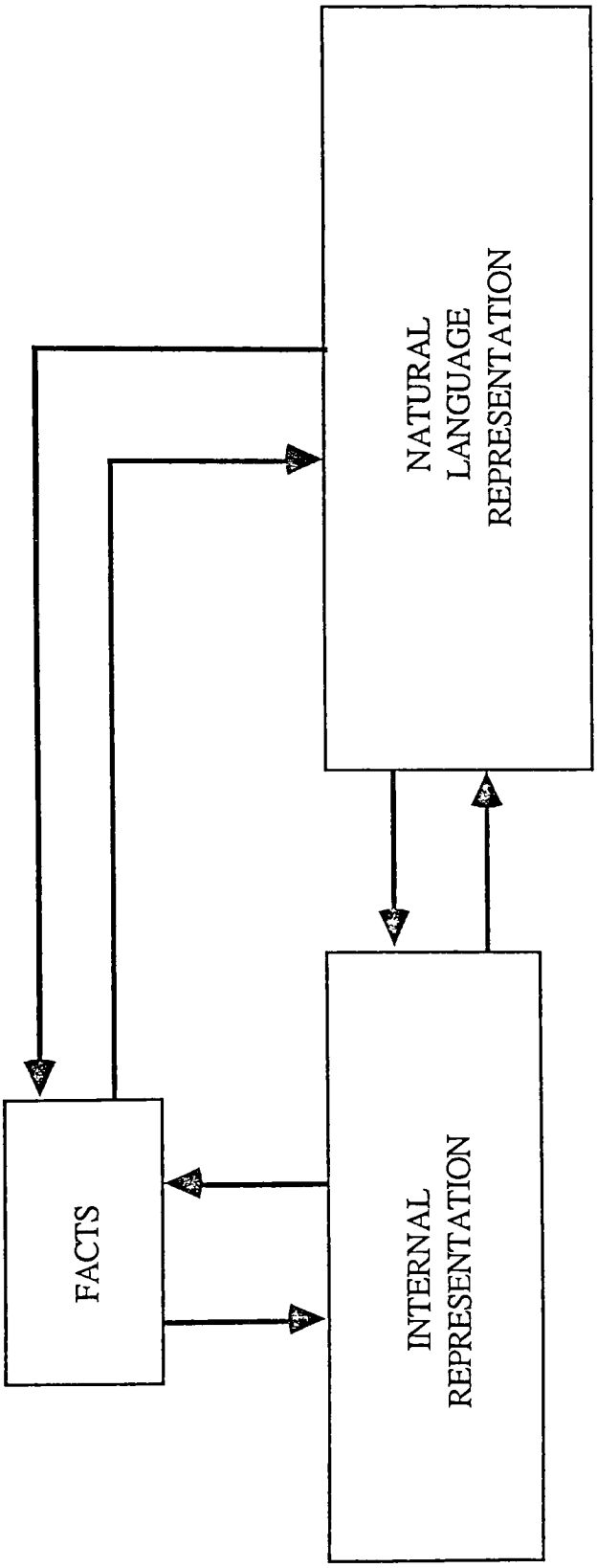


Figure 3.3.1. Mapping Between Facts and Representations

Chapter 4.

Logic and Knowledge Representation

4.1 A Brief History of Logic

Deductive reasoning first appeared in ancient Greece about 600 BC. Thales of Miletus appears to have been the first to attempt a proof using some sort of deductive reasoning.

For a long time the study of logic was made difficult because logicians had to use words in their analysis of the reasoning process, and the meaning of the words is often vague and imprecise. The English mathematician George Boole (1815-1864) was the first to attempt to express the operations of logical reasoning in the language of a calculus. He showed that any class of objects, for example, "analgesics" or "all purple things", can be represented by a symbol such as "x". He then showed that such symbols can be combined by the same rules as those that govern the operations of algebra.

Logic has proved to be relatively straightforward to implement on a computer which is, after all, based on theories of mathematical logic. In this chapter we will examine logic as a method for representing and reasoning with knowledge in expert systems.

4.2 The Logic Debate

The usefullness of logic (first-order logic) in a knowledge representation context became evident during the 1960's primarily as a result of research into mechanical theorem-proving. Much research was directed at investigating the use of the resolution principle as an inferencing technique in various applications (i.e., question-answering). Other research tended toward viewing logical formalisms in a more computationally oriented framework (i.e., Planner [ROS 79] and Strips [FIK 71]). This has led to intense discussion regarding the pros and cons of logic based approaches to representation. Two

major opposing viewpoints are expressed below. John McCarthy [Stanford University] believes that the way to solve the knowledge representation problem is to design computer programs to reason according to the well-worked-out languages of mathematical logic, whether or not that is actually the way people think. "Whether logical reasoning is really the way the brain works is beside the point. This is Artifical Intelligence and so we don't care if it's psychologically real." [KOL 82]. Marvin Minsky [MIT] believes that a good approach is to try to get computers to imitate the way the human mind works, which, he thinks, is almost certainly not with mathematical logic. "Logical reasoning is more appropriate for displaying or confirming results of thinking than for thinking itself. That is, I suspect we use it less for solving problems than we use it for explaining the solutions to other people and much more important - to ourselves." [MIN 82].

4.3 Logic and Reasoning

Logic is the study of valid reasoning. In logic we are interested in the truth value of a given statement; that is, whether a sentence is consistent within a given theory. There are two important components in a logical system axioms and deductive reasoning. Axioms are the statements accepted without proof; they are statements about the relationships between objects and implications about the objects, for example,

(1) Birds have feathers
(2) An eagle is a bird

Deductive (formal) reasoning is the process of arriving at a conclusion by determining what can be inferred from certain axioms. From the above two axioms we can conclude the fact

(3) An eagle has feathers

This deduction is permitted based on the syntax of the two axioms and the idea that the new facts derived through the application of rules of inference are always true as long as the original facts are

true. We can prove that the statement "An eagle has feathers" is true by chaining backward from the fact to the axioms. For example, to prove the statement

An eagle has feathers

is true, we move backwards from this statement to the statement established earlier

An eagle is a bird

thus, concluding by the law of transitivity, the statement

A bird has feathers

is true.

Let's rewrite these two statements to show how the deduction process works:

All birds have feathers
bird (X) -> feathers (X)

An eagle is a bird
bird (eagle).

This deduction requires the substition of "eagle" for the variable X. The substitution is found by a recursive pattern matching algorithm called unification which compares two literals and discovers whether there exists a set of substitutions that makes them identical. The basic idea in an attempt to unify two literals is to check if their first elements match. If so, continue with the second, and so on, calling the procedure recursively. Only identical predicates, constants, or functions can match. A variable can match another variable, any constant, or a function or predicate expression with the restriction that the function or predicate expression must not contain any instances of the variable being matched. It is necessary that a consistant substitution is made for the entire literal [HUN 75]. For example, if we want to unify the expressions

(flies A A)
(flies B C),

the first elements (flies) match. Next we compare A and B and decide to substitute B for A, denoted B/A. (We could just as easily have substituted A for B.) Now the second elements will match. Continuing, the last pair does not match, so another substitution must be made. We must be consistent in our substitution - we cannot substitute B for A in one pair and C for A in another. Therefore, we need to substitute B for A in the remaining literals. Now we can unify these literals by substituting C for B. The unification procedure returns a list containing the substitutions that were performed. Thus, (B/A, C/B) is returned.

One of the most straightforward rules of inference used in proof procedures is Modus Ponens which says that if there is an axiom of the form X1 -> X2 and there is another of the form X1, then X2 logically follows. In our example above, if we assume that both (1) and (2) are true, then Modus Ponens tells us that (3) also must be true.

The mathematician J.A. Robinson proposed another method of problem solving based on a principle called "resolution". Resolution is a method of inference that says, if there is an axiom of the form (X1 OR X2) and another axiom of the form (NOT X2 OR X3), then (X1 OR X3) logically follows. The expression (X1 OR X3) is called the "resolvent" of (X1 OR X2) and (NOT X2 OR X3) [WIN 84].

The resolution principle is refutation complete, which allows it to be used in generating proofs by refutation. In other words, to prove a statement, resolution attempts to show that the negation of the statement produces a contradiction with the known statements. Basically, this is done by converting all statements to a standard clause form. Then, taking two clauses (parent clauses) that each contain the same literal (in positive form in one clause and in negative form in the other), the resolvent is obtained by combining all of the literals of the two parent clauses except the ones that cancel. If the clause that is produced is the empty clause, then a contradiction has been found. In other words, the resolution process takes a set of clauses all of which are assumed to be true and generates new clauses that represent restrictions on the way each of these original clauses can be made true, based on information by the original clauses. A contradiction occurs when a clause becomes so restricted that there is no way it can be true. This is indicated by the generation of the empty clause. We have reasoned backwards from the statement we want to show is a contradiction, through a set of intermediate conclusions,

to the final contradiction.

Resolution has proved to be both valid and complete. In other words, the resolution process continues until either a contradiction is found, no progress can be made, or a predetermined amount of time has elapsed. This procedure, then, is not guaranteed to halt if given a nontheorem (a nonprovable statement) to prove. It is guaranteed to halt and find a contradiction if one exists.

Let us look at a simple example [HUN 75]. Suppose we are given the axioms shown in figure 4.3.1 and we want to prove that "John is not sick". First we convert the axioms and the statement to be proved to clause form as shown in figure 4.3.2. We then negate "John is not sick", producing "John is sick". Next we begin selecting pairs of clauses to resolve. We start by resolving with the clause "John is sick" since that is one of the clauses that must be invalid in the contradiction we are trying to find. Figure 4.3.3 illustrates the sequence of resolvents that might be generated. This leads to a contradiction, and since all the axioms in our list were assumed to be true, it cannot be true that NOTin(john,office) and in(john,office) are both true. Therefore, the assumption that led to this contradiction, sick(john), must be false, hence NOTsick(john) must be true.

"Modus Ponens can be viewed as a special case of resolution because anything concluded with Modus Ponens can be concluded with resolution as well" [WIN 84]. For example, according to Modus Ponens, if X1 and (X1 -> X2) are true, then X2 must be true. Since (X1 -> X2) can be written as (NOT X1 OR X2), resolution can be applied. We see that X1 and NOT X1 cancel, thus producing X2. This is the same result obtained using Modus Ponens.

Similarily, resolution is also a generalization of another rule of inference called Modus Tolens. According to Modus Tolens, if there are two axioms, (X1 -> X2) and (NOT X2), then (NOT X1) logically follows.

Resolution has been shown to be quite successful as a theorem prover given a small set of simple problems. However, as the number of statements in the domain increases, this method becomes increasingly slow due to the combinatorial explosion of choices in resolvents. Strategies do exist for making the choice that can speed up the process considerably. See [ROB 68] for further discussion.

4.4 Knowledge Representation Using Predicate Logic

There are several species of logic. Two of the most commonly used are propositional calculus and predicate calculus. In propositional calculus simple facts are represented as logical propositions written as simple constants. Propositions are statements that are either true or false. The propositions can be linked together with connectives such as, AND, OR, NOT, IMPLIES, and EQUIVALENT. Figure 4.4.1 illustrates some simple facts represented in propositional logic. From fact 5 we can easily deduce (infer) using modus ponens that tomorrow is Tuesday if today is Monday. (i.e., if x is true and (x implies y) is true, then y is true).

Although it is simple and straightforward, propositional calculus has serious limitations. If we want to represent the facts that

Watson is a new student

and

Holmes is a new student

we could write

watsonnewstudent

holmesnewstudent

but would not be able to make any inferences about the relationship between Watson and Holmes. We are even in more of a dilemma when we try to represent the fact,

All freshmen are new students,

because we would have to write separate statements for each new student.

We can conclude that as a representational formalism in expert systems, propositional logic is weak. It does not adequately capture relationships between objects and generalizations of these rela-

tionships over classes of objects.

A solution to this problem is to use predicate calculus. Predicate calculus is an extension of propositional logic in which facts are represented as statements written as well formed formulas (wff's). The five connectives from propositional logic are retained but the focus is changed. We no longer look at statements merely for their truth value. In predicate calculus we are interested in representing statements about specific objects and relations among those objects.

Statements about objects are called predicates. Predicates can address one or more objects, that is, have one or more arguments. For example,

(1) **new-student(watson)**

asserts that Watson is a new student, and this assertion is either true or false. The statement

(2) **son-of(watson,doctor)**

asserts that Watson is the son of a doctor. We can express any one-argument expression as a two-argument expression. Thus, new-student(watson) can be written

(3) **is-a(watson, new-student).**

Quite often we need to use two types of statements called quantifiers to make statements about sets of objects. The universal and the existential quantifiers provide the mechanism to express relationships between a group of objects and make inferences from these relationships. Following is a brief explanation of these quantifiers.

A variable is a symbol used to represent an object. An open sentence is a sentence that contains one or more variables, for example x + 2y = 7x. Two statements can be made about an open sentence, p(x), having variable x:

(4) **for all x, p(x) is true (universal quantification)**

(5) **there exists an x such that p(x) is true (existential quantification).**

Statement (4) is symbolized as

$$\forall \text{ x, p(x),}$$

and is taken as true if and only if for every meaningful value for x, p(x) is true. Statement (5) is symbolized as

$$\exists \text{ x, p(x),}$$

and is true if and only if there is at least one value of x that makes the statement true. As an example, the sentence "All freshmen are new students" can be expressed as

$$\forall \text{ X, freshman(X) -> new-student(X)}$$

and is interpreted as "For all X, if X is a freshman, then X is a new student". The statement "Some freshmen eat custard" can be expressed as

$$\exists \text{ X, freshman(X) ^ eat-custard(X)}$$

and is interpreted as "There exists at least one X, such that X is a freshman and X eats custard".

The statement

$$\forall \text{ X, p(X)}$$

can simply be written p(X), thus eliminating the "for all" quantifier. This allows a move from a general statement to a specific statement. For example, from the statement

$$\forall \text{ X, freshman(X) -> new-student(X)}$$

we can infer

freshman(watson) -> new-student(watson).

Many situations require more than just a TRUE or FALSE answer. There are also times when we need a large number of simple, similar facts such as

less-than(1,2) **greater-than(2,1)**

less-than(2,3) **greater-than(3,2)**

and we do not want to have to write each of these individually. Augmenting predicate calculus with computable predicates or functions is a branch of logic known as "first-order logic" (FOL) or "first-order predicate logic" (FOPL). The two desirable situations mentioned above (returning a value other than true or false and eliminating the need to represent similar facts individually) can easily be achieved using FOL. For example, to evaluate the statement

less-than(2 * 3,4),

the value of the multiply (*) function is first computed then the value of the less-than predicate is computed using the results from the multiply. Using variables, we would eliminate the need to store all the required statements:

less-than(X * Y,Z).

The result of this evaluation is TRUE or FALSE. Looking at another example (from statement (2)), the statement

son-of(watson,X)

would return "doctor", i.e. X would be instantiated to (replaced by) "doctor", when the son-of function is evaluated.

From statements (1) and (2) we can see that that one way to represent declarative knowledge is by means of formulas in first-order logic. Procedural knowledge can also be represented in FOL, if the formulas are suitably interpreted. For example, the formula

$$B_1 \& B_2 \& \cdots \& B_n \rightarrow A$$

can be thought of either as a logical statement that A is true whenever

$$B_1, B_2, \ldots, B_n$$

are true, or as a procedure for producing a state satisfying condition A. In other words, if you want A then do

$$B_1 \text{ then } B_2, \ldots, B_n.$$

In the next section we will examine a programming language that allows us to represent knowledge in terms of logic.

4.5 An Overview of Prolog

The idea that logic could serve as a general purpose programming language was made a reality in 1972 by A. Colmerauer and P. Roussel in the form of Prolog [COL 83]. The introduction of Prolog has made it possible to represent knowledge in terms of logic and to draw inferences automatically -- PROLOG stands for PROgramming in LOGic and the inference mechanism is built into the language. Prolog was at first a theorem prover based on the resolution principle. It has undergone much reformulation, and today seems like an appropriate tool to use in an expert system employing a FOL representation. Both FOL and Prolog use resolution as the basis for inference. The control mechanism provided by Prolog is depth-first search with backtracking. The order of the appearance of the facts and rules in the data base also guides this control process. The essence of a depth-first search is to select an alternative at some point then move forward until the goal is reached or it is no longer possible to move forward. When further forward motion is not possible the search is backed up to resume at the nearest point (nearest ancestor of the failed point) that has an unexplored alternative.

As mentioned in section 4.4, a statement of the form

$$B_1 \& B_2 \& \cdots \& B_n \rightarrow A,$$

can be thought of as a logical statement (fact) or as a procedure (rule). Prolog exemplifies this idea. It can be thought of as a representational formalism a tool for representing knowledge. The basic units for building facts or rules are predications, i.e., expressions that say simple things about the objects in our universe. For example, the fact

watson loves custard

is represented

(1) loves(watson,custard)

and the rule,

watson loves anything that is custard,

could be represented as

(2) loves(watson,X) :- custard(X).

This is not only a rule of Prolog, but also a formula of FOL, with meaning, "for all X, if X is custard, then watson loves X". The ":-" is read "if". Rules have the general form

$$P_1 \text{ if } (P_2 \text{ and } P_3 \text{ and } \cdots P_n).$$

If all the conditions

$$P_2 \cdots P_n \text{ hold, then } P_1 \text{ holds.}$$

Once a set of facts and rules has been defined for a domain, we can use Prolog to check the validity of our assumptions about the domain. We can even ask Prolog to find information that can be deduced from those facts and rules. This is done by writing a query. For example, given (1) and (2) above, we can verify that watson does indeed love custard by writing the query

loves(watson,custard)?

to which Prolog will print some indication of agreement, such as, "yes" or "no". If we want to obtain information about what watson, loves we can write

loves(watson,X)?

This causes Prolog to find instances that make the predicate (loves) true. In this case it finds "X = custard".

Prolog answers a query

$$\mathbf{P_1} \text{ and } \mathbf{P_2} \cdots \text{ and } \mathbf{P_n} \text{ ?}$$

by taking each P_n in turn and trying to find it among the facts and rules. If found among the facts, P is considered proven. If it appears in a rule of the form

$$\mathbf{P_i} \text{ if } \mathbf{Q_1} \text{ and } \mathbf{Q_2} \cdots \text{ and } \mathbf{Q_m}$$

then Prolog tries to prove each and all of

$$Q_1, \ldots, Q_m$$

in the same fashion. For a more detailed explanation see [CLO 81].

Logic programming, fundamentally, is a way of doing an orderly search in attempting to solve a problem (see section 4.6). One of Prolog's strengths is that it has been specifically designed to employ searching and backtracking as fundamental operations. By contrast, in Lisp, though you can program searches, they are not an inherent part of its execution.

The appeal of Prolog as a knowledge representational tool is that the representation is in a rule format. This format is nice because you can collect bodies of knowledge in a fragmentary and accumulating kind of way. You don't have to write the whole program from top to bottom before you can try it out.

As pointed out in section 4.2, AI researchers have debated the appropriateness of adopting as a representational formalism a logic based language. Prolog may not provide the mechanism to fully represent all bodies of knowledge but it is just one of many available tools.

4.6 Implementation

In choosing a knowledge domain I was looking for a subject that was not biased towards any of the four representation schemes. It was a difficult task. The domain I chose is trivial, but does satisfactorily illustrate each of the schemes.

Having always been a Sherlock Holmes fan, I was inspired by the movie "Young Sherlock Holmes". I decided upon a particular scene to implement (see appendix A).

The implementation models the reasoning process that Sherlock Holmes uses to make his deductions about a new student who just arrived at Westbury High School. (See appendix B). The knowledge base contains facts based on Holmes' observations of the student. Some of the facts are:

(1) **All students at Westbury High are males.**

(2) **The new student is carrying a suitcase - the name tag reads J. Watson.**

(3) **The new student is carrying books.**

(4) **The new student's coat and shoes are of a particular style, typical of his native city.**

Along with the facts a series of rules is written to gather the information needed to make the deductions. One of the rules is:

(1) **He is a native of city Y if the length of his coat is C, his lapels are L, and his shoes are S.**

In a FOL based system as well as in Prolog, the facts and rules are stored in the knowledge base

in relation form: predicate(argument1, argument2, ..., argumentn). For example, the fact

His last name is Watson

is written

last_name(watson)

and the rule stated above is represented as

```
native_of(Y) :-
        coat_length(C),
        lapels(L),
        shoes(S),
        stlye(Y,C,L,S).
```

Once the facts and rules are written they are used to draw conclusions. Prolog queries the data-base of facts and rules and automatically produces the conclusions Holmes would ultimately have made. For example, the expert system might be asked to determine if James is a native of London. First it would phrase the problem as a premise to be proved, such as: James is a native of London. Then the program would search for rules to establish the conditions for being a native of London, such as: If James is wearing a long coat with wide lapels and wing-tip shoes, then he is from London.

In its attempt to reason, the program applies various combinations of these and other rules to establish the home town of the new student. The program searches for data, in the form of rules, that will help it either prove or refute the correctness of the original premise in this case, that James is from London. Along the way, the search undoubtedly follows some false leads. When the futility of such a path becomes obvious, the program must backtrack and begin again along an alternate route. For example, rule (1) will succeed if each condition in the rule is satisfied. The rule itself contains references to other rules which in turn must succeed. If any of the conditions fail, the rule fails. Thus if the student is wearing sneekers, none of the shoes' rules will succeed therefore the first native_of rule will fail. The second native_of rule will print a message indicating the native city of the student is unknown.

This is a simplistic, straightforward example but one can see how easily it was implemented in Prolog. The information was expressed in a logic language, a logic query was used to ask questions and logical deductions were made to answer those questions.

4.7 Conclusions

We have examined the use of FOL as a knowledge representation technique and have looked at its implementation in Prolog. The effectiveness of a knowledge representation technique in carrying out a certain task depends on how it is used. Decisions need to be made on how to organize the knowledge base, what set of predicates should be used, and what arguments should these predicates take. These decisions leave out important distinctions that might be valuable later, such as meanings or relationships among objects. It is important, therefore, to choose the representation technique that best enables us to capture the intended meaning of our knowledge domain and also provides a mechanism for solving the problem at hand.

Predicate logic has the virtue of being well-developed and, according to John McCarthy, at least is the "natural" way to express certain ideas. It provides a simple formal precision (syntax), a clear interpretation of an expression (semantics), and a powerful way of deriving new knowledge from old - mathematical deduction. Systems based on FOL are monotonic in a sense that the number of statements known to be true is strictly increasing over time. New statements can be added to the knowledge base, but, will not cause a previously known statement to become invalid. In domains of complete information, FOL would seem to be an adequate tool.

While strict FOL techniques can be used to solve problems in a wide variety of domains, especially mathematical theorems, it is inadequate for reasoning with uncertain or incomplete information. There is no straightforward way of representing ambiguity, degree of certainty, and time relations. For example, it would be very difficult to capture the intent of the following statements using strict FOL:

(1) **The Chicago Bears will probably win the Super Bowl.**

(2) **Some people find jogging very boring.**

(3) **Unless otherwise told, assume that all birds can fly.**

A great deal of the reasoning that people do involves incomplete, vague, and inconsistent knowledge. Therefore, it is necessary to construct a formalism that can handle this kind of information.

In the next section we will look at other logics that provide a mechanism for handling default reasoning and vague and uncertain knowledge.

4.8 Other Logics

4.8.1 Incomplete Knowledge

Traditional (monotonic) logics cannot handle generalizations and reasoning with incomplete knowledge. Such reasoning is critically important in real-world systems. The knowledge base in an expert system is created by humans and much human knowledge is vague. Facts and rules generally are neither totally certain nor totally consistant. For this reason it is necessary to equip a system with the computational capability to deal with this issue. Three approaches will be discussed briefly.

It is not the intention of this thesis to fully expound upon these approaches but to note that other logics exist that provide a mechanism to handle incomplete knowledge. Each of these approaches, in themselves, could be a thesis topic.

4.8.2 Default Reasoning

Common sense reasoning is nonmonotonic in the sense that we often draw conclusions on the

basis of partial information, and later retract those conclusions when we are given more complete information. Default reasoning is a type of nonmonotonic reasoning in which plausible inferences are drawn from less than conclusive evidence in the absence of any information to the contrary. The conclusions are tentative, so given better information, they may be withdrawn. For example, suppose that you want to buy a birthday gift for your niece. You feel that a popular rock album would be appropriate since most young people like rock music. You have no specific knowledge to indicate one way or another whether your niece likes rock music, so you buy a rock album. You made a decision, based on a general rule, with no evidence to the contrary. This is default reasoning. If you later discover that your niece rarely listens to rock music, since she prefers jazz, you must delete your previous belief that she likes rock music and any other beliefs that are based on that belief.

Default reasoning is a common and accepted method of reasoning in real world situations. Decisions need to be made in a timely fashion, thus, for lack of "better" or contradictory information defaults are used.

4.8.3 Fuzzy Logic

In predicate logic a statement is either true or false. No provisions for dealing with uncertainty are provided. No gradations from the truth are allowed. The statement "Some people find jogging very boring" cannot be represented in FOL because the terms "some" and "very boring" are vague (fuzzy).

Fuzzy set theory is a formalism for representing statements involving vague or fuzzy knowledge. Each object in a fuzzy set is assigned a degree of membership, ranging from zero to one. Figure 4.8.3 illustrates an example of the fuzzy subsets "tall" and "not middle-sized" [NEG 85]. The numbers represent degree of membership in the set, that is, measure the plausability of an element being in a particular set.

The use of fuzzy set theory is best applied when a decision does not have a clear analytical or intuitive answer because there are too many facts to consider. Making a decision when faced with

several alternatives which at face value appear equally good or desireable can be a time-consuming and sometimes painful process. Through the use of fuzzy set theory, the facts can be organized and the necessary calculation can be performed with the aid of a computer.

Although fuzzy logic is a fairly well-developed theory, it has not yet been exploited significantly in expert systems. For a detailed study see [NEG 85].

4.8.4 Certainty Factors

There are situations where we cannot be completely certain if some facts are true or certain causal relations hold. The idea of "certainty factors" was introduced to try and address this issue. Certainty factors are informal measures of confidence or certainty for a piece of evidence. They represent the degree to which we believe that evidence is, in fact, true. They can be attached to facts as well as to rules. There are a number of different ways of representing certainty factors. Below is a brief summary of the basic principle.

Associated with each fact or rule is a certainty factor (CF) which indicates the certainty with which each fact or rule is believed to hold. The CF is a number, say in the interval [-1,1]. If the CF is positive, the system believes the fact or rule holds within the particular domain. If the CF is negative, the system believes that there is more evidence that the fact or rule does not hold in the domain. For example, a rule of the form

IF A and B and C, THEN D

asserts D when A, B, and C are certain. A number can be associated with D to indicate one's degree of belief that D follows from A, B, and C:

IF A and B and C, THEN D (.6).

Mycin, an expert system that diagnoses infectious diseases and recommends treatment, has associated with each rule a CF indicating the expert's level of confidence in the rule. The CFs for the rules are provided by the physician at the time that the rule is entered into the system. Facts are represented as four tuples, such as [BUC 84]

(SITE culture-1 blood 1.0)

(IDENT organism-2 klebsiella 0.25).

Mycin combines the CFs of the rules, using a rather informal function, to produce a final estimate of the certainty of its conclusions. For example, to evaluate Mycin's rules, the following steps are taken: [BUC 84]

(1) The CF of a conjunction of several facts is taken to be the minimum of the CFs of the individual facts.

(2) The CF of the conclusion produced by a rule is the CF of its premise multiplied by the CF of the rule.

(3) The CF for a fact produced as the conclusion of one or more rules is the maximum of the CFs produced by the rules yielding that conclusion.

Suppose Mycin is trying to establish fact Z. Suppose further that the only rules concluding anything about Z are:

IF A and B and C, THEN D (CF = 0.6)

IF H and I and J, THEN D (CF = 0.8).

Suppose also that facts A, B, C, H, I, and J have CFs of 0.5, 0.7, 0.4, 0.9, 0.6, and 0.8, respectively. Then the following computation produces a CF of 0.48 for D.

IF A and B and C, THEN D (CF = 0.6)

CF(A) = 0.5 | |
CF(B) = 0.7 | --> MIN = 0.4 | --> 0.4 x 0.6 = 0.24
CF(C) = 0.4 | |

IF H and I and J, THEN D (CF = 0.8)

CF(H) = 0.9 | |
CF(I) = 0.6 | --> MIN = 0.6 | --> 0.6 x 0.8 = 0.48
CF(J) = 0.8 | |

--> MAX = 0.48

One of the fundamental concerns of a numerical representation of certainty is that all evidence is processed in exactly the same way. While numbers are easy to propagate over inferences, what the numbers mean may be unclear. P.R. Cohen and M.R. Grinberg have proposed an alternative to numerical methods for reasoning about uncertainty in which different kinds of evidence are treated differently. Their theory of endorsements "represents factors that affect *certainty* and supports multiple strategies for dealing with uncertainty" [COH 83].

Endorsements are records for believing or disbelieving a hypothesis. For example, consider the process *applying for a mortage*. The applicant must satisfy certain requirements before he/she is endorsed at any given level. Rather than the loan officer, think of a set of rules being used to monitor the development of a line of reasoning. Each rule endorses a step in the argument if it satifies certain requirements. Whenever a rule is used, its conclusion accrues one or more endorsements. These endorsements are the records that a particular kind of inference has taken place. The certainty of a hypothesis can be represented by its endorsements and those of its predecessors. In other words, in terms of the mortage application, one's confidence in the applicant can be determined by the level of endorsement he/she has reached.

Implicit in the theory of endorsements is the idea of ranking. One hypothesis may be endorsed over another based on the confidence in one hypothesis over the other. For example, eyewitness testimony is usually preferable to circumstantial evidence. Thus a conclusion endorsed as having eyewitness testimony for its support is more certain than a conclusion endorsed as supported by circumstantial evidence.

Much heuristic knowledge is needed to rank endorsements. Each domain of expertise needs a set of rules to propagate endorsements over inferences. These serve to make a rule sensitive to the context in which it is used. For further study of endorsements see [COH 83].

Smith is working.

One works in the office.

When in the office, one is at one's desk.

John is in the office.

When at the office, one works.

One does not work when one is sick.

Figure 4.3.1. Statements for Resolution Example

C1 (doing (smith, work))

C2 (NOTdoing (X, work) OR in (X, office))

C3 (NOTin (X, office) OR at (X, desk))

C4 (in (office, john))

C5 (NOTin (X, office) OR doing (X, work))

C6 (NOT doing (X, work) OR NOTsick(x))

Figure 4.3.2. Corresponding Clauses in Database

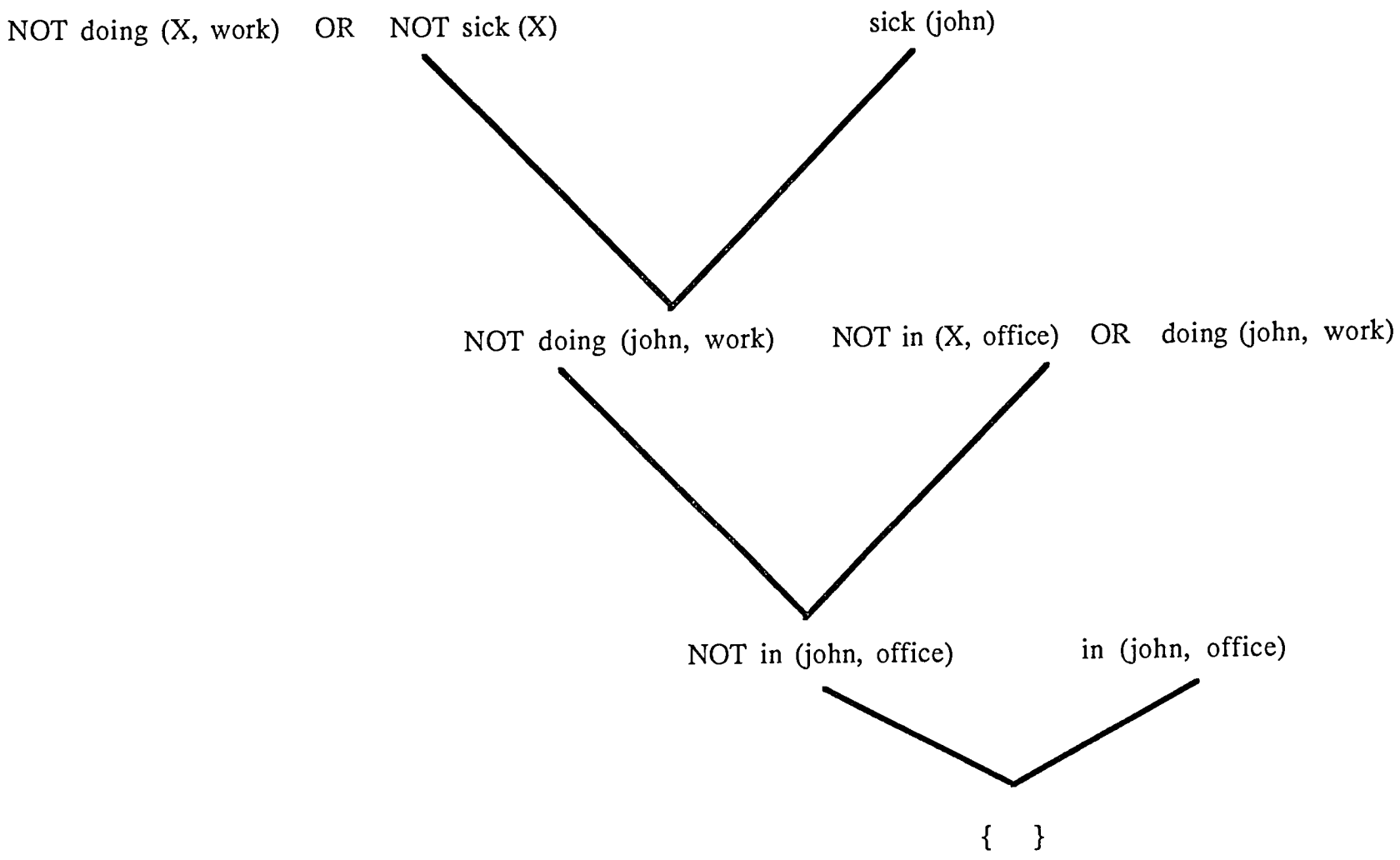


Figure 4.3.3. Sequence of Resolvents

1. It is cold.
   cold

2. The movie was great.
   moviegreat

3. Boole was a mathematician.
   boolemathematician

4. The sky is blue and the grass is green.
   skyblue AND grassgreen

5. If today is Monday, then tomorrow is Tuesday.
   todaymonday IMPLIES tommorrowtuesday

Figure 4.4.1. Simple Facts in Propositional Logic

	Tall	Not Middle-Sized
5'0"	0.00	1.00
5'4"	0.08	0.92
5'8"	0.32	0.68
6'0"	0.50	0.50
6'4"	0.82	0.82
6'8"	0.98	0.98
7'0"	1.00	1.00

Figure 4.8.3. Example of Fuzzy Sets

Chapter 5.

Production Rules and Knowledge Representation

5.1 What is a Production System?

Production systems, one of the most widely used procedural models of knowledge representation, were originally proposed by E. Post in 1943 as a general computational mechanism, "equivalent in power to a Turing machine" [VAN 81]. A production system consists of three parts:

(1) a set of production rules or productions

(2) the data base

(3) an interpreter

The following three sections will describe each of these components.

5.2 Production Rules

A production rule is a two-part rule of the form:

IF condition THEN action.

The left-hand side of the rule, the "IF" clause, is called the antecedent or situation and states the conditions that must be satisfied for the production to be applicable; it represents some pattern to be matched. The antecedent typically contains several clauses linked by the logical connectives AND, OR, and NOT. The right-hand side of the rule, the "THEN" clause, is called the consequent and specifies an action to be taken. The consequent consists of one or more action phrases. For example, in the statement "If it rains, then I will take my umbrella", "it rains" is the antecedent and "I will take my umbrella" is the consequent. When the set of conditions is satisfied, that is, the data matches the pat-

tern, the action part of the rule is executed by the interpreter. The action part of the rule is generally a list of modifications to be made to the data base and can include adding, deleting, or modifying elements. These actions can also include reading or writing data from an input/output device. In the above example, the rule causes the system to add the assertion "I will take my umbrella" to the data base, if the condition, "it rains", is satisfied.

5.3 The Data Base

Data or working memory, the dynamic global data base of a production system, contains representations of facts and assertions about the particular problem. The elements of data memory can be represented as a set of tuples, as a set of property lists, or as a collection of recursively defined records [DAV 77]. For example, the fact "Marie is a patient who has a runny nose" could be stored using the property list structure of Lisp. This structure has associated with each object a list of all those properties of the object relevant to it. Generally, the format is a triple: (object attribute value), but this is dependent upon the implementation language. This fact could appear in data memory using Lisp as:

```
(marie isa patient)
(marie has runny-nose)
```

or using Prolog as:

```
isa(marie,patient)
has(marie,runnynose)
```

The contents may be partially or totally ordered on the basis of their time of creation or most recent modification.

Rules are stored in an area of memory called production or rule memory. Each production rule is relatively independent of every other production rule. There is no direct communication between the rules, and data cannot be explicitly transferred from one rule to another. Thus the knowledge base is inherently modular, making it relatively easy to update. Individual rules can be added, deleted, or

modified without drastically affecting the overall performance of the system. However, productions can indirectly communicate with one another through data memory. Thus a production system knowledge base can be viewed as a bulletin board that provides the only means of communication between rules.

In order for a rule to fire (to have its action part executed), the antecedent must be present in data memory. For example, in the rule

If the patient has a runny nose and a fever,
then prescribe a cold medicine

it is necessary for the facts "the patient has a runny nose" and "the patient has a fever" to be present in data memory if the action "prescribe a cold medicine" is to be executed. Only those rules that match the data in memory are executable. Some systems allow only exact matches of data to the conditions, others allow complex pattern matching using variables. During a match involving a variable, the variable portion of the condition is bound or instantiated (replaced by a constant from data memory) and this binding is used consistently throughout the rule in both the condition and action parts. The following section describes the matching process further.

5.4 The Inference Method

The inference engine in a production system consists of a three step cycle, called the recognize-act cycle. The three states are: match, select, and execute. In the match state the system matches the antecedent, the consequent, or both with the contents of data memory using some sort of pattern matching algorithm, i.e., unification. All the matchings are collected into a set called the "conflict set". Next, the system applies some selection algorithm to choose which rule from the conflict set will be executed. Usually the rule with the largest number of matched conditions is chosen. In the case of a tie, a "conflict resolution" procedure is used to pick one. There are several different approaches to conflict resolution, including choosing the first rule that matches (i.e., first in terms of linear order of

data memory), choosing the rule with the highest priority as defined by the system, or choosing the rule that matches the rule most recently added to data memory. (See [MCD 78] for further discussion of conflict resolution). Finally, the system executes the selected rule and modifies data memory accordingly. This cycle is repeated until no more matches are found. For example, let the following facts be contained in data memory:

Marie is a patient
who has a runny nose
who has a fever.

Also let these three rules be contained in production memory:

(1) **If the patient has a runny nose**
then prescribe an antihistamine

(2) **If the patient has a runny nose and**
the patient has a fever
then prescribe aspirin

(3) **If the patient has a runny nose and**
the patient has a fever and the patient has a cough
then prescribe a cough suppresent with aspirin.

The match state instantiates the variable patient to Marie and finds a match with rule (1) (one condition matches) and with Rule 2 (two conditions match). Both of these matches are put into the conflict set. The select state selects rule (2) as the best match since it matches two conditions. The execute state adds "prescribe aspirin" to data memory for patient Marie. Data memory now contains the facts:

Marie is a patient
who has a runny nose
who has a fever
who is prescribed aspirin.

Several early production systems incorporated techniques based on the resolution principle, but problems with an exploding search space and the canonical forms (much of the heuristic information in a statement can be lost when translating it to a standard canonical form) have led to the use of the

recognize-act cycle.

The control strategy in a production system can be forward chaining, backward chaining or a combination of both. The condition part of a production rule is the "given information" and the action part is the "goal". In forward chaining the left-hand side (condition) of the rule is used for matching. The condition specifies the facts that will be matched against data memory. The inference engine performs the matching and executes the right-hand side of the rule. The sequence is from the given information to the goal. In backward chaining the right-hand side (action part) of the rule is used for matching. The system starts with the initial goal (hypothesis) to be achieved and works backwards finding rules to achieve the goal. The initial goal to be achieved is broken down into simpler subgoals until either a solution is found or all goals have been broken down into their simplest components. The final solution is obtained by combining the solutions to all the subgoals. The chaining is successful if a solution is found; the hypothesis is verified. The chaining is not successful if some condition cannot be matched.

5.5 Uncertainty and Incomplete Knowledge

Production rule systems have been used as a way of representing probablistic information, of operating in a domain of uncertain knowledge. This information is represented as a set of production rules with some probability associated with each rule. The rules are chained together to form deductions, combining the probabilities to compute the overall probability that the conclusion is true. As mentioned in section 4.8.4, Mycin, a production rule based system, augments its rules with CFs which represent the overall estimate of the confidence of the hypothesis. Prospector, an expert system that aids geologists in evaluating mineral sites for potential deposits, uses another approach - Bayes' Rule [DUD 76], which is another method for mathematically calculating the probability of a particular event.

Solutions to the incomplete knowledge problem fall into a continuum where one end uses maximal knowledge but requires large amounts of resources to infer or acquire the missing information,

while the other end uses little knowledge and minimal resources. In particular, the backward chaining of Mycin does a complete search of the rule space looking for rules which may provide the missing information required to evaluate the antecedent [BUC 84]. Another approach reduces the search cost of Mycin's backward chaining by utilizing rule abstractions to plan the backward chaining of rules [KLA 78]. Abstraction reduces the information available. In both these approaches, the rules determine what information is relevant and the method of acquisition. A third approach decreases the cost of choosing a rule even further by not searching for the missing information. Instead, a partial match metric based on specificity and certainty is derived and the highest valued rule is chosen [JOS 78].

5.6 Implementation

Prolog can be viewed as a backward chaining ruled based language. All inferencing is a result of queries to the data base of facts and IF ... THEN rules with a single consequent. Conflict resolution is built into the language. This is the basis of a production rule based system. Therefore, when I attempted to use production rules as the representation method and Prolog as the implementation language, the system was no different from the one I had built using FOL.

Other languages have been developed for production rule based implementations, among them is OPS5. OPS5, unlike Prolog, has the capability to handle multiple consequents of a rule and a built-in facility for applying certainty factors to rules [BRO 85]. In OPS5 a production rule and a fact (working memory element) have a syntax very similar to Prolog's. A production rule has the following syntax:

```
(p rule-name
    condition element
    [ [-](condition element)]*
    -->
    action
    [action]*
).
```

A working memory element (WME) is a datum which can serve as a condition element or the object of an action in a production. A typical WME, in user representation, might be:

(occupation medical doctor)

Appendix C illustrates how the knowledge base derived from appendix A might appear if OPS5 were used. The rules are expressed very similarily to the Prolog rules we see in appendix B. The WMEs are expressed as vectors (simple arrays) rather than as "predicate(argument1, argument2, ..., argumentn)".

5.7 Conclusions

Some of the major advantages of a production system as a knowledge representation method are:

(1) Manageability: It is particularily easy to modify, add or delete productions, or to change the order of the productions in memory. The production rules cannot communicate with one another, thus changing a rule will have no direct effect on the other rules.

(2) Understanding: Knowledge in the form of production rules is very readable so that even someone unfamiliar with a program can understand it.

(3) Many types of applications seem to fit naturally into a production rule format, for example, medical diagnosis, computer system configuration, mineral exploration, and human behavior modeling. Statements like "what action to take" if a particular "situation" occurs are easily encoded as production rules.

(4) Explanation: The format of the facts and rules is restrictive, thus allowing them to be stored in a similar form. This constrained format also allows the system to easily justify its conclusion -- Why was a certain fact used? How was a fact established? The system would chain backward from the goal state finding evidence (facts) to support the conclusion.

There are some major disadvantages also:

(1) Since the rules do not directly interact with one another, only via data memory, there may be side effects from unexpected interactions.

(2) Some domains may not fit easily into the production rule format. For example, many objects in the real world have several properties and relationships may exist among objects. These properties and relationships may need to be collected together to form a single description of an object. The system can then concentrate its efforts on this collection of items rather than having to consider all the facts it knows.

(3) Conflict resolution can pose a problem in that the effect on the data base is different depending upon which rule is executed first. Some systems resolve this problem by adding a routine to monitor flags and switches in the data base as updating occurs [MCD 78].

(4) Every action is performed by means of the match-select-execute cycle. This can cause great overhead in terms of time, especially as the data base becomes larger and more complex.

A production system is basically an extension of a formal logic based system that has the capability to represent incomplete and uncertain information. Unlike strict FOL representations, it is based on nonmonotonic reasoning. The inference mechanisms in both systems are based on deduction FOL based systems use resolution and production systems use the recognize-act cycle.

Many expert systems have been implemented as production systems. Diagnostic systems are the most common. They are used to diagnose diseases, defects or other problems in medicine, science and engineering. Selection/configuration systems that select and specify features of the components required to create some item have also been built. Production systems have also been used to build schedule management and production scheduling systems.

Chapter 6.

Semantic Networks and Knowledge Representation

6.1 A Brief Background

Semantic networks were first introduced in the mid 1960's by M.R. Quillian as a representation for the concepts underlying English words, and since then have become an increasingly popular scheme in natural language understanding and manipulation of taxonomic hierarchies. There are many different forms of the semantic network, thus making it rather difficult to discuss them as one class of representation techniques. However, there is a commonality among them: they all are composed of nodes and links and virtually all semantic nets have been used to represent relationships between concepts. A network comprised of nodes and links is not necessarily a "semantic" network. There must be meaning embodied in the symbols. According to [FIN 79], "the semanticness of semantic nets lies in their being used to represent the semantics of English words".

The distinctive characteristic of semantic nets as stated by W. Shapiro is that "all the information about a given conceptual entity should be reachable from a common place" [ISR 84]. Semantic net theorists believe that in a semantic network knowledge about a given entity is "directly attached" to the node for that entity. The fundamental reason for using semantic nets is that "the knowledge to perform an intellectual task lies in the semantic vicinity of the concepts involved in the task" [ISR 84].

On the other hand, P.J. Hayes states that semantic networks "are equivalent to a set of assertions in first-order-logic and that the only value they have is as syntatic sugar for those assertions" [HAY 77].

Do semantic nets have any real value? Can they be reduced to FOL expressions and maintain the intended meaning and relationships? How do they compare with production systems? These are important issues that I will address in the following sections.

6.2 The Structure of a Semantic Network

The information in a semantic net is represented as a set of nodes connected by a set of labeled arcs (links). There exist many different interpretations of nodes and links and no constraints have been set as to how to name them. However, a few conventions do exist. The nodes represent instances of objects, classes (sets) of objects, situations, actions, and other things. It is important to differentiate between a node representing a class of objects (i.e. flower) and a node representing an instance of a class (i.e. tulip) (See section 6.4). The links represent relations, usually binary, between the nodes. Common links include ISA links that are often used to represent the class/instance relationship. In figure 6.2.1, Watson is an instance of the class male. A male, in turn, is an instance of the class person. There are many different meanings attached to the ISA link. P.J. Hayes suggests that the conditional IF...THEN is the idea represented by ISA. Others tend to feel that the most prevalent use of ISA is as a default. For example, "Watson ISA male" is usually taken to be true until it is explicitly retracted or cancelled. HAS-A or PART-OF links show part/subpart relationships; for example, "A car HAS-A windshield". Other links capture heuristic knowledge, such as, "Success REQUIRES work". All in all, a link can be named whatever one deems necessary as long as it adequately describes the intended relationship.

6.3 The Knowledge Base

The knowledge base in a semantic net is organized along certain axes. Some of the most widely used organizational axes are: [FIN 79]

(1) Classification: An object should be associated with its generic type(s), i.e. watson should be associated with student, male, and person.

(2) Aggregation: An object is related to its parts or components, i.e., tulip is related to its parts: stem, leaf, and petal.

(3) Generalization: Relates a type to more generic ones - organizes types into a generalization or an ISA hierarchy, i.e., robin is a type of a bird.

(4) Partitions: Objects and elements of relations are grouped into partitions that are organized hierarchically, so that if partition S is below partition R, everything visible or present in R is also visible in S unless otherwise specified. Some major difficulties encountered with the use of semantic nets are representing logical connectives, time, and quantification. One way to solve these problems is to partition the network into a hierarchical set of partitions each of which corresponds to the scope of one or more variables. See [HEN 75] for further discussion.

Not all semantic networks use all of these principles or address them in the same way. Classification and generalization are in many cases viewed as the same.

Although the very nature of a semantic net lends itself to a graphic representation, the information is not stored that way. The knowledge base contains collections of objects and relations defined over them usually represented using an attribute-value structure. The network shown in figure 6.2.1 might be represented as lists having the form (object attribute value) as follows:

```
(male ISA person)
(watson ISA male
        ISA student)
(med-book ISA book)
(owns OWNER watson
      OWNEE med-book)
```

Each object is stored with its attribute(s) (link) and its corresponding value. In Lisp each node would be an atom (object), links would be properties, and nodes at the other end of the links would be values.

Modifications to the knowledge base occur through the insertion and deletion of objects and the manipulation of relations. As the knowledge base grows the semantic network can become extremely complicated - like a maze and the search routines could easily get caught in the maze. Thus it is important to use some type(s) of organizational mechanism(s) as described above to get to the information as efficiently as possible. Although hierarchical organizations have traditionally been viewed as computationally undesireable, the organizational principles employed provide the means for a reasonablly efficient organization and management of a large data base.

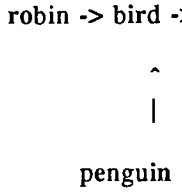
6.4 The Inference Method

There are no formal semantics in a semantic network representation. "Meaning is assigned to the structure only by means of the procedures that manipluate the network." [BAR 81]. Many different techniques have been designed to make inferences in the various types of semantic network based systems. The most popular kind of inference involves the inheritance of information from the top levels of the hierarchy downward along the links. A semantic net is a particularly useful mechanism for representing taxonomic relations among objects and it illustrates beautifully the notion of inheritance. A taxonomy is constructed first, in terms of classes, the most general concepts, then proceeds to deal with more specified subclasses. Instances of a subclass are generally also instances of its superclass. In the following simple net, robin, a subclass of bird, inherits all the attributes of its superclass, bird. Therefore, we can deduce that a robin flies by simply tracing up the hierarchy. Formally, we are using a search procedure (i.e., breadth-first) that looks for appropriate properties of the instances.

robin -> bird -> flies

When an object is an instance of more than one class, it will inherit attributes from all of them - inheritance is cumulative.

There are some serious consequences of property inheritance. Suppose we were to add the fact "a penguin is a bird" to our network,



two problems are evident here:

(1) If penguin is a bird and bird has some properties, then penguin inherits those properties from bird. A penguin is indeed a bird, but penguins do not fly!

(2) Since penguin inherits the property flies and asserting that every bird flies, then can we correctly assert penguins cannot be birds since they do not fly?

Inheriting properties from a more general class is done by default and may lead to invalid assertions. We need some type of procedure to override inheritance to deal with exceptions. Many existing semantic network based expert systems either do not handle exceptions or handle them using a system of local masking in which more local information masks and supercedes any conflicting information found at more abstract levels of the hierarchy. For example, the local information, "penguin does not fly" would be added to the network to mask "bird flies". There are some problems with a masking system:

(1) How can we determine whether the local information conflicts with the inherited information or whether it is additional information?

(2) If a node has multiple upward branches, how can we determine which of the information inherited is the more local?

(3) The masking system works only for properties, it does not extend naturally to the situation where inherited membership in a class is to be cancelled.

These problems can be greatly reduced if the information to be superseded is explicitly cancelled. NETL, a language developed for semantic networks, handles exceptions as follows [FAH 79]. Suppose the following facts are contained in the data base:

(1) **a mollusc is a shell bearer**

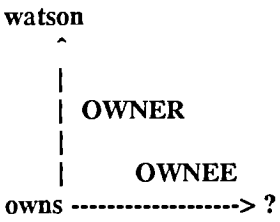
(2) a **cephalopod** is a **mollusc, but is not a shell bearer**

(3) a **nautilus** is a **cephalopod, and is a shell bearer**

And we now want to create the subclass naked nautilus which is not a shell bearer. All that is necessary is to add another node (naked nautilus) and cancel link. (See figure 6.4.1.) A cancel (or uncancel) link is a relation used to specify inheritance semantics (i.e., information passing characteristics)

indicating which nodes (values) can be inherited over that relation. In figure 6.4.1 the cancel link from "naked nautilus" to "shell bearer" indicates that "naked nautilus" will inherit all of the properties of "mollusc", "cephalpod", and "nautilus" except those inherited from "shell bearer". Relations can be fully defined in terms of inheritance semantics, transitivity, and scope. This makes it possible to use inheritance to make inferences that would otherwise require the use of rules. During traversal of the network every relevant cancel link is checked. This involves a great deal of overhead especially if the network is modified frequently.

Inheritance is based on matching network structures. The idea is to construct a network fragment representing the question to be answered, then match this fragment against the data base. The matching process binds any variables to the nodes that create a match. For example, referring to figure 6.2.1, to answer the question "What does Watson own?", we would first create the network fragment

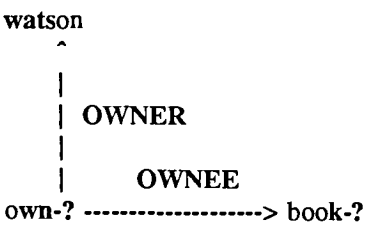


representing the query. This fragment would be represented as

```
(owns OWNER watson
     OWNEE ?)
```

where "?" is the variable to be instantiated. The network fragment is then matched against the data base (in figure 6.3.1) searching for an "owns" node that has an "OWNER " link to Watson. When a match is found, the node that the "OWNEE" link in the fragment points to is bound to the value "med-book", which is the answer to the question.

Suppose now that we wanted to answer the question "Does Watson own a book?". Creating the network fragment,

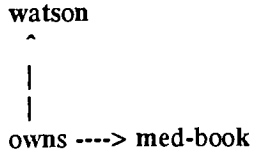


the nodes "watson", "own-?", and "book-?" represent the relation we are looking for. This fragment would be represented as

(own-? OWNER watson
OWNEE book-?)

Searching the data base in figure 6.3.1 we do not find this relation. The matching procedure has to construct an ISA link from own to book to make the match possible. In other words, the data base is searched for a match for "book-?". A match is found in the triple "(med-book ISA book)". Thus the "book-?" node is bound to "med-book" and "own-?" is bound to "owns". The answer to our question is "yes, a med-book". Here we see how an inference is made when the required network structure is not explicitly stored in the data base.

Another method of reasoning, deveopled by R. Quillian, is called "spreading activation" [WAT 78]. This method uses procedures to make inferences about the relationships among objects. Starting with two nodes, all the nodes connected to them are activated. All the nodes connected to each of these nodes are in turn activated. When some object is simultaneously activated from two directions, a connection is found. Using this procedure let us see how it is possible to answer the question "What is the relationship between Watson and medical book?". (See figure below).



First the data base is searched for the two nodes "watson" and "med-book" then for all the nodes connected to these two. The object "owns" is found. Since both nodes, "watson" and "med-book", are connected to the same object, "owns", we can conclude that a relation exists between "watson" and

"med-book" and that relation is "owns".

One can see that a semantic network can simplify the process of deduction. For example, in figure 6.2.1, it is readily apparent that Watson is a person and owns a book, even though there is no direct link connecting the WATSON node to the PERSON and BOOK nodes. In English, you might reason like this: "A male is a person; Watson is a male; therefore, Watson is a person". In a semantic net, discovering the relationships is considerably simpler: all you have to do is follow the arrows.

6.5 Uncertainty and Incomplete Knowledge

When specific information about an object is not available a default value is placed in the object's node and the link(s) to that node is (are) appropriately flagged. If and when definite information is provided, the flag would be removed (changed) and the value in the node would be modified. The network in figure 6.5.1 indicates that we are not sure what the "first-name-is" for a person whose first initial is J, but to the best of our knowledge it is John.

Uncertainty can be handled in a semantic network in much the same way as it is handled in a production system. Numbers (CFs), indicating the degree of confidence in an object, could be associated with each object. The system could determine the overall confidence of a situation by searching the network and combining certainty factors using the methods described in section 4.8.4. In figure 6.5.2 we could determine which would be the best cold medicine to prescribe, in terms of least chance of side effects. "COLD-MED-A" has a CF of 0.4 indicating that there is a 4% chance of it producing side effects while "COLD-MED-B" has a 7% chance of producing side effects. Thus "COLD-MED-A" would be prescribed.

6.6 Demons

Sometimes certain information must be provided upon request. This is done through a system of "demons" or "if-needed" procedures, which are similar to productions [WIN 84]. The demons reside in the data base and are used upon request. For example, in the network in figure 6.6.1, a request for Mary's best-weight would activate the best-weight procedure. This procedure would search the network for her age and height then compute her best-weight.

6.7 Implementation

Using Prolog as the implementation language and semantic nets as the representation technique, I once again realized that the resulting system would be the same as when using FOL or production rules. A semantic network can be expressed as a FOL expression (if the information is certain and complete). The nodes and links play roughly the same role as do arguments and predicates, respectively, in FOL. The semantic network in figure 6.2.1 translated into FOL or Prolog might look like this:

isa(male,person)
isa(watson,male)
isa(watson,student)
isa(med-book,book)
ownee(owns,med-book)
owner(owns,watson)

and a set of corresponding rules could be written as:

(1) ISA(X,person) -> ISA(X,Y) and ISA(Y,person)
(2) ownee(X,book) -> ownee(X,Y) and ISA(Y,book).

Again we can see that we are constrained by the format of Prolog. In using Prolog we lose the concept of an inheritance hierarchy since the inference mechanism is resolution. Thus data is redundant - some knowledge must be duplicated. Since the relations in a semantic network can be fully defined in terms of inheritance semantics, it is possible to use inheritance to make inferences that would otherwise require the use of rules. For example, in the semantic net (figure 6.2.1), it is evident that Watson is a person through the inheritance mechanism, but in Prolog we would have to write rule (1).

Computer languages, such as NETL, have been developed to capture the value and flavor of semantic nets. Appendix D illustrates how the knowledge base of Appendix B might appear using a semantic network based language.

6.8 Conclusions

One of the advantages of a semantic network is the ability to represent taxonomic information in terms of elements of subclasses and superclasses. Related objects are clustered so that facts about an object can be deduced from the nodes to which they are directly linked, without having to search an entire data base, because the relation between objects defines access paths for traversing the network. Other features are the availability of the organizational principles for the data base, their graphical notation- which enhances their understandability, the ability to incorporate default values and handle exceptions.

In a domain, that includes a great deal of knowledge with complex interrelations, a semantic network can provide the foundation for a sophisticated inference engine. Semantic nets have been used for knowledge representation in expert systems, such as SRI's Prospector [DUD 78], which has proven to be successful in predicting the location of mineral ores.

The major disadvantages are the lack of a semantic convention and a standard terminology. Also, as the data base becomes increasingly larger, it is more difficult to manage, especially if there are many exceptions. Incremental builds of the data base could create much overhead since you cannot simply add on the information. New data must be tied into the correct place in the hierarchy so that integrity is maintained and existing relationships are not disrupted.

Much research has been done on the significance of inheritance in semantic networks, although it does not appear to make a difference in the expressive power of the system that uses it. At best, it saves storage space since:

(1) objects are clustered with their properties

(2) inheritance, rather than rules, is used to make inferences.

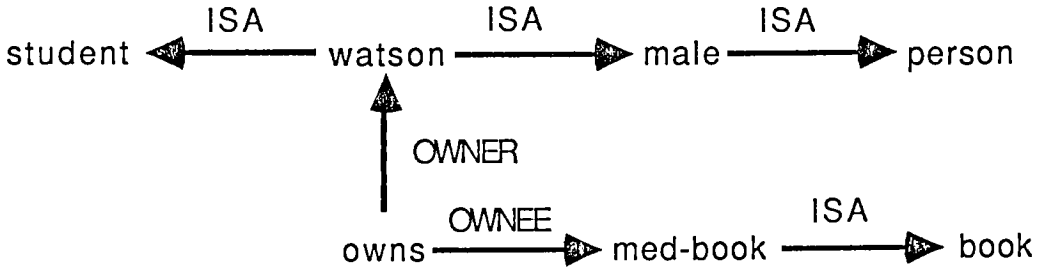


Figure 6.2.1. A Semantic Network

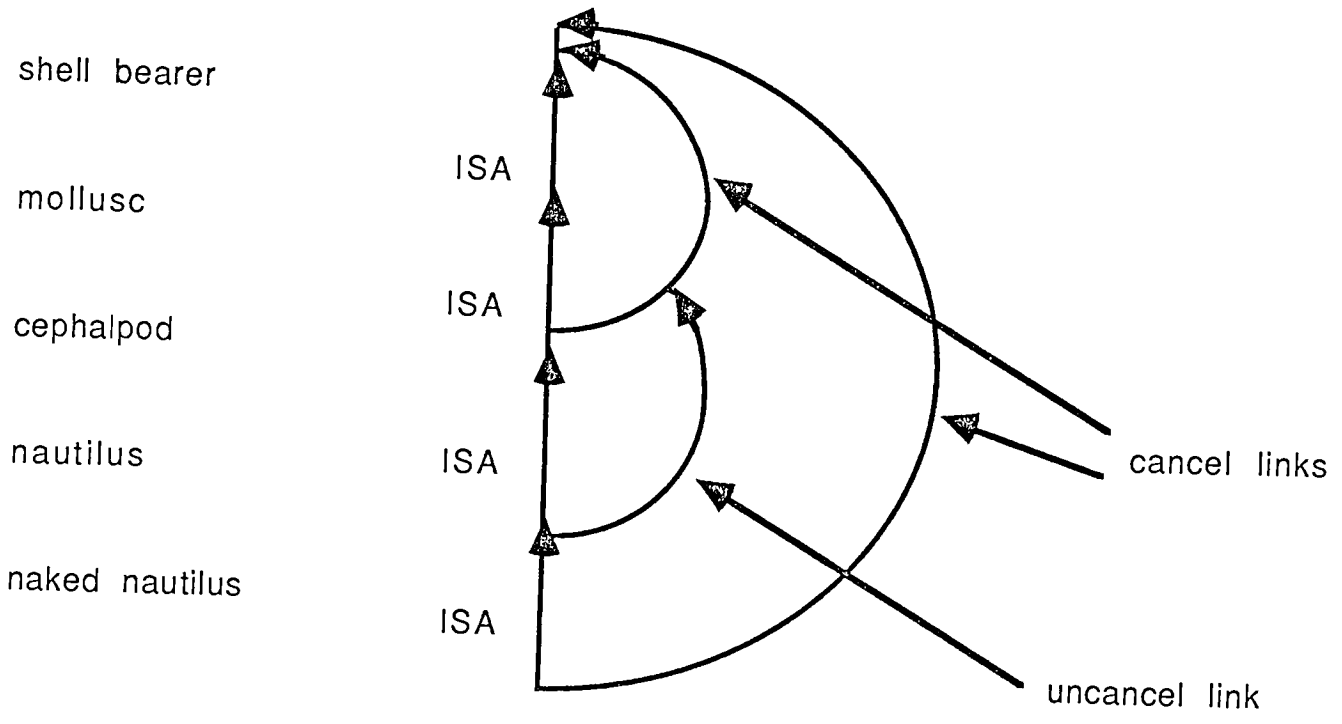


Figure 6.4.1. Exception Handling in NETL

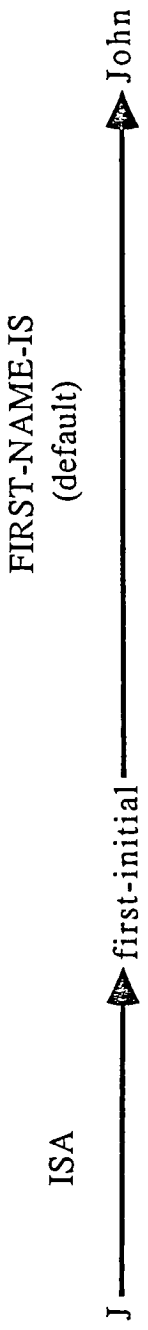


Figure 6.5.1. Dealing with Incomplete Knowledge in a Semantic Net

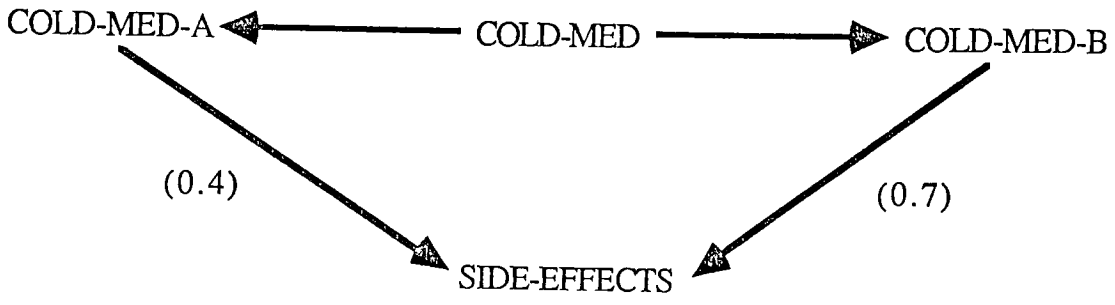


Figure 6.5.2. Dealing with Uncertainty in a Semantic Network

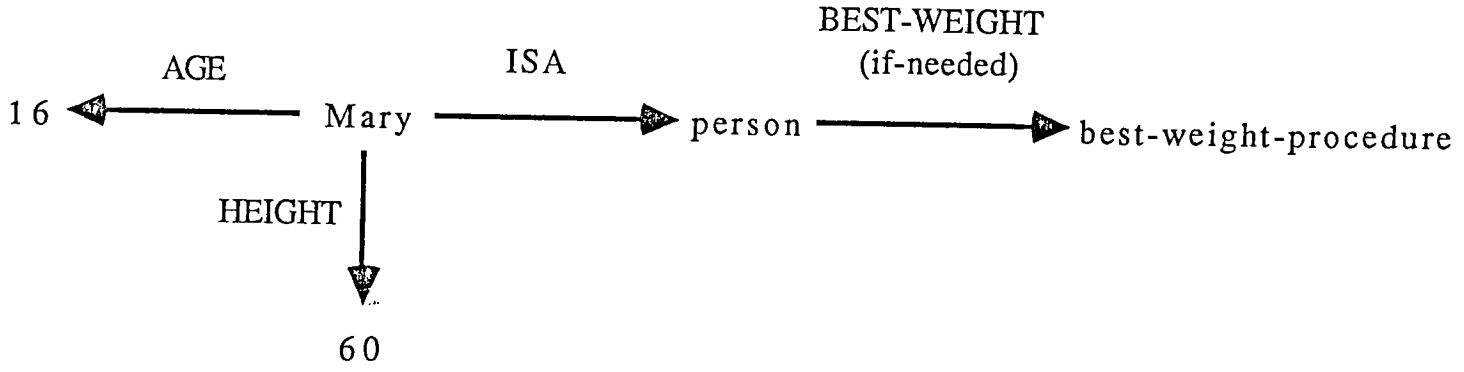


Figure 6.6.1. Example of a demon in a Semantic Network

Chapter 7.

Frames and Knowledge Representation

7.1 The Structure of a Frame

Marvin Minsky postulated that a useful way to organize a knowledge base was to break it into highly modular, " almost decomposable" chunks called frames [MIN 75]. Frames, also referred to as schemata, (schema is the singular form) have become another major method of representing knowledge. They are particularily useful when used to represent knowledge of certain stereotypical concepts, such as a tennis match, the living room in a house, a class of ships, or a trip to the movies. Frame based systems have been developed for use in natural language understanding, computer vision, military and scientific applications.

Frames can be viewed as prototypes that represent concepts by certain standard properties and relations to other concepts. For example, a robin frame might represent robins as belonging to the class BIRDS and having such attributes as PARTS, COLOR, HABITAT, etc. A frame based system is essentially a semantic network in which concepts are represented by frames instead of atomic symbols.

One of the basic ideas behind the development of frames is the general theory that people use a large collection of facts from previous experiences to analyze new situations. For example, when we go to a movie we might have particular expectations based on our previous experiences at the movies (if we've gone before) such as, price of ticket, location of ticket booth, where to buy popcorn, and where the best seats are. A frame provides the mechanism to organize around the object or situation being described. The main intent is the representation of things by a collection of frames. Many different variations have been developed, but most of them include the idea of having different types of frames for different types of objects or situations, with fields or slots in each frame to contain the information relevant to that type of frame as well as express relations between objects. For example, a frame for a book might have slots for the author, title, publication date, and number of pages. To describe a particular book, a copy of this book frame would be created, and the slots would be filled

with the information about the particular book being described. The generic frame "book" would appear as:

name-of-frame: book
type-of-frame: reading-material
author: _____
title: _____
publication-date: _____
number-of-pages: _____

and a particular instance of "book" appears as:

name-of-frame: historical-novel
type-of-frame: book
author: Robert Shea"
title: Shike
publication-date: 1981
number-of-pages: 453

Notice that the two frames have the same named slots. The "type-of-frame" slot establishes a property inheritance (relation) hierarchy among the frames which allows information (both slots and values) about the parent frame to be inherited by its children. Thus, the "historical-novel" frame inherits information from the "book" frame.

Slots may store values and may also contain default values, pointers to other frames, and sets of rules or procedures by which values may be obtained. Slots may also be blank. The default value in the frame in figure 7.1.1 asserts that, in lieu of contradictory or missing information, we assume that the tennis match is presently being played. When an instance of this frame is created, and if the "when-played" value is not known, the default value is placed in the slot and updated appropriately. The "name-of-players" and "result-of-match" slots contain procedures for determining the names of the players and the result of the tennis match, respectively. Procedures (demons) are sequences of instructions for computing the value of a slot. The instructions may combine information from other slots: the "result-of-match" slot requires information from the "when-played" slot and from the "tennis-results" frame. The contents of the "range" slot represent the possible states of the tennis match.

Frames are generally a way of storing declarative knowledge, but the inclusion of procedures creates a procedural-declarative representation.

7.2 The Knowledge Base

The knowledge base is a collection of frames organized in terms of some of the organizational axes discussed in section 6.3. The frames are arranged hierarchically , for example:

name-of-frame: **animals**
slot1: _____
slot2: _____
slot3: _____

name-of-frame: **bird**
type-of-frame: **animals**
slot1: _____
slot2: _____
slot3: _____

name-of-frame: **robin**
type-of-frame: **bird**
slot1: _____
slot2: _____
slot3 _____

Each slot of a frame can be filled by another frame structure and a given structure can fill more than one slot. Therefore, there is no redundant storage of information.

A generic frame represents a concept or an object while instances are represented by instantiations of the generic frame. For example, "animals" is a generic frame, whereas, "robin" is an instance of "animals". The "animals" frame contains three slots, some of which may contain values. When the "robin" frame is instantiated it inherits the slots and values from the "animals" and "bird" frames. Certain slots may be filled only at certain levels of the hierarchy; levels where the values are pertinent. The "animal" frame may have a slot called "diet" which is inherited through all subsequent levels of the hierarchy but is filled in at the "robin" level since different animals (and birds) have different diets.

Modifications to the data base are relativly easy: adding/deleting values to/from slots and creating/deleting slots. Creating a new frame would require a little more work since one would want to be sure to maintain the intended concepts and relations between the concepts.

One of the key features of a frame structure is this data organization. Retrieval of information (a

frame could be retrieved via some kind of indexing mechanism on its name) and inference processes are greatly facilitated since all the relevant information for each concept is collected together.

7.3 The Inference Method

One form of inference in a frame based system is based on "seeking confirmation of expectations", that is, by filling in the slots [MIN 75]. The first step is to select a frame representing the particular concept. One way to do this is to use some partial information, perhaps provided by the user. That frame will then be instantiated, to create a specific instance of the current concept, by filling in the slots. The slots can be filled in various ways: by direct inheritance from the corresponding slot in a parent frame, using the default value, or using the attached procedure. Since inheritance provides a means of reasoning by default, any values that are not specified in a frame may be inherited from another frame. In the "tennis-match" frame, the value of the "equipment" slot is inherited from the "court-played-sport" frame and the value of the "name-of-players" slot is determined by the "consult-table-X" procedure. Once all the slots in the appropriate frame(s) have been filled we can infer that an appropriate instance of the concept does indeed exist. If, on the other hand, all the slots could not be filled appropriately, a new frame must be selected. Information as to why an instantiation failed could possibly be used in selecting the next frame.

When inheriting slots, a search mechanism (i.e., breadth-first) is used to check all frames separated from the original frame by one realtion, then two, and so on. Slot inheritance continues until a slot is found. When inheriting values, the inherited slots are checked to determine if they contain values. If not, the search is continued for another occurrence of the slot. If so, the values are inherited. Whether the search continues depends on the inheritance specifications. (See section 7.4).

Another form of inference often used is called "matching" [HAY 81]. Suppose we have an instance of a concept and want to know whether it can be regarded as also being an instance of another concept. For example, referring to figure 7.3.1, can we view Candy Bar as a dogowner, where C.B. is an instance of the woman frame and dogowner is another frame? In other words, can we find an

instance of the dogowner frame that matches the C.B. frame?

Suppose woman has a slot called pet, then C.B. also has a slot called pet, and a condition necessary for C.B. to match dogowner would be that her pet slot is filled with an object that is a type of dog. Suppose also that dogowner has slots for dog and name. Then we could create an instance of dogowner, corresponding to C.B., by filling in the name slot with C.B.'s name and the dog slot with C.B.'s pet (that is, the type of pet!). What we have done is transferred the contents of the slots of one frame to the slots of another frame. This slot-to-slot transfer can be expressed using simple logical implications:

isdog(X) ^ pet(X,Y) ---> dogof(X,Y)

So given

(1) **name(C.B., Candy Bar) AND**

(2) **pet(collie, C.B.) AND**

(3) **isdog(collie)**

then it follows directly that

(4) **dogof(collie, C.B.)**

and therefore by (1) and (4) we conclude

dogowner(C.B.).

We have found an instance of the dogowner frame that does match the C.B. frame, since once all the slots have been filled we have an instance of that concept.

The idea of match here may pose some problems. Is this the same kind of match as in the unification procedure the exact match of two literals? Two concepts can have entirely different meanings depending upon their context. For example, "induction" could mean a type of mathematical reasoning in one situation and an initiation in another. Sometimes one frame cannot throughly describe a

situation so it is necessary to integrate several partially matched frames. See [WAT 78] for further discussion.

7.4 Exception Handling

As with semantic networks, inheritance can be controlled by using a set of inheritance specifications that specify inheritance paths through the frame hierarchy. Paths can be used to restrict the scope of inheritance, and thus eliminate unproductive search. The search path is defined in terms of which relations are to be stepped across and which are to be passed over when inheriting information. For example, slot3 of the "robin" frame in section 7.2, may be restricted from inheriting the value(s) in slot3 of the "bird" frame.

7.5 Implementation

As with semantic networks, attempting to represent the knowledge base derived from appendix A using frames as the knowledge representation technique and Prolog as the implementatiopn language, resulted in a set of Prolog facts and rules. Frames, slots and values were expressed as predicates, arguments and IF...THEN rules. The ability to organize the knowledge into an inheritance hierarchy and to control that inheritance was not present.

Appendix E illustrates how the knowledge base of appendix A might be represented using a frame based language such as CRL. The Prolog rule, (from Appendix B)

```
native_of(Y) :-
        coat_length(C),
        lapels(L),
        shoes(S),
        style(Y,C,L,S)
```

in attempting to be satisfied, calls three other rules that must all be satisfied and searches the set of

facts to match the arguments of the "style" predicate. The "city" frame indicates the possible values for coat_length, lapels and shoes, corresponding to the three rules called in the "native_of" rule. The instances of the city frame correspond to the three "style" Prolog facts.

7.6 Conclusions

It appears that the important attributes of a frame based system are the efficiencies in storage, and the inference mechanism. The information for a class of concepts can be concentrated in a single prototype. Therefore, the amount of storage is minimized and the set of concepts that must be searched to find a match is reduced. We have seen that frames are structures in which all knowledge about a particular object or event is stored together. Such a formalism cannot represent any more concepts than FOL, but the organization of the knowledge can be useful for modularity and accessability of the knowledge. In addition, frame systems allow ways to specify default values when that information is not explicitly given.

As mentioned earlier, a frame is essentially a semantic network. Therefore, the basic processes of inference, inheritance, and matching, are the same. In both schemes, related concepts are clustered. Both are general structures used to represent objects and events. However, frames allow for the representation of something from several points of view. For example, a frame system could be created to represent the different perspectives of a house. One frame could describe the front view, a second an aerial view, and another the back. Each view frame would be an instance of the general house frame with slots representing the particular view.

The idea of a frame is a fairly recent development. Researchers have taken different approaches in organization and interpretation of these systems. A number of experimental systems have been implemented, among them are GUS [GOL 77] and NUDGE [BOB 77]. GUS, a prototype of an automated airline reservation assistant, is an experiment in natural language understanding. NUDGE was designed as a management scheduling system, with the ability to provide for incomplete and inconsistent requests.

name-of-frame:	tennis-match
type-of-frame:	court-played-sport
number-of-players:	2 or 4
name-of-players:	
if-needed:	consult-table-X
equipment:	racket and ball
type-of-court:	grass or clay
when-played:	
range:	past-match, present-match, future-match
default:	present-match
result-of-match:	
if-needed:	find when-played then tennis-results frame

Figure 7.1.1. Tennis Match Frame

name-of-frame:	woman
type-of-frame:	person
name:	———
pet:	dog-frame, cat-frame, horse-frame
name-of-frame:	C.B.
type-of-frame:	woman
name:	Candy Bar
pet:	collie
name-of-frame:	dogowner
dog:	———
name:	———
name-of-frame:	dogowner-A
type-of-frame:	dogowner
dog:	collie
name:	Candy Bar

Figure 7.3.1. Example of the Matching Procedure

Chapter 8.

Summary

8.1 Conclusions

In the previous chapters of this thesis we examined four different knowledge representation techniques. An attempt was made to use each, in conjunction with Prolog, to build an expert system for the same given knowledge domain. Essentially, each implementation reduced to a collection of Prolog facts and rules. We are constrained by the format of Prolog - facts expressed as "predicate(argument1,argument2,...,argumentn)", IF ... THEN rules, and a built-in inference mechanism. Hence, we have found that FOL expressions, production rules, semantic networks, and frames will all be implemented in Prolog in the same way. The conclusion being, the implementation language often dictates which representation technique will be used.

Basically, I have shown that these four techniques are loosely based on logic. Some of them do provide for special things, such as: exception handling, defaults, organization into related groups and various inferencing mechanisms. Semantic nets and frames are not radically different from each other. They share the notion that an entity can be described as a collection of attributes and associated values; also, inference is through inheritance. Having discussed the advantages, disadvantages, and appropriate domains for each of the representation formalisms, the important critera in choosing a representation for a particular problem is to select one that allows all the necessary knowledge to be represented and that facilitates its use in solving the problem at hand.

Early expert systems tended to use one scheme or another exclusively. More recently the tendency seems to be to combine representations, with each scheme being used for the knowledge it represents best. For example, a system might use production rules to define procedures for discovering attributes of objects, semantic networks to define the relationships among the objects referenced in the rules, and frames to describe the objects' typical attributes.

8.2 Related Future Thesis Topics

Much research is ongoing in knowledge representation techniques. New methods for representation as well as knowledge representation languages are continuously being developed. Several computer languages have been developed to provide ways to manipulate different representation techniques. Examples are, OPS5, KRL, CRL, FRL, NETL, and KLONE. These languages capture the qualities and flavor of the particular representation technique while providing the inference mechanisms.

My research for this thesis has sparked an interest in knowledge engineering, the process of extracting knowledge from a domain expert and representing that knowledge in an expert system. Knowledge engineering lends itself to many research topics, some of which include: knowledge engineering methodologies, knowledge acquisition techniques and knowledge engineering using multiple experts. Chapter 4 mentioned the importance of dealing with default information and incomplete or uncertain data. Further research topics might include examining these other logics and developing an expert system that supports one of these ideas.

Appendix A

The scene from the movie Young Sherlock Holmes used as the domain for the knowledge representation examples.

The film opens as a coach roles through the busy streets of Victorian London. After a brief glimpse of urban British life during this period, the coach comes to a halt. A young boy disembarks, to be faced with an intimidating stone facade. He quickly weighs the pros and cons, and reasons for his ominous relocation.

Upon entering his new school, he is directly taken to the boys' dormitory. Young Watson, curious of his new surroundings, notices an interesting fellow grabling with a violin. As the frustration mounts, the violin player suddenly moves to smash his instrument against a bureau. Watson shouts "stop". The violinist momentarily taken aback, stares at the intruder. Watson follows up by stating that the instrument must be of value. The violinist, young Sherlock Holmes, replies, "I'll never learn to play this thing". Watson queries, "How long have you been studying the violin?" Holmes calmly replies, "Since yesterday".

Holmes quickly inspects his new quarry. Then he casually states, "Your name is John Watson, your father is a doctor, and you are from Oxford". The amazed Watson endeavors to discover the method used to correctly identify him. Holmes replies that he simply noted the obvious, and based on that he made his deductions. He goes on to explain that by observing the name tag on his suitcase, which reads J. Watson, also noting that the most common name with first initial J is John, he guessed the new student's name was John Watson. Next, because he was carrying medical books, he assumed that Watson's father was a doctor. Finally, he was able to ascertain where Watson was from, because the coat and shoes he wore were of a particular style unique to one specific English locality.

Appendix B

This is the Prolog implementation of the expert system described in chapter 4.

```
/* These are the rules */

name_is(F,L) :-                          /* determines the student's name */
        first_initial(I),
        first_name(F,I),
        last_name(L).

first_initial(X) :-                      /* queries user for first initial */
        write('What is his first initial?'), nl,
        read(X), nl.

son_of(X,Y) :-                           /* determines the occupation of */
        type_books(B),                           /* the student's father */
        occupation_is(B,Y).

type_books(B) :-                         /* queries user for type of books */
        write('Is he carrying medical, law, or art books?'), nl,
        read(B), nl.                                  /* student is carrying */

native_of(Y) :-                          /* determines student's native city */
        coat_length(C),  asserta(coat_size(C)),
        lapels(L), asserta(lapel_size(L)),        /* asserta adds the fact */
        shoes(S), asserta(shoe_style(S)),        /* to the data base */
        style(Y,C,L,S).

native_of(Y) :-
        write('Due to incomplete information I cannot determine native city.')
        area(Y), nl, nl.


/* The following rules query the user for information regarding the */
/* student's attire.  This information is used in determining the native */
/* city.

coat_length(long) :-                                  /* checks if this fact has */
        coat_size(long).                              /* been asserted */

coat_length(long) :-
        write('Is his coat long ? '), nl,
        read(Reply), nl,
        positive(Reply), !.
```

```
coat_length(mid_size) :-                    /* checks if this fact has */
        coat_size(mid_size).                /* been asserted */

coat_length(mid_size) :-
        write('Is his coat mid_size?'), nl,
        read(Reply), nl,
        positive(Reply), !.

lapels(wide) :-                             /* checks if this fact has */
        lapel_size(wide).                   /* been asserted */

lapels(wide) :-
        write('Are his lapels wide? '), nl,
        read(Reply), nl,
        positive(Reply), !.


lapels(narrow) :-                           /* checks if this fact has */
        lapel_size(narrow).                 /* been asserted */

lapels(narrow) :-
        write('Are his lapels narrow?'), nl,
        read(Reply), nl,
        positive(Reply), !.

shoes(wing_tip) :-                          /* checks if this fact has */
        shoe_style(wing_tip).               /* been asserted */

shoes(wing_tip) :-
        write('Are his shoes wing_tip? '), nl,
        read(Reply), nl,
        positive(Reply), !.

shoes(oxford) :-                            /* checks if this fact has */
        shoe_style(oxford).                 /* been asserted */

shoes(oxford) :-
        write('Are his shoes oxford?'), nl,
        read(Reply), nl,
        positive(Reply), !.


positive(yes).                              /* checks if user's response */
positive(y).                                /* is "yes" or "y" */
```

```
/* This rule drives the system - it fires the three rules that are  used
/* in the deduction process.

run :-
                nl, nl,
                write('Young Sherlock Holmes is about'),
                write(' to identify the new student'), nl,
                write('at Westbury Boys" Prep School.'),
                write(' He mentally asks himself the'), nl,
                write('following questions, observing very'),
                write(' closely the  boy"s attire. After a '), nl,
                write('couple of minutes he casually states'),
                write(' his conclusions.'), nl, nl,
                nl, nl,
        name_is(F,L),
        son_of(L,O),
        native_of(C),
        write('His name is '),
        write(F), write(' '), write(L), nl,
        write('His father is a '),
        write(O), nl,
        write('He is from '),
        write(C), nl.

/* These are the facts */

first_name(john,j).
first_name(james,j).
last_name(watson).

occupation_is(medical,doctor).
occupation_is(law,lawyer).
occupation_is(art,artist).

style(london,long,wide,wing_tip).
style(cambridge,long,narrow,oxford).
style(oxford,mid_size,narrow,wing_tip).


area(unknown).
```

```
/* Below is the output of a sample run of the program. */

Script started on Tue Feb 25 14:45:34 1986
*isis-jth2274[1]
*isis-jth2274[1] pœlog
C-Prolog version 1.4
| ?- run.

no
| ?- [h2].
h2 consulted 3156 bytes 0.85 sec.

yes
| ?- run.


Young Sherlock Holmes is about to identify the new student
at Westbury Boys" Prep School. He mentally asks himself the
following questions, observing very closely the  boy"s attire. After a
couple of minutes he casually states his conclusions



What is his first initial?
|: j.

Is he carrying medical, law, or art books?
|: art,

Is his coat long ?
|: yes.

Are his lapels wide?
|: no.

Are his lapels narrow?
|: yes.

Are his shoes wing_tip?
|: no.

Are his shoes oxford?
|: yes.

His name is john watson
His father is a artist
He is from cambridge

yes
| ?- run.
```

Young Sherlock Holmes is about to identify the new student at Westbury Boys" Prep School. He mentally asks himself the following questions, observing very closely the boy"s attire. After a couple of minutes he casually states his conclusions

What is his first initial?
|: j.

Is he carrying medical, law, or art books?
|: medical.

Is his coat long ?
|: no.

Is his coat mid_size?
|: yes.

Are his lapels wide?
|: no.

Are his shoes wing_tip?
|: yes.

His name is john watson
His father is a doctor
He is from oxford

yes
| ?- run.

Young Sherlock Holmes is about to identify the new student at Westbury Boys" Prep School. He mentally asks himself the following questions, observing very closely the boy"s attire. After a couple of minutes he casually states his conclusions

```
What is his first initial?
|: j.

Is he carrying medical, law, or art books?
|: law.

Is his coat long ?
|: no.

Are his lapels wide?
|: yes.

Are his shoes wing_tip?
|: no.

Are his shoes oxford?
|: no.

Is his coat mid_size?
|: no.

Due to incomplete information I cannot determine native city.

His name is john watson
His father is a lawyer
He is from unknown

yes
| ?- ^D
[ Prolog execution halted ]
*isis-jth2274[2] ^D
Use "exit" to leave csh.
*isis-jth2274[2] exit
*isis-jth2274[3]
script done on Tue Feb 25 14:48:48 1986
```

Appendix C

An OPS5 implementation of a portion of the knowledge base in appendix A.

```
(p name_is
      (first_initial I)
      (first_name F I)
      (last-name L)
      -->
      (call
         son_of))

(p son_of
      (type_books B)
      (occupation_is B Y)
      -->
      (call
         native_of))

(defun first_initial (I)
      (decalre (special X))
      (print "What is his first initial?")
      (setq I (read X)))

(defun type_books (B)
      (declare (special X))
      (print "Is he carrying medical, law or art books?")
      (setq B (read X)))

(first_name john j)
(first_name james j)
(last_name watson)
(occupation_is medical doctor)
(occupation_is law lawyer)
(occupation_is art artist)
```

Appendix D

A semantic network representation of a portion of the knowledge base in appendix A as it might appear in a system using a semantic network based language.

```
(name          (isa idform)
               (first-init-is j)
               (first-name-is john)
               (last-name-is watson))

(son-of        (isa idform)
               (fathers-work lawyer)
               (fathers-work doctor)
               (fathers-work artist))

(lawyer        (has law-books))

(doctor        (has med-books))

(artist        (has art-books))

(native-of     (isa idform)
               (city london)
               (city oxford)
               (city cambridge))

(london        (coat-length long)
               (lapels wide)
               (shoes wind-tip))

(oxford        (coat-length mid-sized)
               (lapels narrow)
               (shoes wing-tip))

(cambridge     (coat-length long)
               (lapels narrow)
               (shoes oxford))
```

Appendix E

A frame-based representation of a portion of the knowledge base in Appendix A as it might appear in a system using a frame based language. Frames 1, 2, 4, and 8 are generic frames and the others are instances of these.

1.
name-of-frame: student1
type-of-frame: student
identification: name frame
son-of: occupation frame
native-of: city frame

2.
name-of-frame: name
first-initial:
if-needed: procedure-A
first-name:
if-needed: procedure-B
last-name:
if-needed: procedure-C

3.
name-of-frame: name1
type-of-frame: name
first-initial: j
first-name: john
last-name: watson

4.
name-of-frame: occupation
worker: father or mother
job:
range: lawyer, doctor, artist
type-of-books:
range: law, medical, art

5.
name-of-frame: oc1
type-of-frame: occupation
worker: father
job: lawyer
type-of-books: law

6.
name-of-frame: oc2
type-of-frame: occupation
worker: father
job: doctor
type-of-books: medical

7.
name-of-frame: oc3
type-of-frame: occupation
worker: father
job: artist
type-of-books: art

8.
name-of-frame: city
coat-length:
    range: long, mid-sized, short
lapels:
    range: wide, narrow
shoes:
    range: oxford, wing-tip

9.
name-of-frame: london
type-of-frame: city
coat-length: long
lapels: wide
shoes: wing-tip

10.
name-of-frame: oxford
type-of-frame: city
coat-length: mid-sized
lapels: narrow
shoes: wing-tip

11.
name-of-frame: cambridge
type-of-frame: city
coat-length: long
lapels: narrow
shoes: oxford

Bibliography

1. [BAR 81] Barr, A. and Feingenbaum, E. A., The Handbook of Artifical Intelligence, Vol. 1. Los Altos, Ca.: William Kaufmann, 1981.

2. [BAR 82] Barr, A. and Feingenbaum, E. A., The Handbook of Artifical Intelligence, Vol. 2. Los Altos, Ca.: William Kaufmann, 1982.

3. [BOB 77] Bobrow, D. G., Kaplan, R. M., Kay, M., Norman, D.A., Thompson, H., and Winograd, T., "Gus, a Frame-Driven Dialog System", *Artifical Intelligence*, Vol. 8, 1977, pp. 155-173.

4. [BRO 85] Brownston, L., Farrell R., Kant, E. and Martin, N., Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming. Reading, Ma.: Addison-Wesley, 1985.

5. [BUC 84] Buchanan, B. G. and Shortcliffe, E. H., Rule-Based Expert Systems. Reading, Ma.: Addison-Wesley, 1984.

6. [CLO 81] Clocksin, W. F. and Mellish, C. S., Programming in Prolog. Germany: Springer-Verlag, 1981.

7. [COH 83] Cohen, P. R. and Grinberg, M. R., "A Theory of Heuristic Reasoning About Uncertainty", *Artifical Intelligence*, Vol. 4, 1983, pp. 17-24.

8. [COL 83] Colmerauer, A., "Prolog in 10 Figures", *Proceedings International Joint Conference Artifical Intelligence*, 1983, pp. 488-499

9. [DAV 77] Davis, R., Buchanan, B. G. and Shortcliffe, E. H., "Production Rules as a Representation for a Knowledge-Based Consultation System", *Artifical Intelligence*, Vol. 8, 1977, pp. 15-45.

10. [DAV 82] Davis, R., Lenat D. B., Knowledge-Based Systems in Artifical Intelligence. N.Y.: McGraw Hill, 1982.

11. [DUD 76] Duda, R. O., et al., "Subjective Baysian methods for rule-based inference systems", *AFIPS Conference Proceedings*, Vol. 45, 1976, pp. 1075-1082.

12. [DUD 78] Duda, R. O., et al., Semantic network representations in rule-based inference systems. In Waterman, D. and Hayes-Roth, F., eds., Pattern-Directed Inference Systems. N.Y.: Academic Press, 1978.

13. [FAH 79] Fahlman, S., "NETL: A System for Representing and Using Real-World Knowledge". Cambridge, Ma.: MIT Press, 1979.

14. [FIK 71] Fikes, R. E., and Nilsson, N. J., "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", *Artifical Intelligence*, Vol. 2, 1971, pp. 184-208.

15. [FIN 79] Findler, N. V., Associative Networks. N.Y.: Academic Press, 1979.

16. [GOL 77] Goldstein, I. P. and Roberts, R. B., "NUDGE: A Knowledge-Based Scheduling Program", *Proceedings International Joint Conference Artifical Intelligence*, 1977, pp. 257-263.

17. [HAY 77] Hayes, P. J., "In Defense of Logic", *Proceedings International Joint Conference Artifical Intelligence*, 1977, pp. 559-565.

18. [HAY 81] Hayes, P. J., The Frame Problem and Related Problems in Artifical Intelligence. In Webber, B. L. and Nilsson, N. J., eds., Readings in Artifical Intelligence. Palo Alto, Ca.: Tioga, Publishing Co., 1981.

19. [HAY 83] Hayes-Roth, F., Waterman, D. A. and Lenat, D. B., Building Expert Systems. Reading, Ma.: Addison-Wesley, 1983.

20. [HEN 75] Hendrix, G. G., "Expanding the Utility of Semantic Networks through Partitioning", *Proceedings International Joint Conference Artifical Intelligence*, 1985, pp. 115-121.

21. [HUN 75] Hunt, E. B., Artifical Intelligence. N.Y.: Academic Press, 1975.

22. [ISR 84] Israel, D. J. and Brachman, R. J., Some Remarks on the Semantics of Representation Languages. In Brodie, M. L., Mylopoulus, J. M. and Schmidt, J. W., eds., On Conceptual Modling. N.Y.: Springer-Verlag, 1984.

23. [JOS 78] Joseph, L. P., "Production System Conflict Resolution Strategies". Technical report TR-78-3, Dept. of Computer Science, University of British Columbia, Vancouver, BC, 1978.

24. [KIN 84] Kinard, R., "Expert Computer Systems", *High Technology*, January, 1984, pp. 12-34.

25. [KLA 78] Klahr, P., Planning Techniques for Rule Selection in Deductive Question-Answering. In Waterman, D. and Hayes-Roth, R., eds., Pattern Directed Inference Systems. N.Y.: Academic Press, 1978.

26. [KOL 82] Kolata, G., "How Can Computers Get Common Sense", *Science*, Vol. 217, 1982, pp. 3-15.

27. [MCD 78] McDermott, J. and Forgy, C., Production System Conflict Resolution Strategies. In Waterman, D. and Hayes-Roth, F., eds., Pattern-Directed Inference Systems. N.Y.: Academic Press, 1978.

28. [MIC 82] Michie, D., Introductory Readings in Expert Systems. N.Y.: Gordon and Breach Science Publications, 1982.

29. [MIC 85] Michaelson, R. N., Michie, D. and Boulanger, A., "The Technology of Expert Systems", *BYTE*, Vol. 10:4, 1985, pp. 303-312.

30. [MIN 75] Minsky, M., A Framework for Representing Knowledge. In Winston, P. H., ed., The Psychology of Computer Vision. N.Y.: McGraw Hill, 1975.

31. [MIN 82] Minsky, M., "Why People Think Computers Can't", *Artifical Intelligence*, Vol. 3, 1982, pp. 3-15.

32. [NEG 85] Negoita, C. V., Expert Systems and Fuzzy Sets. Menlo Park, Ca.: Benjamin/Cummings, 1985.

33. [RIC 83] Rich, E., Artifical Intelligence. N.Y.: McGraw Hill, 1983.

34. [ROB 68] Robinson, J. A., The Generalized Resolution Principle. In Michie, D., ed., Machine Intelligence 3. N.Y.: American Elsevier, 1968.

35. [ROS 79] Rosenbloom, P. S., XAPS Reference Manual. Technical Report, Department of Computer Science, Carnegie Mellon University, 1979.

36. [VAN 81] VanMelle, W. J., System Aids in Constructing Consultation Programs. UMI Research Labs, 1981.

37. [WAT 78] Waterman, D. A. and Hayes-Roth, F., An Overview of Pattern-Directed Inference Systems. N. Y.: Academic Press,1978.

38. [WIN 84] Winston, P. H., Artifical Intelligence, second ed. Reading, Ma.: Addison-Wesley, 1984.