A Knowledge-Based Expert Systems Primer and Catalog

Bruce K. Hillyer

Department of Computer Science
Columbia University

October 1985

CUCS-195-85

Abstract

For more than 20 years, artificial intelligence techniques have been applied to the development of computer programs that solve difficult problems. Although several expert systems are well known, it is all too easy to circumscribe the field based on these few examples. The purpose of this paper is to present the fundamentals of the field (the Primer), and to give a broad overview via concise descriptions of many rule-based expert systems and knowledge engineering frameworks (the Catalog).

1 Introduction

This paper is offered to persons who have some awareness of artificial intelligence techniques, and who would like to gain an overview of the rule-based expert systems field. The fundamental terminology and concepts are presented here, together with numerous concise descriptions of individual expert systems and tools for expert system construction. Attention is given to the manner in which knowledge is represented, and the ways inferencing is performed.

Previously published articles have described seminal expert systems, elucidated general principles, and have presented techniques for constructing expert systems. Among these are [Davis and King, 1976; Feigenbaum, 1977; Hayes-Roth, Waterman,

and Lenat, 1978; Waterman and Hayes-Roth, 1978; Buchanan, 1982; Ennis, 1982; Stefik et al., 1982; Buchanan and Duda, 1983; Duda and Shortliffe, 1983; Minker, 1983; Nau, 1983; Hayes-Roth, Waterman, and Lenat, 1983; Kobler, 1984; Hayes-Roth, 1985]. The present paper differs from these in that it covers a greater number of systems, rather than treating a smaller number in depth.

The next section presents the basic terminology and concepts of the field, and section three describes several tools, languages, and environments for building expert systems. Section four gives concise reviews of many expert systems, indicating the variety of domains, approaches, and aims. Concluding remarks are in section five, followed by references and an index.

2 Preliminaries

Two fundamental definitions are offered here.

- An *expert system* is a computer program to solve the difficult problems that a human expert solves.[1]

- A *knowledge-based expert system* is an expert system that has the ability to solve its problems by virtue of explicit, declaratively represented knowledge of the problem domain, not just clever algorithms.

Thus *expert* refers to the quality of the problem solving, and *knowledge-based* to the means of solution. These definitions are broad. The first admits all techniques that work well in practice, rather than focusing on approaches necessarily considered "AI-like". The second states a defining characteristic without explicitly requiring subsystems such as knowledge acquisition and explanation, or specifying a strategy such as heuristically guided search space exploration. The given definitions are intended to connote a weak Turing test[2] quality: an expert system should behave as if it understands the problem area at the level of a human expert; it should not appear to perform a simple task, or to compute by rote formulae.

---

[1] We appeal to the reader's understanding of "human expert".

[2] The Turing test of artificial intelligence [Turing, 1963] may be formulated as follows. A human tester holds conversations via two terminals, one of which is connected to a computer, and the other to another human. If the tester is unable to determine from the conversations which terminal is connected to the computer, the computer system has passed the Turing test.

Duda and Shortliffe [1983] gave somewhat similar definitions, but pointed out the problematic nature of such terminology:

> The phrase "knowledge-based systems" is often preferred to "expert systems," since there are no uniquely qualified human experts for a large number of AI applications; however, both phrases are sufficiently vague that the latter can be applied to almost any program that works well and the former to almost any program at all.

The term *performance* as applied to expert systems, refers not to the speed of inferencing, but rather, to the quality of the conclusions drawn by the system.

The structure most used for implementing the reasoning portion of high-performance expert systems is the *production system*, although there is interest in applying logic programming languages such as Prolog to the construction of expert systems [Fuchi, 1982; Treleaven and Lima, 1982; Genesereth and Ginsberg, 1985]. As a formal mathematical construct, the production system was proposed by Post [1943] as a general computational mechanism, and has power equivalent to a Turing machine. Many variations on this formalism have been devised for computer implementation, so despite the common ancestry, there is no single definition or nomenclature for production systems today. There is, however, a widespread emphasis on the explicit declarative representation of knowledge and factual data.

The basic components of an artificial intelligence production system,[3] as illustrated in figure 1 are:

1. A body of knowledge represented by rules having an antecedent-consequent structure.

2. A collection of facts represented as constants (either simple or structured).

3. A mechanism for applying rules to facts to deduce new facts.

Insert figure 1 (Basic components of an AI production system) here.

The first component is called the *production memory* or the *knowledge base* or *rule*

---

[3]See [Hayes-Roth, 1985], p. 928 for a diagram of a complex rule-based system.

base. The minimal components of a rule are the *antecedent* and the *consequent*. Other names for the antecedent are the left-hand side (*LHS*), the *condition*, or the *pattern*. The consequent is also known as the right-hand side (*RHS*) or the *action*.

In *small-rule* production systems, rules consist of a few text lines stating the antecedent and consequent. The term *pure* refers to small-rule systems in which the rules are purely declarative, as opposed to systems in which rule portions represent executable procedures. Small rules are illustrated in figure 2.

In *large-rule* models, a rule may consist of a pair of very large functions expressed in a programming language such as Lisp. In this case, the rules are commonly called ***knowledge sources*** to emphasize the fact that they encapsulate large independent "chunks" of domain knowledge. One common data structure for representing a knowledge source is a *frame*[4] [Fikes and Kehler, 1985], with *slots* representing antecedent tests, consequent actions, screening and triggering tests that quickly determine the potential applicability of the knowledge source, values such as heuristic estimates of the quality or certainty associated with conclusions asserted by the rule consequent, hierarchical relationships with other knowledge sources, and data structures to maintain private state information for the knowledge source. A large rule is illustrated in figure 3.

The essential characteristic of a rule, whether in a small or large model, is that it represents a discrete portion of knowledge concerning the problem domain. This is quite different from a routine in a procedural programming language, which may simply support other computational units without having meaning external to the program. The collection of rules in a production system may be one global pool, or may form structured taxonomies or partitioned sub-collections of knowledge about portions of the domain.

---

[4]An AI frame is an aggregate data object similar to a record in languages such as Pascal. The components of a frame are called slots. Each slot, representing an attribute of the object modeled by the frame, holds a value or list of values, or refers to a computational procedure that produces a value for the slot when invoked. Frames, linked via pointers in slots, form larger structures such as semantic networks or generalization hierarchies that represent relationships such as "is-a-member-of". The traversal of links in such a structure is a computational method for obtaining deductions. For example, an individual member of a class "inherits" a default characteristic of a prototypical class member, in the absence of an explicit value stored for that individual, by traversing the class membership link.

Insert figure 2 (Small-rule productions) here.

Insert figure 3 (Frame-based knowledge source) here.

The second component of a production system is commonly called the *data base* or the *working memory*. This normally represents the temporary state of the world modeled by the problem-solving process, although some artificial intelligence production systems also store certain long-term knowledge in the data base. Members of the data base are commonly called *facts* or *working memory elements*. As with the rule base, the data base may be structured or partitioned to obtain efficient execution or to form a more natural model of the problem domain.

The third portion of a production system is the mechanism that applies the knowledge base to the data base to obtain inferences. This mechanism is often called the *interpreter*, even though it may be implemented by compiling the knowledge base to a procedural program in the native instruction set of a computer.

Two basic classes of production system interpreters are distinguished by the manner in which deductions are made. The first class is known by the names *forward-chaining*, *antecedent*, *data-driven*, or *event-driven*. The term forward-chaining refers to sequential chains of rule activations, in which the state of working memory causes the selection of a rule, which executes to change the state, leading to the selection of another rule. Antecedent refers to the selection of rules by examination of their left-hand sides; the terms data-driven and event-driven suggest that the modification of working memory leads to the selection of the next rule. This pattern of execution, as illustrated in figure 4, may be obtained by iterating the following three step cycle.

1. MATCH: The interpreter compares the facts in working memory with the rule antecedents to find those rules that are *satisfied*. A minimal set of facts that jointly satisfy the tests of one rule is called an *instantiation* of that rule. The collection of instantiations of all rules is known as the *conflict set*.

2. SELECT: One or more instantiations are chosen for execution. This

selection is important: it must prevent infinite inferencing loops, and should focus the system's attention on the most important and promising subproblems. One common approach chooses an instantiation by a *conflict resolution* strategy based on static properties such as the recency of data in the instantiations and the specificity of the instantiated rules. Another scheme selects an instantiation from a dynamically prioritized *agenda* maintained by reasoning processes.

3. ACT: The selected instantiation(s) are executed (*fired*), performing the actions specified by the consequents of the instantiated rules. These actions typically add new facts to working memory, delete or modify old facts, and perform I/O.

Insert figure 4 (Forward-chaining rule execution) here.

The other principal class of production system interpretation is known as *backward-chaining*, *consequent*, or *goal-driven*. In this pattern, inferencing begins with a goal for the system to achieve. The rule consequents are examined to determine which rules could achieve the goal, then the antecedents of those rules become new sub-goals. Thus the reasoning proceeds backwards from a desired goal state to facts in the current working memory, as illustrated in figure 5. The term "goal-directed" has been criticized by Clancey [1984], who states:

> In fact, "goal directed" characterizes any rational system and says very little about how knowledge is used to solve a problem.

Nevertheless, the backward-chaining, goal-driven style of inferencing is a common and effective computational mechanism for systems that perform classification, diagnosis, and analysis.

Insert figure 5 (Backward-chaining rule execution) here.

It is to be noted that production systems have been employed in AI research for two distinct purposes. In "cognitive AI", at the juncture of computer science, psychology, and philosophy, production systems have served as a tool for modeling and simulating theories concerning human thought processes. In expert systems research, production systems have served as a programming language for implementing high performance systems that solve difficult problems effectively,

without necessarily imitating human reasoning. The distinction is easily seen in the answer to the following question: "Is it desirable for the system to commit the same types of errors that humans do?"

The previous discussion has given a general overview that touches on several issues of interest to expert systems researchers. These matters, as described in more detail below, fall into three categories.

1. The computational mechanisms of production systems.
2. The implementation and continuing development of an expert system throughout its lifespan.
3. The utility of an expert system.

2.1 Computational Mechanisms

Fundamentally, an expert system is a computer program. It differs from procedural programs in that the computation is viewed as the application of knowledge to a collection of facts, rather than the execution of algorithms on data structures. As such, there are issues of how to represent the knowledge, how to apply the knowledge to the facts, and how to focus the attention of the reasoning mechanisms.

Typically, the knowledge is captured in rule form as a production system. Several varieties of production systems have been identified, based on whether the rule set is global or clustered into "subroutines", whether there is an implicit global reasoning mechanism or explicit control structures for subsets of rules, and the *granularity*: the amount of knowledge captured by each rule. Rules with fine granularity consist of a few simple conditions and actions, while coarse-grain rules may comprise complex frame structures as described earlier. In addition, rules may be purely declarative, or may contain procedural code. Furthermore, the rules may be organized in a hierarchy of reasoning levels, with abstract *meta-rules* controlling the application of the concrete problem-domain rules [Davis, 1980]. The structure of the data base is also subject to wide variations. Factual knowledge may be represented by tuples, frames, linked-lists, or graph structures, and facts may be global, or may be clustered into subsets representing different portions of the

abstract problem solving space. It is beyond the scope of this paper to discuss the implications and applicability of all these variations; books such as *Pattern-directed Inference Systems* [Waterman and Hayes-Roth, 1978] and *Building Expert Systems* [Hayes-Roth et al., 1983] discuss these issues at length.

The *direction of inferencing* of a production system is a major characteristic. For diagnosis and classification problems, which tend to have a small number of solution states, backward-chaining systems are frequently chosen, since straightforward and reasonably efficient techniques exist for searching from a potential solution back to data that confirm or refute it. Forward-chaining is more natural for systems with a large number of ending states, as well as systems that must be responsive to external input, and systems that process endlessly with no final goal state. In addition, forward-chaining systems can implement *demons*: modules that observe the data base, automatically activating to perform specific tasks when necessary. A number of existing expert systems include elements of both backward-chaining and forward-chaining; the former to control the overall process of solution, and the latter to make immediate deductions when new facts are asserted into the data base. Clancey [1984] points out the distinction between the processing mode of the production system interpreter, which may be forward- or backward-chaining, and the abstract searching technique in the problem space, which may be data- or hypothesis-directed. The expert system named R1 [McDermott, 1980] is cited as an example of a system performing hypothesis-directed search via a forward-chaining implementation.

Another issue pertaining to the interpretation of production systems involves the means for deciding, in case multiple rules are applicable, which to pursue. One common technique is *static conflict resolution*, in which the conflict set is ordered by factors such as the recency of the data matching the rules' antecedents, and the specificity of the rules' conditions. The former criterion tends to focus the system's attention on one matter at a time, and the latter applies special-case knowledge in preference to general defaults. A second technique is *agenda/refinement*. In this method, the rule instantiations are screened for applicability, and the collection of applicable instantiations is refined (i.e., made smaller) by a reasoning or evaluation process. The refined set is then prioritized, with the result recorded in a data

structure known as the agenda. The agenda/refinement process is performed either by selection and scheduling procedures built into the interpreter, or by meta-rules or knowledge sources written for this purpose. Appropriate control of the order of rule execution can cause production systems to implement typical artificial intelligence search techniques such as means-ends analysis, heuristic search, and problem reduction. Forgy and McDermott [1977] describe a static conflict resolution strategy, and Davis [1980] discusses agenda/refinement by meta-rules. An example of a frame-based production system with agenda control is given in [Smith and Clayton, 1980]. Georgeff [1982] describes a technique for the control of a production system by a finite automaton, stack machine, or Turing machine.

An important class of coarse-grain systems utilize the *blackboard* model of computation. This model consists of a global data structure called the blackboard, a number of specialist knowledge sources, and a scheduler. The blackboard contains data organized into "areas" on each of several "levels", representing the state of the problem-solving process at several levels of abstraction. Knowledge sources serve as specialists attentive to changes in the blackboard, acting to refine hypotheses, integrate information from multiple areas, and map between differing levels of abstraction. A typical knowledge source consists of a declarative trigger indicating which regions of the blackboard are of interest, together with a pair of functions forming an antecedent-consequent pair. The scheduler coordinates the actions of the domain knowledge sources, granting processing resources to those that seem most likely to make significant progress. To do this, the scheduler first examines the triggers to determine which knowledge sources are applicable to the current situation. Then it gives control to the antecedent of each relevant knowledge source in turn. The execution of an antecedent determines whether the associated consequent is currently applicable, and if so, extracts and preprocesses relevant blackboard data. After the antecedents finish, the scheduler selects a consequent to execute, based on measures of credibility of competing potential conclusions, the probable effects of running each consequent, and the global significance of those effects. The selected consequent performs a computation on the extracted facts (together with private data), modifies the blackboard as a means of communicating its conclusions to the other knowledge sources, and returns control to the scheduler.

The scheduler is frequently implemented by special knowledge sources that operate on a separate scheduling blackboard. Erman et al. [1981] describe Hearsay-II, which introduced the blackboard model.

2.2 Implementation and Development

Software engineering concepts such as rapid prototyping, modularity, modifiability, and reusability apply to expert systems just as they do to more traditional programming forms. This becomes increasingly evident as larger expert systems are constructed.

Early experiences in developing expert systems showed that *knowledge acquisition*, i.e., obtaining the knowledge of human experts, is difficult and time-consuming. Human experts do not seem to maintain their knowledge in the form of explicit, consciously accessible rules. Consequently, ***knowledge engineers*** cannot first interview a human expert and then write a competent expert system. An approach similar to rapid-prototyping has proven effective. A small trial system is constructed from information obtained in initial interviews, and then the knowledge base is refined, corrected, and extended to increase the quality and breadth of the system's expertise. This process of *incremental growth* is best performed by having the human expert modify the rule base in the context of erroneous deductions obtained via the initial rule set. Computer support for this process has been developed, including programs to facilitate the editing of the rule base, programs that compare a proposed rule with the knowledge base in an attempt to find inconsistencies, and programs that inductively form rules from collections of statistical data [Davis and Buchanan, 1977; Davis, 1981; Boose, 1984]. Although research dealing with automated machine learning may hold promise, it has not matured sufficiently to supplant the knowledge engineer.

Since incremental growth of an expert system's knowledge is a practical necessity, the rule base is designed to have properties such as modularity and additivity. The term *modularity* applies to a knowledge base in which each rule captures a portion of domain knowledge relatively independently of all other rules. *Additivity* is the property according to which additional rules augment the breadth or depth of knowledge of an expert system without disturbing the operation of previously

installed rules. Present to varying degrees, additivity can be elusive in expert systems based on many carefully choreographed rule interactions.

The development of early expert systems required years of effort, providing a large motivation for identifying reusable components. Naturally, the designers observed that portions of the expert systems were functionally independent of the particular domain knowledge encoded in the rules. These portions were abstracted, generalized, and augmented with support tools such as rule editors, rule consistency checkers, and automated testing subsystems. The resulting expert system skeletons, known as ***knowledge engineering frameworks***, are the topic of section three of this paper.

2.3 Utility

An expert system must obtain correct results to be useful, but this alone does not suffice. Another desirable property, perhaps a requirement, is that an expert system be able to produce ***explanations*** of its deductions. In a practical sense, explanations are useful during the development of a knowledge base for examining erroneous lines of reasoning. During normal operation, proper explanations increase the credibility of the conclusions presented by an expert system. This is particularly important for such critical tasks as medical diagnosis, or nuclear reactor monitoring. The usual technique for producing explanations is to maintain a history of deductions during the reasoning process, and then to print a list of stored phrases that render into English the sequence of rule activations leading to the conclusion. McKeown et al. [1985] are investigating the application of natural language processing techniques to generate explanations tailored to the particular user and situation.

In many domains, both the problem data and the conclusions of human experts are associated with ***uncertainty*** or imprecision. Consequently, in some expert systems it is necessary to quantify the likelihood or strength of belief of input data, as well as the inferential strength of rules in the knowledge base. During execution, the evaluation of a rule includes a calculation of the degree of certainty with which the rule asserts its conclusion, as a function of the strength of the rule and the certainty of its antecedents. Some systems also have techniques for aggregating the

assertions of multiple rules that independently obtain similar conclusions. The methods by which existing expert systems combine evidence are drawn from a variety of ad hoc and formal mathematical techniques. Notable among these are the uncertainty calculus [Shortliffe, 1981], Bayesian inferencing [Duda, Hart, and Nilsson, 1981; Lemmer and Barth, 1982; Pearl, 1982], the Dempster-Shafer theory [Barnett, 1981; Strat, 1984], fuzzy logic and possibility theory [Zadeh, 1983] and other techniques [Quinlan, 1983]. Even the methods derived from formal mathematics seem to require human "tuning" of the confidence measures until the expert system produces reasonable results. No technique for reasoning with uncertainty has been widely accepted as "best". An example to illustrate reasoning with uncertainty is given in figure 6.

Insert figure 6 (Reasoning with uncertainty) here.

Another matter related to the utility of expert systems is the speed of inferencing. The naive approach of matching all rules with all data elements on every inferencing cycle is impractically slow for all but the smallest systems. In most implementations, the matching of rules with working memory is the dominant cost. Some software techniques for higher speed of matching and inferencing are reported in [Hayes-Roth and Mostow, 1975; Lenat and McDermott, 1977; Cohen, 1978; Lenat, Hayes-Roth, and Klahr, 1979; Ghallab, 1981; Forgy, 1982; deKleer, 1984]. In addition, techniques utilizing parallel hardware have been examined, including [Stolfo and Shaw, 1982; Deering; 1984; Forgy et al., 1984; Hillyer and Shaw, 1984; Oflazer, 1984; Stolfo and Miranker, 1984].

3 Tools for Building Expert Systems

The creation of an expert system can be a major undertaking. In view of the resources required, several languages and knowledge engineering frameworks have been introduced to facilitate expert system development. By providing generally applicable support software such as knowledge acquisition subsystems, explanation subsystems, and rule interpreters, these facilities seek to allow the builders of expert systems to concentrate on capturing and utilizing problem-specific knowledge. This section presents brief descriptions of a number of such tools. Several of these, as noted, are generalizations of specific expert systems described in section four.

AGE [Nii and Aiello, 1979] is a collection of tools and partial frameworks for building expert systems, based on the model originally developed for the HEARSAY-II expert system, together with an intelligent front-end that assists the user in constructing knowledge-based programs. The principal portions of an expert system implementation in AGE are domain-specific knowledge sources, and modules that schedule the execution of the knowledge sources. An AGE knowledge source has several components, including a collection of production rules, lists of events that may trigger other knowledge sources, levels in the hypothesis space to which each knowledge source is applicable, a choice of single or multiple hit strategy (either one or all of the triggers need to be satisfied), and facilities for binding variables. Uncertainty is modeled by a technique similar to that of the MYCIN expert system. Components written by the user select and schedule knowledge sources for execution, but standard modules are furnished for common control regimes such as event-driven and goal-driven inferencing.

ARBY [McDermott, 1982] is a special-purpose environment for writing expert systems that diagnose faults in electronic equipment. This problem domain permits reasoning with shallow models of electronic subsystems connected by signal flows, but is complicated by the fact that most diagnostic information is not readily available, and has considerable cost to obtain (cutting wires, replacing subsystems....) ARBY is partly rule-based, and utilizes mechanisms similar to those of the CADUCEUS expert system to refine and combine hypotheses. The system has two main modules, written in Franz Lisp. The first module reasons about the electronic system, generating hypotheses and sifting evidence by performing deductions on a set of predicate-calculus rules. The second component handles interaction with the user. Question asking is ordered based on the importance of the evidence for confirming or denying the leading candidate hypotheses, balanced with costs of getting the information, and subject to precedence constraints supplied by the expert system designer.

ARS [Stallman and Sussman, 1977] is a rule language for domains in which problem solving may proceed by the symbolic relaxation of local constraints. Rules are implemented as pattern-directed invocation demons monitoring an associative data base, performing single-step forward-chaining deductions. Demons having satisfied

trigger conditions are placed on queues at various priority levels, and operate on facts stored in a data base. To facilitate matching, the data base is hash-indexed on the atoms contained by facts. ARS also maintains records linking premises to deductions, to provide a basis for explanation and to support dependency-directed backtracking when a contradiction is obtained. ARS also has routines for algebraic manipulation. Some limitations of ARS include difficulty modeling time-dependent behavior, a single level of detail (burying "explanations" in minutiae), and a lack of goal-directed and attention-focusing control facilities.

CENTAUR [Aikins, 1979; 1980; 1983] is a framework for developing expert systems that use a hypothesize and match approach to problem solving. Patterns of knowledge in the domain are organized into frame structures called *prototypes*. Slots in the prototypes represent information such as values, plausible ranges, importance, control knowledge, and production rules that infer missing values. The development of CENTAUR was motivated in part by an observed deficiency in certain of the early backward-chaining expert systems. In these systems, the order of the clauses in a rule determines the order of the backward-chaining search, and also the order in which questions are asked of the user. This means that rules have significant non-modular interactions, since there is no clear separation between control knowledge and domain knowledge. A generated explanation of the system's behavior that treats control rules the same as rules containing domain knowledge can be confusing, and implicit control knowledge is not readily explained at all. The CENTAUR prototypes explicitly organize and focus the searching and question asking, to diminish the amount of hidden control knowledge in a system.

EMYCIN [van Melle, 1979] is an environment for implementing knowledge-based consultation programs, developed by generalizing the basic framework of the MYCIN expert system. EMYCIN provides a goal-directed, backward-chaining interpreter for production rules grouped by contexts, an editor for the data base, an explanation facility that paraphrases rules in either English or an Algol-like language, and a knowledge acquisition subsystem. The data base is in the form of attribute-object-value triples with associated certainty factors. Although normal execution is goal-driven, there is a limited ability for data-driven deductions to be made upon the assertion of new facts into the data base. Inverted indices on the goals are

maintained to increase the speed of backward-chaining, and techniques are employed to avoid redundant testing of facts in the data base, and redundant testing of patterns [Stefik et al., 1982]. Shortliffe et al. [1981] point out several aspects of EMYCIN that potentially limit its range of application. EMYCIN has been extended by Teknowledge to become the commercial product named KS-300.

EXPERT [Weiss and Kulikowski, 1979] is a system for designing and building models for consultation, developed by generalizing the CASNET expert system. In an EXPERT model, input attributes termed *findings* take on numerical or boolean values that, once determined, remain constant. Support for conclusions called *hypotheses*, which may be structured into taxonomic-causal networks, is derived from findings and other hypotheses by three classes of decision rules. The rules executed first make "common-sense" forward deductions from existing findings to new findings, and evaluate correlated findings to generate modifiers. There is no automatic mechanism for aggregating the evidence of multiple findings, which leads to predictable system behavior since rules only interact in ways specified by the designer. The rules executed second reason from findings to hypotheses. If multiple rules suggest differing confidence ratings for a hypothesis, the largest rating is accepted (this is the fuzzy logic technique). The rules executed third are examined in the order listed by the designer, to perform backward-chaining searches through the taxonomic-causal network. Data collection induced by this search is via prepackaged sets of questions that are to be asked of the user, which organizes the interaction in ways deemed reasonable by the system builders. Questions already satisfied by previous deductions are automatically suppressed. An EXPERT model is compiled into an intermediate form that is interpreted by a runtime package written in FORTRAN for speed and portability. SEEK is the knowledge acquisition subsystem of EXPERT.

HAPS [Sauers and Walsh; 1983] is a hierarchical, augmentable production system environment directed towards future expert systems that require large rule and fact bases, and speed sufficient to meet real-time constraints. HAPS uses goal-directed execution to focus system efforts, and a hierarchical working memory structure parallel to the goal structure to reduce matching and to facilitate garbage collection when a goal is achieved. This structure also permits the storage of rule sets in

secondary memory, to be fetched only when needed. One may view this as a "library of production rule sets". To achieve real-time execution, the system is designed to make approximations and ignore inessential tasks when overloaded. A suggested approach is to identify, by the collection of statistics, rules that derive useful conclusions quickly, or are likely to result in genuine progress, and restrict execution to those rules when the system is overloaded. Real-time external events may well occur faster than the system's rule firing rate. This would render unusable any organization in which, each time any fact is updated, all production instantiations depending on that fact must be removed or tagged invalid, and the new fact must be matched to form an up-to-date conflict set. One potential solution to this problem is to store real-time input in efficient data structures that are examined at conflict resolution time to narrow the set of applicable rules. The authors note that logical deduction, needed to support frame hierarchies with attribute inheritance, may interfere with speedup techniques based on discrimination nets.

HEARSAY-III [Balzer et al., 1980] is a framework for building expert systems with multiple knowledge sources and multiple levels of representation, reasoning, and control. A knowledge source in HEARSAY-III consists of a declarative trigger pattern together with Interlisp procedures serving as antecedent and consequent. At appropriate times, trigger patterns are matched with the configuration of data in a structured data base called the blackboard (as described in section 2.1 on p. 9). Successful matches cause the corresponding antecedents to calculate a scheduling-blackboard class (a priority level) on which an activation record will be created, and to collect data for the activation record. Scheduling knowledge sources examine the activation records and choose one. The selected consequent runs to completion, typically aggregating interpretations at one blackboard level to composite interpretations at more abstract levels, manipulating alternative competing interpretations, and criticizing alternatives. If constraint violations are detected (say, upon attempting to aggregate incompatible interpretations), the blackboard context containing them is considered "poisoned", and only special knowledge sources ("poison-handlers") are permitted to run in that context until it is unpoisoned. A typical poison handler might split the context into two competing

ones, each self-consistent. HEARSAY-III was designed for power and flexibility, not speed. It is an extension of the AP3 relational database system, which in turn is written in Interlisp. AP3 has strong typing, and implements contexts, demons, and constraints.

HPRL [Rosenberg, 1983], an extension of the artificial intelligence language FRL, is a heuristic programming and representation language. Data, rules, and the rule interpreter are all represented by frames. A supplied set of Lisp functions execute rules by strategies such as forward- or backward-chaining, and manipulate agendas, build and traverse decision trees, and record information for backtracking. Additional functions may be added by the user. In particular, "rule domains" may be created to partition rules into subsets relevant to restricted subproblems, with different methods for evaluating the rules in various domains. Since rules and data have the same form, meta-level reasoning is easily accommodated. Since rules are frames, they are not restricted to condition-action form, and may have additional information such as caveats and suggested uses.

KEE [Fikes and Kehler, 1985; Kehler and Clemenson, 1984], a commercial product of IntelliCorp, is a framework for developing knowledge-based systems. It combines a frame data language and inheritance mechanism with rule-based reasoning. The system uses frames to represent both individual entities and classes. A class frame contains prototypical characteristics of class members as well as attributes of the class as a whole. Inheritance hierarchies are supported through automatic inferencing on is-a-member-of and is-a-subclass-of links. The slots in KEE frames are flexible and powerful, having the ability to represent partial descriptions and constraints on unknown values. The attachment of Lisp procedures to slots permits production rules to be represented, with rules structured by class membership links, and having access to the database of inheritance hierarchies. Both backward-chaining and forward-chaining rule interpretation disciplines are supported.

LOOPS [Stefik, Bell, and Bobrow, 1983] is an artificial intelligence programming language designed in the belief that some tasks are best accomplished by rules, and others by procedures, demons, or object-oriented programs. LOOPS integrates modules written in any of these styles into a hierarchical subroutine organization,

transferring control by message passing, or automatic activation as a side-effect of fetching or storing values, or by procedure call. For instance, the execution of a rule can set up procedural demons. Rules are grouped into small ordered sets embedded in control frameworks that concisely represent how the rules are to interact. Given that clustering and control are needed, LOOPS implements carefully chosen explicit structures, rather than forcing the programmer to include extra clauses in all domain rules, and extra rules to assert and retract control tags that direct a static conflict resolution strategy. Rules in a LOOPS rule set are tested in the order of appearance, with an option for starting over at the top when a rule fires. Rule prefixes may specify "execute only once" or "test only once", and rules may be embedded in standard iteration constructs, with automatic history gathering to support explanation or belief revision, and built-in certainty factor calculations. Rule antecedents and consequents may call other entities in any of the four programming styles, and rule sets may be called from entities of any of the four types. Multi-tasking and co-routining are supported, facilitating agenda-based control. There is a compiler that translates rule sets to Interlisp.

OPS [Forgy and McDermott, 1977; Forgy, 1979; 1981; 1983] is an evolving language and execution system for forward-chaining rule-based programming. OPS systems have a global production memory of rules and a global working memory of facts in the form of lists of literal attribute-value pairs. In each recognize-act cycle a complete match of rules with working memory is effectively made, and one rule instantiation is chosen for execution by a static conflict resolution scheme. OPS was designed under the principle that knowledgeable rules may undertake substantial actions, but conditions should be simple. The rationale is that powerful patterns would cause overhead during matching, retarding processing, but complex actions consume resources only when executed. Consequently, OPS actions may call user-written Lisp functions. Normally, OPS rules are simple so that the ratio of matching to acting remains large, which is considered the appropriate utilization of rule-based systems. Consistent with these ideas, OPS does not provide structures for rule subsetting, partitioned working memory, conflict resolution by executing all instantiations or by meta-rule reasoning, or automatic backtracking. Rules may modify the production memory, however, and both partitioned working memory and

backtracking schemes may be simulated by the explicit inclusion of grouping tags in working memory elements. The Rete algorithm [Forgy, 1982] provides an efficient discrimination network technique for executing OPS. The most recent member of the OPS family, OPS83, allows user-written predicates in rule antecedents, and has strong typing and composite data structures so that rules can be compiled into efficient native code for execution, rather than being interpreted. [Ennis, 1982] reports an experience in developing an expert system in OPS5. System development proceeded quite rapidly, but the primitive control of OPS5 was seen as a drawback: control and domain knowledge become interspersed in almost every production.

RITA [Anderson and Gillogly, 1976; Waterman, 1979] is a language for implementing rule-directed agents that insulate users from low-level details of a computer system. To facilitate this, RITA programs have the ability to exert low-level control of a computer, including starting and monitoring multiple subprocesses. RITA rules, expressed in an English-like syntax, are executed both by forward-chaining and by goal-driven deduction. Rules are examined in their textual order, the first applicable one is executed, then matching starts over at the beginning. All objects and rules are global, and require processing on each cycle, impairing execution speed. Performance is acceptable for the intended tasks, but is insufficient for large expert systems. In addition, hierarchical and inheritance data structures are difficult to represent.

ROSIE [Fain et al., 1981; Waterman, 1979] is a language for building expert systems. The fundamental building blocks are production rules, expressed in an English-like syntax, with both forward- and backward-chaining execution. There are two types of rules. *If-then* rules are existence-driven, firing as long as the conditions are true, even repeatedly on the same data, with the actions performed once during each production system cycle. *When-then* rules fire just once for each knowledge element matching their conditions, giving event-driven, demon-like execution. Rule matching is organized for efficiency by recognition nets. The data may be structured hierarchically via *instance-of* and *member-of* relations to support abstraction and inheritance. Additionally, rules and data may be grouped for separate access and execution, in which case rule and data sets are examined only when deemed relevant by the user or the ROSIE monitor. This provides the

potential of maintaining only currently active and relevant modules in primary memory. The event-driven monitor also is said to simplify the programming, permitting the user to create rule sets that act as collections of independent demons.

YAPS [Allen, 1983], a language for the implementation of rule-based systems, may be viewed as an extension and generalization of OPS5. Yaps generalizes rule antecedents to include user predicates, comparison of arithmetic expressions, and nested lists of constants and variables to be matched with the Lisp lists in working memory. A rule is represented by a Lisp function, which may be compiled, whose arguments are the antecedent variables and whose body is composed of unrestricted Lisp code (including calls to the usual OPS actions). Discrimination net techniques similar to Rete match evaluate rule antecedents. YAPS may be used in conjunction with object-oriented programming, and a YAPS program may be subordinate to other Lisp code. The form *(fact xxxx)* asserts a new fact into the working memory, and *(goal xxxx)* adds a goal element to the working memory. Whenever a goal is added, whether by the production system or by another Lisp program running in the same environment, the production system begins execution, continuing until all goals are removed from the system or no satisfied productions remain. Similarly, if there are outstanding goals and a fact is asserted, production system execution is initiated. Thus YAPS implements demons. YAPS supports multiple working memories and multiple production memories, as well as a global working memory. Insertions into the global working memory cause duplicate copies to be inserted into all local working memories.

4 Descriptions of Expert Systems

The expert systems briefly described in this section are organized into four clusters to facilitate browsing. The first consists of fine granularity rule-based systems: systems that represent domain knowledge in numerous small rules. The second comprises coarse granularity systems that utilize large knowledge sources, and the third contains systems designed to reason with uncertainty. These categories do not represent a disjoint partition of the possibilities, they simply gather systems that have overt similarities in features and emphases. A fourth subsection mentions

interesting systems that do not fall naturally into one of the three preceding groups. The description of each system ends with a list (in braces) of index terms that identify significant properties and issues. Within each cluster the seminal systems are mentioned first, and others follow in alphabetical order, except that related systems are mentioned together.

4.1 Small Rules: The OPS Group

Most of the systems in this group have their roots in early work at Carnegie-Mellon University. The rules in these systems typically have a small number of conditions in the antecedent, and few actions in the consequent. These systems generally maintain a model of the problem state in the form of lists of constant attribute-value pairs stored in a global working memory, and rules are chosen for execution by a static conflict resolution strategy. The R1 expert system is a premier example of this style.

R1 [McDermott, 1980; 1982] is an expert system that configures mainframe computers manufactured by Digital Equipment Corp. R1 is distinguished from many other expert systems in that the problem it solves is one of synthesis, not analysis or classification. The original experimental system of 500 forward-chaining OPS rules has been expanded ten-fold as of mid 1985; the rule-based knowledge representation greatly simplified this growth. The system has sufficiently strong knowledge that it performs little search. With few exceptions, it follows the *match method*: subproblems are investigated in an order having the property that there is sufficient knowledge at each step to make a correct choice. Consequently, backtracking is largely unnecessary, and approximately 2000 rule executions suffice to configure a VAX-11/780. During a run, factual knowledge about needed components is retrieved from a separate database and deposited into working memory. The search is guided by the OPS static conflict-resolution mechanism, together with a control strategy implemented by the assertion of *context elements* into working memory. A context element serves to restrict execution to a single rule cluster (about 10 rules). Each cluster has rules that recognize when the subtask is complete, and alter the context elements to pass control elsewhere. {VAX configuration, small rules, OPS5, forward-chaining}

XSEL [McDermott, 1982] is an expert salespersons' assistant for Digital Equipment Corp. It assesses the customer's wishes as expressed in a partial order, explains the various additional items necessary to support those wishes, provides the ability to query the component database, and communicates the final component selections to R1, which configures them into a system. XSEL comprises approximately 3000 rules in small clusters similar in nature to those of R1, and executes 50 rules per customer interaction. {VAX ordering, small rules, OPS5, forward-chaining}

PTRANS [Haley et al., 1983; McDermott, 1983] is a manufacturing management assistant for Digital Equipment Corp. It suggests when and where on the assembly floor to build each VAX, ensuring that the necessary parts are on hand when needed, and tracking progress as problems arise. The system has approximately 1400 rules, but these are not organized as in R1 and XSEL. Approximately 700 rules are particular to 175 subtasks, but the other 700 are applicable to more than one subtask. On the average, there are 34 demon rules applicable to a given subtask, but the clustering of demons is not parallel to the subtask rule clusters. Indeed, the designers have discovered no general characterization of the demon interrelationships. {VAX assembly, small rules, OPS5, forward-chaining}

ACE [Stolfo and Vesonder, 1982] is an expert system designed to provide timely trouble-shooting analyses for the management of telephone cable systems. It automates the search through trouble reports for patterns of faults, and suggests likely causes, appropriate repairs, and preventive maintenance. When plausible patterns are discovered, ACE formulates queries to a database management system that contains trouble reports and maintenance information, to collect other data that may confirm or further illuminate a pattern. Reports are generated and sent via electronic mail to appropriate personnel. ACE deals with wide variations in input; one of its subproblems is to recognize when different trouble reports discuss the same location or problem. Incipient patterns are hypothesized, and the hypotheses are carried forward for future examination with respect to new trouble reports. The inferencing knowledge of ACE is contained in a forward-chaining production system expressed in the OPS4 language [Forgy, 1979]. Rules are selected for execution based on a static conflict-resolution strategy that considers recency of data and specificity of production rules. The "state of the world" is maintained in

a single working memory of constant lists. The initial system reported in 1982 comprised approximately 100 rules, together with 50 lisp functions to interact with the database management and electronic mail systems. ACE is now a commercial product of AT&T. {telephone cable maintenance, OPS4, forward-chaining, external data base, expert user}

DAA [Kowalski and Thomas, 1983; Thomas et al., 1983] is an expert system that designs VLSI chips. It is written in approximately 300 rules in the OPS5 language. A typical snapshot during a DAA execution would show 700 objects and relationships in the working memory, with many instantiations of just a few rules ready to be selected for execution. The system is notable for the amount of work it does: 50,000 rule executions may be required to complete a chip design. {VLSI design, OPS5, forward-chaining}

LDS [Waterman and Peterson, 1980] is a rule-based system with expertise in product liability law. The rules are partitioned into five rule sets (formal doctrine, informal principles, strategies, subjective considerations, and secondary effects) that can be called as subroutines, but only 90 rules for the first two sets were implemented as of the time of writing of the referenced work. The anticipated number of basic concepts in a full implementation is in the hundreds, and the number of rules to adequately represent legal doctrine and strategies is in the thousands. The rules are executed by forward-chaining. Facts (which are also in antecedent-consequent form) are used for backward-chaining: the information that can be inferred from the hierarchy of facts by inheritance and abstraction is a "virtual data base". LDS is implemented in the ROSIE expert system development environment. {legal advice, partitioned rules, ROSIE, forward-chaining}

REACTOR [Nelson, 1982] is an expert system to assist the operators of a nuclear reactor by detecting deviations from normal operating conditions, determining the significance of events, and recommending appropriate responses. It normally reasons forward from known facts, but if insufficient information is available to reach a conclusion, the system reasons backward to determine what information it needs to know. It then queries plant instruments or operators as necessary. Its knowledge is stored in rules and response trees. Rules, which are event-driven, aggregate

observations into known accident classes. Response trees for each safety function represent all possible ways to achieve or maintain the safe condition. If failures disrupt some paths in a response tree, the least-cost remaining path indicates the proper actions. Speed of inferencing has been observed to be a problem. {nuclear reactor monitoring, forward-chaining}

YES/MVS [Griesmer et al., 1984] is a continuous real-time expert system that monitors the complex and dynamic environment of a computer system, taking operator-like actions. Implemented by three communicating processes, it is built upon OPS5, with extensions to allow priority-levels of rules, functions that facilitate communication with the computer under control and operator interface software. Rule consequents are compiled for speed. For this system, OPS5 has also been extended to deal with temporal actions and relations, such as "Assert fact X at future time Y." An additional extension permits the transmission of facts from working memory to jobs running on other virtual machines. YES/MVS comprises 500 rules, including some that are driven from tables stored in working memory. {computer system monitoring, OPS5, forward-chaining, table-driven rules, compiled rules}

4.2 Large Knowledge Sources: The Hearsay-II Group

The systems described here stem from the efforts of the Heuristic Programming Project at Stanford, and from work at Carnegie-Mellon University. The seminal expert systems in this group are DENDRAL and HEARSAY-II, respectively. Common features of these systems include large procedural codes that serve as knowledge sources invoked at appropriate points in the problem-solving process, and large private data structures maintained by these procedures. Sophisticated algorithms schedule the executions of the knowledge sources, and integrate the partial solutions they produce. Many of these systems are based on the blackboard model of computation, described in section 2.1 on p. 9.

CONGEN, DENDRAL, and META-DENDRAL [Buchanan and Feigenbaum, 1981] are portions of an early expert system[5] that seeks to determine the structure of a

[5]The Dendral project commenced in 1965.

chemical from a few hundred points of mass spectroscopy data. CONGEN, written in the Fortran and Sail languages, generates all structures that fit the data, subject to specified constraints. The authors of DENDRAL, which originally was an Interlisp program, were among the first to recognize that a declarative rule base could solve the problems of maintaining the knowledge of a system during substantial evolution and growth. The rules encode heuristics that confine the space of possible solutions. It is a forward-chaining front-end to CONGEN, supplying constraints, handling bookkeeping, and dealing with the combinatorics of placing substitutents (i.e., DENDRAL enumerates gross structures, and CONGEN fills in the details in all possible ways). Thus the ability to generate all possible candidate structures is combined with powerful knowledge to confine the search space, so that the potential examination of millions of structures is reduced to the generation and testing of several of the best. META-DENDRAL is designed to infer, by induction on empirical data, heuristic rules for DENDRAL to use. In particular, it seeks patterns of correlation between molecular fragmentations observed in a mass spectroscope and the substructural features of the source molecules. Aggregated correlations reveal processes such as the breaking of bonds and the migration of atoms between fragments. {chemistry, generate and test, knowledge acquisition}

HEARSAY-II [Erman et al., 1981] is an expert system for speech understanding. It receives the output of a microphone, and determines the sentence that was spoken, given sentences that are well-formed in a restrictive grammar with a 1000 word vocabulary. The blackboard model of computation was introduced by this system to deal with several characteristics of this problem. In particular, the problem search space is very large, there are many diverse sources of knowledge that may be brought to bear on the problem, and both the input data and the knowledge sources are subject to error and inaccuracy. Additionally, the system was highly experimental, so modularity and independence of knowledge sources was important, and a severe processing resource constraint required that processing power be applied judiciously. The knowledge sources in Hearsay-II consist of declarative triggers and pairs of antecedent-consequent procedures written in Sail[6]. They

[6]Sail is a language in the Algol family.

perform aggregation operations such as converting digitized input data to segment hypotheses, segment hypotheses to syllables, syllables to words, and words to phrases. Other knowledge sources feed information back to lower levels by predicting words adjacent to phrases, or syllables that would form predicted words. Additional tasks include the evaluation of competing hypotheses for promotion or disqualification, and scheduling. Processing stops when only one hypothesis remains on the blackboard, or when a time limit is exceeded (in which case the highest rated hypothesis is returned as the solution). The HEARSAY-II source code consists of several hundred pages of Sail instructions. {speech understanding, blackboard, large rules, uncertainty}

CRYSALIS [Engelmore and Terry, 1979], also known as SU/P, is an expert system to determine 3-dimensional protein structure from x-ray crystallography data. The matching of electron density maps with other information to deduce protein structures is "a black art". Consequently, each new protein requires different ad hoc techniques, so the system is designed for change. The system organization is derived from that of HEARSAY-II. Although most knowledge is represented in rule form, knowledge sources that do heavy computation are expressed in procedural code. Knowledge sources are organized into three conceptual levels, known as the domain, task, and strategy levels. Domain knowledge sources post their conjectures, with confidence ratings, on the blackboard. Task knowledge sources organize the work of the domain sources, based on the state of the computation as recorded on the blackboard. Strategy rules select tasks for execution based on a heuristic estimate for each task of the expected progress towards a global solution ("opportunistic scheduling"). {chemistry blackboard, large rules, multiple levels, uncertainty}

MOLGEN [Stefik, 1980] is an expert system to plan gene-cloning experiments in molecular genetics. The technique employed performs the planning task by defining and integrating abstract subplans, which are progressively refined to approximate a solution by the method called *difference reduction* (find a difference between the current state and goal, and develop a plan to eliminate the difference). This planning occurs at three decreasing levels of abstraction, in the strategy, design, and laboratory problem spaces. The interactions between subplans are posted as

constraints to be satisfied in the next more concrete space; corresponding partial solutions are integrated by heuristics under the principle of *least commitment* (delay making arbitrary decisions; wait until circumstances force a choice). An interpreter applies operators to the current configuration of the spaces to (1) generate and test partial solutions subject to constraints, (2) find and reduce differences between goals and the current state, and (3) map from one space to another. The operators are procedural, and state is maintained in frame structures. {planning, large rules, frames, multiple levels, difference reduction, least commitment}

SPEX [Iwasaki and Friedland, 1982] is an extension of MOLGEN that integrates the idea of stepwise refinement of skeletal plans with MOLGEN's layered control structure. In this system, all the domain-specific knowledge is contained in the knowledge base of frames; the procedures that operate in the planning spaces are claimed to be general. Agendas of tasks are managed in the strategy space by simple disciplines such as queue, stack, or priority queue. {planning, frames, agenda, multiple levels}

SU/X [Nii and Feigenbaum, 1977] is a classified military system that, given spectral lines from multiple sources, recognizes and tracks objects through a physical space. For this problem, the HEARSAY-II architecture was extended to implement multiple layers of control structure, and the blackboard was partitioned into distinct areas. The control layers are named hypothesis-formation, hypothesis-activation, and strategy. HASP and SIAP [Nii et al., 1982] are related systems that perform intelligent signal processing for ocean surveillance. {military, blackboard, multiple levels, uncertainty}

4.3 Inference Nets and Uncertainty: The Prospector/Mycin Group

The systems in this group are characterized by two principal ideas. The first is to model the domain by an inference network that reflects some underlying structure in the application domain. Rather than an unstructured collection of production rules, these systems have rules that link the nodes in the network. The second idea is to implement mechanisms for inexact inferencing, either because the input data are approximate or because the knowledge that human experts have in the domain is too weak to obtain definitive conclusions. One common means for

dealing with uncertainty, adopted by the expert system called Prospector, is based on Bayesian probability. Another is the "calculus of uncertainty" developed for the expert system named Mycin.

PROSPECTOR [Duda, Gaschnig, and Hart, 1981; Gaschnig, 1982] is an expert consultant system to aid geologists in evaluating regions for potential ore deposits. Knowledge is represented by production rules indicating how a change in the probability of the antecedent condition influences the probability of the consequent condition. The rules form an inference net that is tuned by ad hoc methods until correct results are produced; probability updating is by an approximate form of Bayes' Rule. Rigor is diluted by assumptions that the set of outcomes are exhaustive and mutually exclusive, and that the input observations are independent [Szolovits and Pauker, 1978]. Also, there is no mechanism for the system to discard wildly wrong observations, or to allow evidence to imply composite propositions [Quinlan, 1983]. In practice, however, the results are insensitive to small perturbations of the input data, which is important since geologists will disagree somewhat about a set of observations. Since the productions neither bind variables nor form inference loops (a->b; b->c; c->a) [Konolige, 1979], the rules forming the inference net can be compiled into straight-line code. This code propagates probabilities from observations to results in one sweep, with no control strategy overhead and no recomputation of intermediate nodes. One PROSPECTOR model for evaluating copper deposits has 94 nodes and 105 productions; compiling reduced the running time from 30 seconds to 3.1 milliseconds. Knowledge acquisition and intelligent consultation assistance have been addressed by Reboh [1980]. {resource exploration, Bayesian inferencing, domain network, knowledge acquisition, uncertainty, compiled rules}

MYCIN [Davis, Buchanan, and Shortliffe, 1977; Clancey, 1983a] is an expert system to diagnose bacteremia and meningitis, and to suggest treatments. The system comprises six components: the patient data base, the knowledge data base, and programs for consultation, explanation, question answering, and knowledge acquisition. The patient data base consists of associative triples of the form object-attribute-value, with 80 attributes and 11 entities pre-defined. An example of such a triple is (identity, organism, bacteroides). The knowledge base is a collection of

production rules (600 in 1981) forming an AND/OR tree; rule antecedents are conjunctions of tests on triples, and rule consequents produce conclusions about triples. In addition, a context hierarchy defines the universe of discourse and the ways objects are related. The rules are applied in an exhaustive backward-chaining search,[7] but reasoning is complicated by the uncertainty of medical knowledge and the imprecision of diagnostic signs. Hence the certainty of an antecedent, calculated by the fuzzy set rule on the triples examined, is multiplied by the certainty factor of the rule itself to obtain a measure of belief of the conclusion. A combining formula increases the belief measure of a conclusion supported by multiple rules, rather than taking the maximal certainty factor imputed by any single rule. {medicine, calculus of uncertainty, fuzzy logic, explanation, backward-chaining, uncertainty}

TEIRESIAS [Davis and Buchanan, 1977; Davis, 1981], originally a portion of the MYCIN system, assists in knowledge acquisition. It is a large Interlisp program designed to facilitate the interactive transfer of expertise from a human to an expert system. In the context of a particular erroneous consultation, it systematically leads a human through the expert system's line of reasoning to discover the point of error. Then it prompts for and monitors the modification or addition of rules, noticing inconsistencies between new knowledge and patterns established by previous knowledge. This latter behavior is made possible by meta-rules that give TEIRESIAS a model of what the expert system knows and does not know. Finally, it performs certain bookkeeping functions, and automatically re-runs consultations to verify the correctness of the changes. {knowledge acquisition, meta-rules}

NEOMYCIN [Clancey, 1983] is a revision of MYCIN in which the search control knowledge is represented explicitly, rather than being implicit in the domain rules and interpreter, so that this knowledge is accessible for computer-aided instruction. A top-down diagnostic strategy is represented in a set of domain-independent meta-rules. Domain rules represent causal relationships, trigger the addition of hypotheses

[7] In the June 1978 version of MYCIN, the largest number of rules relevant to a particular goal was about 50.

to the set under consideration, notice and combine circumstantial evidence (with associated certainty measures), and make deductions, when possible, to determine data without asking the user. Knowledge about disease processes is stored in frames keyed to a disease taxonomy, together with lists of follow-up questions. The system may be viewed as a general procedure that searches a network containing advice at each node suggesting which branch to examine next. One consequence is that there is no backward-chaining present in the domain-level rules; the top-down refinement and screening activities are performed by the meta-rules. {teaching, medicine, frames, domain network, backward-chaining, meta-rules}

GUIDON [Clancey and Letsinger, 1981]; later GUIDON2 [London and Clancey, 1982] is a system designed to teach the information in the NEOMYCIN database. GUIDON2 has three components: a domain expert (NEOMYCIN), a student modeler (IMAGE), and an instructional manager. During a consultation, the system simulates NEOMYCIN to form multiple sets of predictions of student behavior. Then it obtains descriptions of actual behavior. If these are consistent, the student is following a correct approach, reflecting a reasonable set of active hypotheses. If not, forward-chaining rules seek to rationalize the student's behavior, giving a basis for guidance and evaluation. {teaching, forward-chaining}

ONCOCIN [Shortliffe et al., 1981] is an expert system to manage oncology protocols, which are the patterns of treatment and data collection for cancer patients. The system was designed to be acceptable to doctors. It is implemented in two concurrent processes: the *reasoner*, a forward-chaining rule-based system in Interlisp, and the *interviewer*, a Sail program for high speed full-screen display interaction. Control blocks contain scripts of steps to accomplish the tasks of a protocol, which separates control knowledge from the domain rules. Rules, grouped by the contexts to which they apply, are executed by forward-chaining to draw conclusions from new data, and by backward-chaining to deduce needed values. The control blocks cause system behavior to be focused and responsive to the user, avoiding a potential weakness of backward-chaining systems. The design of ONCOCIN facilitates storing and reasoning about temporal patterns of data as well as current values. {medicine, control knowledge, forward- and backward chaining, temporal reasoning}

ABEL [Patil, Szolovits, Schwartz, 1982] is a medical expert system to diagnose acid-base and electrolyte disturbances. It is designed to utilize both shallow and deep causal models of disease to plan the diagnostic questioning. By decomposing the tree of all potential diagnoses before asking the first question, the system can order the set of necessary questions for efficiency and coherency. In particular, implausible responses from the user can be recognized and challenged, invoking an "excuse-finding mechanism". The system plans the diagnostic questioning in a goal-directed manner, but the actual question asking process is performed in a forward-chaining environment so that the system is responsive to the answers given. {medicine, deep models, forward- and backward-chaining, uncertainty}

CASNET [Weiss, Kulikowski, Amarel, Safir, 1978] is an expert system for the diagnosis of glaucoma. The principal idea is to store a network of causal connections among *dysfunctional states*, and test whether a patient has those dysfunctions. Relationships between nodes are stored as production rules, and results of tests for specific states are combined by fuzzy logic applied to causal links from other confirmed or disconfirmed states. Diagnostic question generation is guided by paths in the network; the initial node of a path represents a probable cause (disease), and the terminal node indicates the extent of disease progression. A causal model is attractive because "people seem happier if they understand why something happens than if they merely know that, under the circumstances, it does." Casnet has 100 states, 75 classification tables, and 200 diagnostic and treatment statements [Nau, 1983]. {medicine, domain network, fuzzy logic, uncertainty}

DART [Bennett and Hollander, 1981] is an expert system for fault diagnosis in computer teleprocessing systems. It is seen as the first step in the development of an automated diagnostician for an entire computer system. Its main task is to implement reasoning that bridges the gap between the very fine level of device diagnostics and coarse observations of system-level failure. Implemented under the EMYCIN framework, DART consists of 190 rules and 300 EMYCIN parameters. {electronics diagnosis, EMYCIN}

DIGITALIS ADVISOR [Silverman, 1975; Gorry et al., 1978] is an expert consulting

program that advises a physician on the proper dose of a heart drug called digitalis. The dosage calculation is subtle and complex, involving many facts about the patient's condition and treatment. Inferencing is goal-directed, and the domain is sufficiently narrow that solutions are developed rapidly. {medicine, backward chaining, uncertainty}

XPLAIN [Swartout, 1981; 1983] is a system that generates explanations for the advice given by the DIGITALIS ADVISOR. It is notable in that it does not merely produce explanations from the text of the rules that were executed, it has much deeper knowledge. Indeed, XPLAIN contains an automatic programming subsystem that generates the DIGITALIS ADVISOR: it can explain the advice because it built the advisor. {medicine, explanation, deep models, automatic programming}

The DIPMETER ADVISOR [Davis et al., 1981] is a system to assist in geological exploration. It infers subsurface structures based on data from an instrument known as a dipmeter. This task is difficult because the data are very sparse and noisy, with few mutual constraints. However, the search space is small, and there is less ambiguity than in domains such as speech understanding. Since the data features must be evaluated in isolation, deep knowledge of the underlying geological processes are needed. This knowledge is captured in production rule form. The productions are organized into five groups, corresponding to the major steps in the reasoning algorithm. Rule antecedents consist of simple constant comparisons, and working memory elements representing data and intermediate results are flat lists of constants. The developers estimate that a few hundred rules would be required for a complete version of the system. {resource exploration, deep models, uncertainty}

ELAS [Weiss et al., 1982] is an expert system to assist in oil exploration. Implemented in the EXPERT rule-based system framework, it interacts with the user to control Amoco proprietary software for well-log analysis and display. ELAS extends ideas of the classification systems such as CASNET, MYCIN, and PROSPECTOR, but it operates in real-time, keeps a model of the user, and is closely integrated with the Amoco programs. This integration is facilitated by having a common underlying language, Fortran. {resource exploration, expert user, EXPERT, Fortran}

GAMMA [Barstow, 1980] is an expert system that determines the composition of unknown substances by examining the gamma ray activation spectra resulting from neutron bombardment. Reasoning proceeds by backward-chaining from gamma ray detections to emissions, to unstable isotopes, to isotopes after bombardment, to isotopes in the original sample, to elements in the original sample. Formulae, tables, and rules at each level form a deep model capable of predicting hypotheses at the next lower level, eventually reaching predictions that can be compared with the observed spectra. Data are triples (*o*,*e*,*c*), where *o* represents an object such as an isotope at one of the 6 deductive levels, *e* is an estimate of concentration, and *c* is a chain encoding a path from an element in the original sample to the triple. An ad hoc hill-climbing algorithm is applied to an "interpretation measure" that tells how well a set of constituent *(element,concentration)* pairs accounts for the spectral peaks. {chemistry, backward chaining, deep models}

INTERNIST [Pople, 1977] is an expert system intended to cover the entire field of internal medicine. Disease knowledge is stored in an organ-based hierarchy, with relations between disease nodes denoted by four classes of weighted links: causal, associative, manifestation of, and invoked by. The search space is viewed as an AND/OR graph, with *constrictors* suggesting regions of the graph to which attention is confined ("jaundice means there is trouble with the liver"), and a *multi-problem generator* suggesting constellations of problems that can account for the observed symptoms. Processing evaluates competing hypotheses by calculating numerical scores that measure such qualities as goodness of fit and explanatory power. The highest scoring hypothesis is pursued, with new data (such as test results) causing a recomputation of all scores (possibly resulting in a shift of focus). Thus the quality of the scoring function is crucial to the performance of the system, and the hidden intelligence in this function is inaccessible for the generation of explanations. Although breadth of knowledge does not imply accurate diagnoses, certainly it is prerequisite. Thus it is interesting to note the growth of the INTERNIST knowledge base. INTERNIST-I contains information concerning 400 disease entities and 2000 manifestations, and executes in 3 to 7 cpu minutes in a PDP-10/Interlisp environment. INTERNIST-II covers about 80% of the diagnoses of internal medicine, and requires from 20 seconds to 2 minutes to perform a

diagnosis. CADUCEUS [Pople, 1981] is the successor of INTERNIST. As of 1982 it possessed 100,000 associations in a semantic network that includes 500 diseases and 350 manifestations, which represents nearly 85% of the potential diagnoses of internal medicine. {medicine, domain network, uncertainty}

MDX [Chandrasekaran et al., 1979; Chandrasekaran and Mittal, 1982] is a system to diagnose liver problems in the cholestasis syndrome. The system's expertise is derived from diagnostic structures that form a hierarchy reflecting the deep structure of knowledge in the field. In this organization, the representation of a concept calls on subconcepts much as a physician calls on specialists. Each concept has attached code that, given a diagnostic problem, first decides whether the problem lies within the scope of expertise of this subtree, and if so, decides which subconcepts to call on as specialists. If the problem is outside the scope of the subtree, the calling super-concept is advised where else to look. The authors assert that these diagnostic structures contain compiled knowledge that is intermediate between the extremes of a data base of patterns on one hand, and representations of deep knowledge (in whatever form) on the other, and that all the diagnostic problems that could be solved by deep knowledge can still be solved, but more efficiently. {medicine, domain network, uncertainty, deep models, scope of expertise}

PIP [Szolovits and Pauker, 1978] is an expert system to diagnose renal disease. The medical knowledge is stored in frame structures that represent possible disorders. The structure of a frame is as follows. The trigger slot has tests to be compared with observations about the patient to see if the frame's hypothesis should be considered. The findings slot accumulates additional evidence for a hypothesis already under consideration. Slots named is-sufficient, must-have, and must-not-have contain categorical tests to determine whether the hypothesis applies to the patient. The differential-diagnosis slot contains a list of tests to check alternatives to this hypothesis, while complementary hypotheses are linked through the caused-by, cause-of, complicated-by, complication-of, and associated-with slots. The score slot contains a complex numerical likelihood estimation function specialized to the hypothesis. One observation concerning PIP is that the scores are sensitive, so question-asking by the system tends to flitter from hypothesis to hypothesis as the

odds fluctuate, which is not reassuring to the user. Also, PIP doesn't know when to stop. It continues exploring additional less and less reasonable hypotheses, until none remain or every finding has already been examined. {medicine, frames, domain network, large rules, uncertainty}

PLANT/DS [Uhrik, 1982] is an expert system for the diagnosis of soybean diseases, using a combination of rules derived from human experts and from machine-induction on exemplary cases. The system operates in a domain in which the discriminations are very subtle, and in which diagnoses must be made from constellations of many weakly suggestive observations. The concomitant difficulty is that several weakly believed inferences may aggregate to indicate with near certainty an incorrect result. {plant disease, uncertainty, knowledge acquisition}

PUFF [Osborn et al., 1979] is an expert system for diagnosing pulmonary disease. Written in EMYCIN, it has 55 rules, and required fewer than 50 hours of human expert interaction and fewer than ten weeks of knowledge engineering time to construct [Feigenbaum, 1977]. It has a fixed order for exploring the diagnostic space (by simple backward-chaining), so it occasionally asks unreasonable questions in the context of previous answers. Nevertheless, it produces high quality diagnoses, and the system is in routine clinical use. {medicine, uncertainty, backward-chaining}

SACON [Bennett and Engelmore, 1979] is an expert system written in EMYCIN to advise on the operation of a large structural analysis program named MARC. MARC applies finite-element analysis techniques to the simulation of properties of structures such as aircraft wings, reactor pressure vessels, rocket motor casings, bridges, and buildings. The properties of interest include fatigue, responses under varying load, stability, and deflection. MARC is sufficiently rich in options that a year of experience is typically required to become a proficient user. SACON provides this experience on behalf of a less-seasoned user. First, it obtains from the user a description of the geometry, materials, loadings, and required accuracy for the structure to be analyzed. It then determines the analysis class into which the object falls, and recommends an appropriate analysis strategy to be used in MARC. A typical consultation to prepare for the analysis of an object with two

substructures, three loadings, and three load components requires about 25 minutes at an interactive terminal. SACON's knowledge resides in approximately 170 rules and 140 EMYCIN parameters, which were obtained in two months of a human expert's time, with two additional months for implementation and testing. The marginal time cost of new rules was found to be two hours, although the first 170 rules required four hours each. EMYCIN had previously only been used for medical systems; it proved to be effective in this domain as well. {structural analysis, expert user, EMYCIN}

4.4 Other Expert Systems

EL [Stallman and Sussman, 1977] is an expert system, written in the ARS language, to analyze the behavior of analog electronic circuits. Knowledge about circuits is represented by rules that act as demons, monitoring an associative relational database that models the circuit under analysis. The medium-priority demons apply electrical laws to make deductions. When no further deductions are possible, low-priority demons make or retract device-state assumptions ("method of assumed states"), which are checked by high-priority demons for consistency with the remainder of the circuit ("propagation of constraints"). If the state is found to be inconsistent, backtracking is taken at a point chosen to remove the conflict ("dependency-directed backtracking"). The device-state dependencies thus obtained, stored in an inheritance hierarchy, prevent the generation of circuit states containing previously discovered conflicts. When a new state is asserted, a decision tree of patterns selects demons to be enqueued for execution at appropriate priorities. For speed, there are separate databases to hold facts, demons, and dependencies. It is noted that although rules give local modularity, the overall structure of a rule-based system is quite rigid. {electronics diagnosis, demon, assumed states, propagation of constraints, dependency-directed backtracking, frames, partitioned rules}

HARPY [Lowerre and Reddy, 1980] is an expert system for speech understanding, solving the same problem as Hearsay-II. In HARPY, the syntax, lexical, and word juncture knowledge is compiled from context free production rules into a discrimination network representing all legal utterances in the domain. During

execution, a relatively simple interpreter then compares speech with this structure by beam search to find the best matching interpretation. This approach gives high speed, which permits search space pruning decisions to be postponed until larger partial solutions are developed and evaluated, which in turn leads to accuracy. For a 1000-word vocabulary, the discrimination network has 15,000 nodes and requires 13 hours on a DEC-10 (KL) to compile. {speech understanding, uncertainty, compiled rules}

PARADISE [Wilkins, 1981] is a chess-playing expert system that uses deep strategies rather than rapid iteration of brute-force search techniques. A production system organization was chosen to facilitate modification and extension of the knowledge base, despite the penalty in execution speed. In PARADISE, a knowledge source is a group of rules about some abstract concept, together with variables whose joint instantiation represent a specific instance of the abstract concept. Some productions participate in more than one knowledge source, and some concepts are not contained in any knowledge source. The rules discover patterns in chess positions, and post ideas to the database for consideration by other rules, ultimately generating chess plans that are then verified by a small tree search process. This problem domain is quite unlike that of systems such as MYCIN: the solution is not implicit in a codification of the current situation or in the knowledge base; rules must deal with higher concepts, linking them to create plans. However, this process is unlike robot planning, in that details must not be suppressed. Details are of the essence. Also, firing a production does not add a new fact to the system, because the rules do not make deductions. They produce ideas that may or may not be correct. There are no clear facts with probabilities that can be reasoned about, so productions must record their reasoning, not just their result, for inspection by subsequent rules. An example of the exceptionally well focused search performed by PARADISE is given by a chess problem that was solved through a search 19 ply deep. This search tree had only 109 nodes. {chess, deep knowledge, planning}

ROSS [Klahr and Faught, 1980] is a rule-oriented system for simulating military air battles. A rule representation was chosen because procedurally coded simulators have proven to be unintelligible, unmodifiable, not credible, and slow. Rules in ROSS specify object behaviors, and an underlying object-oriented message-passing

system conducts the activity. Each major event has a descriptor; these are linked to form scenarios. Trees of event descriptor chains represent "activities". These structures can be browsed, and serve as the basis for explanations of the occurrences during a simulation. The system has approximately 75 behavioral rules and 10 object types, and has simulated battles containing 250 objects. The authors feel the current system will not scale up without improvements in speed of execution, perhaps derived from parallel processing, abstraction, sampling, and focusing on user queries to avoid irrelevant processing. A reimplementation in ROSIE or a hybrid ROSIE/object-oriented language is under consideration so that the rule-based language will be more English-like. {military, domain network, ROSIE}

SAINT, SIN, [Moses, 1971] and MACSYMA [Martin and Fateman, 1971] form a progression of systems with expertise in symbolic mathematics. SAINT was viewed as an AI approach to symbolic integration. Further development led to SIN, which runs two orders of magnitude faster as a consequence of explicit tables of integrals, and special-purpose solution strategies for standard problem types. MACSYMA is an extension with broad expertise in differential and integral calculus and algebraic simplification. It is in routine use, and is more highly skilled than humans. {mathematics}

5 Conclusion

Numerous domains have proven fruitful for the development of expert systems. One principal area is medical diagnosis, and more generally, the diagnosis of systems, whether the human body, a nuclear reactor, a telephone system, or an electronic circuit. Other prominent areas include the ordering, configuration, and assembly of computer systems, evaluating geographical regions for oil and mineral deposits, and the elucidation of molecular structures.

A variety of effective approaches to the design of expert systems are known. Among these are the OPS systems of hundreds or thousands of small rules with a global working memory, the Hearsay-II style of systems having tens of knowledge sources observing and communicating through a blackboard, and the systems such as LOOPS that integrate rules with other programing paradigms.

From the work in this field, several issues are understood to a degree. It is known that large systems should assist in the process of knowledge acquisition. Means for accomplishing this include automatic checking to determine whether new information is consistent with the old (to the degree possible), and debugging of the knowledge in the context of erroneous deductions by the system. Successful knowledge representations have been found, including production rules for inferencing, frames to capture characteristics of entities, and networks for the representation of relationships. Guiding the process of deduction has been effected by static conflict resolution and by agendas under the control of meta-level knowledge sources. Efficient inferencing techniques have been developed for certain programming techniques, such as those employed by R1 and Prospector.

Many challenges remain. Among these are what might be termed the "software engineering" of expert systems, involving techniques for the design, implementation, and maintenance of large systems. Another open question is the appropriate granularity of knowledge representation. Effective applications of large knowledge sources and very small rules have been seen, but we lack general principles. In particular, current expert systems are individually hand-crafted (possibly within knowledge engineering frameworks); we lack general principles applicable to classes of problems, which would permit the construction of "generic experts".[8] Also, the means for capturing broad knowledge are not yet known; current expert systems are unaware of their limits, and draw incorrect conclusions when working outside the scope of their narrow expertise. Another area in infancy is the application of parallel hardware to obtain large increases in inferencing speed.

6 Acknowledgments

Thanks for insightful comments and helpful suggestions are due Richard Korf, Michael Lebowitz, and Salvatore Stolfo. This work was supported in part by an IBM Fellowship.

[8]This was pointed out by S. Stolfo in a private communication.

REFERENCES

Janice S. Aikins, "Prototypes and production rules: an approach to knowledge representation for hypothesis formation", *IJCAI-79, Proceedings of the sixth international joint conference on artificial intelligence*, August 20-23, pp. 1-3.

Janice S. Aikins, "Representation of control knowledge in expert systems", *AAAI-80, Proceedings of the first annual national conference on artificial intelligence*, August 18-21, pp. 121-123.

Janice S. Aikins, "Prototypical knowledge for expert systems", *Artificial Intelligence*, vol. 20, no. 2, February 1983, pp. 163-210.

Elizabeth Allen, "YAPS: a production rule system meets objects", *AAAI-83, Proceedings of the national conference on artificial intelligence*, pp. 5-7.

R. H. Anderson and J. J. Gillogly, "Rand intelligent terminal agent (RITA): design philosophy", Rand Corporation Technical Report R-1809-ARPA, Santa Monica, California, 1976.

Robert Balzer, Lee Erman, Philip London, Chuck Williams, "HEARSAY-III: a domain-independent framework for expert systems", *AAAI-80, Proceedings of the first annual national conference on artificial intelligence*, August 18-21, pp. 108-110.

David R. Barstow, "Exploiting a domain model in an expert spectral analysis program", *AAAI-80, Proceedings of the first annual national conference on artificial intelligence*, August 18-21, pp. 276-279.

Jeffrey A. Barnett, "Computational methods for a mathematical theory of evidence", *IJCAI-81, Proceedings of the seventh international joint conference on artificial intelligence*, August 24-28, pp. 868-875.

James S. Bennett and Robert S. Engelmore, "SACON: a knowledge-based consultant for structural analysis", *IJCAI-79, Proceedings of the sixth international joint conference on artificial intelligence*, August 20-23, pp. 47-49.

James S. Bennett and Clifford R. Hollander, "DART: an expert system for computer fault diagnosis", *IJCAI-81, Proceedings of the seventh international joint conference on artificial intelligence*, August 24-28, pp. 843-845.

John H. Boose, "Personal construct theory and the transfer of human expertise", *AAAI-84, Proceedings of the national conference on artificial intelligence*, August 6-10, 1984, pp. 27-33.

Bruce G. Buchanan, "New research on expert systems", *Machine Intelligence 10*, J. E. Hayes, Donald Michie, Y-H Pao (eds.), Halsted Press, New York, 1982, pp. 269-299.

Bruce G. Buchanan, Richard O. Duda, "Principles of rule-based expert systems", *Advances in computers*, vol. 22, Academic Press, 1983, pp. 163-216.

Bruce G. Buchanan and Edward A. Feigenbaum, "Dendral and Meta-Dendral: their applications dimension", *Readings in Artificial Intelligence*, Bonnie Lynn Webber and Nils J. Nilsson (eds.), Tioga, Palo Alto, 1981, pp. 313-322. Also *Artificial Intelligence*, vol. 11, no. 1-2, August 1978, pp. 5-24.

B. Chandrasekaran, F. Gomez, S. Mittal, and J. Smith, "An approach to medical diagnosis based on conceptual structures", *IJCAI-79, Proceedings of the sixth international joint conference on artificial intelligence*, August 20-23, pp. 134-142.

B. Chandrasekaran and Sanjay Mittal, "Deep versus compiled knowledge approaches to diagnostic problem-solving", *AAAI-82, Proceedings of the national conference on artificial intelligence*, August 18-20, pp. 349-354.

William J. Clancey, "The advantages of abstract control knowledge in expert system design", *AAAI-83, Proceedings of the national conference on artificial intelligence*, pp. 74-78.

William J. Clancey, "The epistemology of a rule-based expert system--a framework for explanation", *Artificial Intelligence*, vol. 20, no. 3, May 1983a, pp. 215-251.

William J. Clancey, "Classification problem solving", *AAAI-84, Proceedings of the national conference on artificial intelligence*, August 6-10, 1984, pp. 49-55.

William J. Clancey and Reed Letsinger, "NEOMYCIN: reconfiguring a rule-based expert system for application to teaching", *IJCAI-81, Proceedings of the seventh international joint conference on artificial intelligence*, August 24-28, pp. 829-836.

Brian L. Cohen, "A powerful and efficient structural pattern recognition system", *Artificial Intelligence*, vol. 9, no. 3, December 1977, pp. 223-255.

Randall Davis, "Meta-rules: reasoning about control", *Artificial Intelligence*, vol. 15, no. 3, December 1980, pp. 179-222.

Randall Davis, "Interactive transfer of expertise: acquisition of new inference rules", *Readings in Artificial Intelligence*, Bonnie Lynn Webber and Nils J Nilsson (eds.), Tioga, Palo Alto, 1981, pp. 410-428.

Randall Davis, Howard Austin, Ingrid Carlbom, Bud Frawley, Paul Pruchnik, Rich Sneiderman, J. A. Gilreath, "The DIPMETER ADVISOR: interpretation of geologic signals", *IJCAI-81, Proceedings of the seventh international joint conference on artificial intelligence*, August 24-28, pp. 846-849.

Randall Davis and Bruce G. Buchanan, "Meta-level knowledge: overview and applications", *IJCAI-77, Proceedings of the fifth international joint conference on artificial intelligence*, pp. 920-927.

Randall Davis and Bruce Buchanan, Edward Shortliffe, "Production rules as a representation for a knowledge-based consultation program", *Artificial Intelligence*, vol. 8, no. 1, February 1977, pp. 15-45.

Randall Davis and Jonathan King, "An overview of production systems", in *Machine Intelligence 8*, E. W. Elcock & D. Michie (eds.), Wiley, New York, 1976, pp. 300-332.

Johan de Kleer, "Choices without backtracking", *AAAI-84, Proceedings of the national conference on artificial intelligence*, August 6-10, pp. 79-85.

Richard Duda, John Gaschnig and Peter Hart, "Model design in the Prospector consultant system for mineral exploration", *Readings in Artificial Intelligence*, Bonnie Lynn Webber and Nils J. Nilsson (eds.), Tioga, Palo Alto, 1981, pp. 334-348. Also in Donald Michie (ed.), *Expert Systems in the Micro-electronic Age*, Edinburgh University Press, Edinburgh, Great Britain, 1979, pp. 153-167.

Richard O. Duda, Peter E. Hart and Nils J. Nilsson, "Subjective Bayesian methods for rule-based inference systems", *Readings in Artificial Intelligence*, Bonnie Lynn Webber and Nils J. Nilsson (eds.), Tioga, Palo Alto, 1981, pp. 192-199.

Richard O. Duda and Edward H. Shortliffe, "Expert systems research", *Science*, vol. 220, no. 4594, April 15, 1983, pp. 261-268.

Robert Engelmore, Allan Terry, "Structure and function of the CRYSALIS system", *IJCAI-79, Proceedings of the sixth international joint conference on artificial intelligence*, August 20-23, pp. 250-256.

Susan P. Ennis, "Expert systems: a user's perspective of some current tools", *AAAI-82, Proceedings of the national conference on artificial intelligence*, August 18-20, pp. 319-321.

Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, D. Raj Reddy, "The Hearsay-II speech-understanding system: integrating knowledge to resolve uncertainty", *Readings in Artificial Intelligence*, Bonnie Lynn Webber and Nils J. Nilsson (eds.), Tioga, Palo Alto, 1981, pp. 349-389.

J. Fain, D. Gorlin, F. Hayes-Roth, S. J. Rosenschein, H. Sowizral, and D. Waterman, "The ROSIE language reference manual, Rand Corporation Technical Report N-1647-ARPA, Santa Monica, California, 1981.

Edward A. Feigenbaum, "The art of artificial intelligence: I. themes and case studies of knowledge engineering", *IJCAI-77, Proceedings of the fifth international joint conference on artificial intelligence*, pp. 1014-1029. Also in *Proceedings of the NCC*, 1978. Also in Donald Michie (ed.), *Expert Systems in the Micro-electronic Age*, Edinburgh University Press, Edinburgh, Great Britain, 1979, pp. 3-25.

Richard Fikes and Tom Kehler, "The role of frame-based representation in reasoning", *Communications of the ACM*, vol. 28, no. 9, September 1985, pp. 904-920.

Charles L. Forgy, "OPS4 user's manual", Carnegie-Mellon Computer Science Technical report CMU-CS-79-132, 1979.

Charles L. Forgy, "OPS5 user's manual", Carnegie-Mellon Computer Science Technical Report, CMU-CS-81-135, July 1981.

Charles L. Forgy, "Rete: a fast algorithm for the many pattern/many object pattern match problem", *Artificial Intelligence*, vol. 19, no. 1, September 1982, pp. 17-37.

Charles L. Forgy, "Overview of OPS83", Carnegie-Mellon Computer Science Technical Report, June 19, 1983.

Charles L. Forgy and John McDermott, "OPS, a domain-independent production system language", *IJCAI-77, Proceedings of the fifth international joint conference on artificial intelligence*, pp. 933-939.

Kazuhiro Fuchi, "Aiming for knowledge information processing systems", in *Fifth Generation Computer Systems*, Tohru Moto-oka (ed.), North-Holland, Amsterdam-New York-Oxford, 1982, pp. 107-120. (This is the proceedings of the international conference on fifth generation computer systems, Tokyo, October 1981.)

John Gaschnig, "Application of the PROSPECTOR system to geological exploration problems", *Machine Intelligence 10*, J. E. Hayes, Donald Michie, Y-H Pao (eds.), Halsted Press, New York, 1982, pp. 301-323.

Michael R. Genesereth and Matthew L. Ginsberg, "Logic Programming", *Communications of the ACM*, vol. 28, no. 9, September 1985, pp. 933-941.

Michael P. Georgeff, "Procedural control in production systems", *Artificial Intelligence*, vol. 18, no. 2, March 1982, pp. 175-201.

Malik Ghallab, "Decision trees for optimizing pattern-matching algorithms in production systems", *IJCAI-81, Proceedings of the seventh international joint conference on artificial intelligence*, August 24-28, pp. 310-312.

G. A. Gorry, H. Silverman, and S. G. Pauker, "Capturing clinical expertise: a computer program that considers clinical responses to digitalis", *American Journal of Medicine*, vol. 64, March 1978, pp. 452-460.

J. H. Griesmer, S. H. Hong, M. Karnaugh, J. K. Kastner, M. I. Schor, R. L. Ennis, D. A. Klein, K. R. Milliken, H. M. VanWoerkom, "YES/MVS: a continuous real time expert system", *AAAI-84, Proceedings of the national conference on artificial intelligence*, August 6-10, pp. 130-136.

P. Haley, J. Kowalski, J. McDermott, R. McWhorter, "PTRANS: a rule-based management assistant", Carnegie-Mellon Computer Science Technical Report, 1983.

Frederick Hayes-Roth, "Rule-based Systems", *Communications of the ACM*, vol. 28, no. 9, September 1985, pp. 921-932.

Frederick Hayes-Roth and David J. Mostow, "An automatically compilable recognition network for structured patterns", *IJCAI-75, Proceedings of the fourth international joint conference on artificial intelligence*, September 3-8, pp. 246-251.

Frederick Hayes-Roth and D. A. Waterman, Douglas B. Lenat, "Principles of pattern-directed inference systems", in *Pattern-directed inference systems*, Waterman and Hayes-Roth (eds.), Academic Press, New York, 1978.

Frederick Hayes-Roth and D. A. Waterman, Douglas B. Lenat (eds.), "An overview of expert systems", in *Building expert systems*, Addison Wesley, Reading, Massachusetts, 1983, pp. 3-29.

Bruce K. Hillyer, David Elliot Shaw, "Execution of OPS5 production systems on a massively parallel machine", *Journal of parallel and distributed computing*, (accepted for publication).

Yumi Iwasaki and Peter Friedland, "SPEX: a second-generation experiment design system", *AAAI-82, Proceedings of the national conference on artificial intelligence*, August 18-20, pp. 341-344.

T. P. Kehler and G. D. Clemenson, "An application development system for expert systems", *Syst. Softw.*, vol. 3, no. 1, January 1984, pp. 212-224.

Philip Klahr and William S. Faught, "Knowledge-based simulation", *AAAI-80, Proceedings of the first annual national conference on artificial intelligence*, August 18-21, pp. 181-183.

Kurt Konolige, "An inference net compiler for the PROSPECTOR rule-based consultation system", *IJCAI-79, Proceedings of the sixth international joint conference on artificial intelligence*, August 20-23, pp. 487-489.

T. J. Kowalski and D. E. Thomas, "The VLSI design automation assistant: prototype system", *Proceedings of the 20th design automation conference*, June 1983.

John F. Lemmer and Stephen W. Barth, "Efficient minimum information updating for Bayesian inferencing in expert systems", *AAAI-82, Proceedings of the national conference on artificial intelligence*, August 18-20, pp. 424-427.

Douglas B. Lenat, Frederick Hayes-Roth, Philip Klahr, "Cognitive economy in artificial intelligence systems", *IJCAI-79, Proceedings of the sixth international joint*

conference on artificial intelligence, August 20-23, pp. 531-536. (Condensed from "Cognitive Economy", Memo HPP-79-15, Stanford, also issued as Rand Report N-1185.)

Douglas B. Lenat and John McDermott, "Less than general production system architectures", *IJCAI-77, Proceedings of the fifth international joint conference on artificial intelligence*, pp. 928-932.

Bob London and William J. Clancey, "Plan recognition strategies in student modeling: prediction and description", *AAAI-82, Proceedings of the national conference on artificial intelligence*, August 18-20, pp. 335-338.

B. T. Lowerre, and R. Reddy, "The HARPY speech understanding system", in *Trends in speech recognition*, W. A. Lea ed., Prentice-Hall, Englewood Cliffs NJ, 1980, Chapter 15.

W. A. Martin and R. J. Fateman, "The MACSYMA system", *Proceedings of the second symposium on symbolic and algebraic manipulation*, Los Angeles, 1971, pp. 59-75.

Drew McDermott, "ARBY: diagnosis with shallow causal models", *AAAI-82, Proceedings of the national conference on artificial intelligence*, August 18-20, pp. 370-372.

John McDermott, "R1: an expert in the computer systems domain", *AAAI-80, Proceedings of the first annual national conference on artificial intelligence*, August 18-21, pp. 269-271.

John McDermott, "R1: a rule-based configurer of computer systems", *Artificial Intelligence*, vol. 19, no. 1, September 1982, pp. 39-88.

John McDermott, "XSEL: a computer sales person's assistant", *Machine Intelligence 10*, J. E. Hayes, Donald Michie, Y-H Pao (eds.), Halsted Press, New York, 1982, pp. 325-337.

John McDermott, "Extracting knowledge from expert systems", *IJCAI-83, Proceedings of the eighth international joint conference on artificial intelligence*, pp. 100-107.

Kathy R. McKeown, Michael Wish, and Kevin Matthews, "Tailoring explanations for the user", in *IJCAI-81, Proceedings of the ninth international joint conference on artificial intelligence*, August 1985, pp. 794-798.

William van Melle, "A domain-independent production-rule system for consultation programs", *IJCAI-79, Proceedings of the sixth international joint conference on artificial intelligence*, August 20-23, pp. 923-925.

Jack Minker, "Issues in developing expert systems", *Proceedings of Logic Programming Workshop 1983*, June 26 - July 1, pp. 204-215.

Joel Moses, "Symbolic integration: the stormy decade", *Communications of the ACM*, vol. 14, no. 8, 1971, pp. 548-560.

Dana S. Nau, "Expert computer systems", *IEEE Computer Magazine*, vol. 16, no. 2, February 1983.

William R. Nelson, "REACTOR: an expert system for diagnosis and treatment of nuclear reactor accidents", *AAAI-82, Proceedings of the national conference on artificial intelligence*, August 18-20, pp. 296-301.

H. Penny Nii et al., *AI Magazine* vol. 3, no. 23, 1982.

H. Penny Nii and Nelleke Aiello, "AGE (attempt to generalize): a knowledge-based program for building knowledge-based programs", *IJCAI-79, Proceedings of the sixth international joint conference on artificial intelligence*, August 20-23, pp. 645-655.

H. Penny Nii and Edward A. Feigenbaum, "Rule based understanding of signals", *Proceedings of the conference on pattern-directed inference systems, 1977*, also in *Pattern-directed inference systems*, D. A. Waterman and F. Hayes-Roth, eds., Academic Press, New York, pp. 483-501.

J. Osborn et al., "Managing the data from respiratory measurements", *Medical Instrumentation*, vol. 13, no. 6, November 1979.

Ramesh S. Patil, Peter Szolovits, William B. Schwartz, "Information acquisition in diagnosis", *AAAI-82, Proceedings of the national conference on artificial intelligence*, August 18-20, pp. 345-348.

Judea Pearl, "Reverend Bayes on inference engines: a distributed hierarchical approach", *AAAI-82, Proceedings of the national conference on artificial intelligence*, August 18-20, pp. 133-136.

Harry E. Pople, Jr., "The formation of composite hypotheses in diagnostic problem solving an exercise in synthetic reasoning", *IJCAI-77, Proceedings of the fifth international joint conference on artificial intelligence*, pp. 1030-1037.

Harry E. Pople, Jr., "Heuristic methods for imposing structure on ill-structured problems: The structuring of medical diagnostics" in *Artificial intelligence in medicine*, P. Szolovits, ed., Westview Press, Boulder Colorado, 1981, pp. 119-185.

E. Post, "Formal reductions of the general combinatorial problem", *American Journal of Mathematics*, vol. 65, 1943, pp. 197-268.

J. R. Quinlan, "Consistency and plausible reasoning", *IJCAI-83, Proceedings of the eighth international joint conference on artificial intelligence*, pp. 138-144.

Rene Reboh, "Using a matcher to make an expert consultation system behave intelligently", *AAAI-80, Proceedings of the first annual national conference on artificial intelligence*, August 18-21, pp. 231-234.

Steven Rosenberg, "HPRL: A language for building expert systems", *IJCAI-83, Proceedings of the eighth international joint conference on artificial intelligence*, pp. 215-217.

Ron Sauers and Rick Walsh, "On the requirements of future expert systems", *IJCAI-83, Proceedings of the eighth international joint conference on artificial intelligence*, pp. 110-115.

H. Silverman, "A digitalis therapy advisor", MIT Project MAC technical report TR-143, 1975.

Edward H. Shortliffe, "Consultation systems for physicians: the role of artificial intelligence techniques", *Readings in Artificial Intelligence*, Bonnie Lynn Webber and Nils J. Nilsson (eds.), Tioga, Palo Alto, 1981.

Edward H. Shortliffe, A. Carlisle Scott, Miriam B. Bischoff, A. Bruce Campbell, William Van Melle, Charlotte D. Jacobs, "ONCOCIN: an expert system for oncology protocol management", *IJCAI-81, Proceedings of the seventh international joint conference on artificial intelligence*, August 24-28, pp. 876-881.

David E. Smith and Jan E. Clayton, "A frame-based production system architecture", *AAAI-80, Proceedings of the first annual national conference on artificial intelligence*, August 18-21, pp. 154-156.

Richard M. Stallman and Gerald J. Sussman, "Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis", *Artificial Intelligence*, vol. 9, no. 2, October 1977, pp. 135-196.

Mark Stefik, "Planning with Constraints (MOLGEN: Part 1)", "Planning and meta-planning (MOLGEN: Part 2)", *Artificial Intelligence*, vol. 14, no. 2, September 1980.

Mark Stefik, Jan Aikins, Robert Balzer, John Benoit, Lawrence Birnbaum, Frederick Hayes-Roth, Earl Sacerdoti, "The organization of expert systems, a tutorial", *Artificial Intelligence*, vol. 18, no. 2, March 1982, pp. 135-173.

Mark Stefik, Alan G. Bell, Daniel G. Bobrow, "Rule-oriented programming in loops", Xerox PARC Memo KB-VLSI-82-22, June 12, 1983.

Salvatore J. Stolfo and Daniel P. Miranker, "DADO: a parallel processor for expert systems", *Proceedings of the 1984 international conference on parallel processing*, August 21-24, 1984.

Salvatore J. Stolfo and David Elliot Shaw, "DADO: a tree-structured machine

architecture for production systems", *AAAI-82, Proceedings of the national conference on artificial intelligence*, August 18-20, pp. 242-246.

Salvatore J. Stolfo and Gregg T. Vesonder, "ACE: an expert system supporting analysis and management decision making", Columbia Computer Science, October 1982.

Thomas M. Strat, "Continuous belief functions for evidential reasoning", *AAAI-84, Proceedings of the national conference on artificial intelligence*, August 6-10, pp. 308-313.

William R. Swartout, "Explaining and justifying expert consulting programs", *IJCAI-81, Proceedings of the seventh international joint conference on artificial intelligence*, August 24-28, pp. 815-823.

William R. Swartout, "XPLAIN: a system for creating and explaining expert consulting programs", *Artificial Intelligence*, vol. 21, no. 3, September 1983, pp. 285-325.

Peter Szolovits and Stephen G. Pauker, "Categorical and probabilistic reasoning in medical diagnosis", *Artificial Intelligence*, vol. 11, no. 1-2, August 1978, pp. 115-144.

Donald E. Thomas, Charles Y. Hitchcock III, Thaddeus J. Kowalski, Jayanth V. Rajan, and Robert A. Walker, "Automatic data path synthesis", *IEEE Computer Magazine*, vol. 16, no. 12, December 1983, pp. 59-69.

Philip C. Treleaven and Isabel Gouveila Lima, "Japan's fifth-generation computer systems", *Computer*, vol. 15, no. 8, August 1982, pp. 79-88.

Alan Turing, "Computing machinery and intelligence", in Edward A. Feigenbaum and J. Feldman (eds.), *Computers and Thought*, McGraw-Hill, New York, 1963.

Carl T. Uhrik, "PLANT/ds revisited: non-homogeneous evaluation schema in expert systems", *AAAI-82, Proceedings of the national conference on artificial intelligence*, August 18-20, pp. 217-220.

D. A. Waterman, "User-oriented systems for capturing expertise: a rule-based approach", in Donald Michie (ed.), *Expert Systems in the Micro-electronic Age*, Edinburgh University Press, Edinburgh, Great Britain, 1979, pp. 26-34.

D. A. Waterman and F. Hayes-Roth (eds.), *Pattern-directed inference systems*, Academic Press, New York, 1978.

D. A. Waterman and Mark Peterson, "Rule-based models of legal expertise", *AAAI-80, Proceedings of the first annual national conference on artificial intelligence*, August 18-21, pp. 272-275.

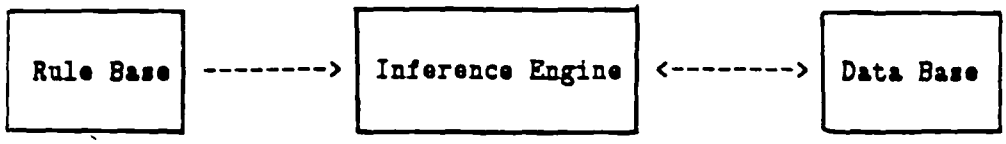
Sholom M. Weiss and Casimir A. Kulikowski, "EXPERT: a system for developing consultation models", *IJCAI-79, Proceedings of the sixth international joint conference on artificial intelligence*, August 20-23, pp. 942-947.

Sholom Weiss, Casimir Kulikowski, S. Amarel, A. Safir, "A model-based method for computer-aided medical decision-making", *Artificial Intelligence*, vol. 11, pp. 145-172, 1978.

Sholom M. Weiss, Casimir A. Kulikowski, Chidanand Apte', Michael Uschold, Jay Patchett, Robert Brigham, Belynda Spitzer, "Building expert systems for controlling complex programs", *AAAI-82, Proceedings of the national conference on artificial intelligence*, August 18-20, pp. 322-326.

David Wilkins, "Using patterns and plans in chess", *Readings in Artificial Intelligence*, Bonnie Lynn Webber and Nils J. Nilsson (eds.), Tioga, Palo Alto, 1981, pp. 390-409.

Lotfi A. Zadeh, "Commonsense knowledge representation based on fuzzy logic", *IEEE Computer Magazine*, vol. 16, no. 10, October 1983, pp. 61-65.



The rule base contains the system's reasoning knowledge for the problem domain. Its contents are usually obtained from human experts by knowledge engineers.

The data base contains facts modeling the problem state under solution. Facts specifying the initial state are obtained from the user of the expert system.

The inference engine is a program that applies the knowledge expressed in the rule base to the facts stored in the database in order to make deductions. The inference engine reads both the rule base and the knowledge base, and writes updated information into the data base as problem solution progresses.

Figure 1: Basic components of an AI production system.

```
(p sort-work                                  ; Production named sort-work
   (task    ↑taskname SORT)                   ; If current task is to sort
   (number  ↑value <x>    ↑used NO)          ;   and there is an unused number x
 - (number  ↑value < <x>  ↑used NO)          ;   but no smaller unused number
   (counter ↑value <n>)                      ;   and the output counter is n
  -->                                         ; Then
   (write <x>)                               ;   write number to output
   (modify 2 ↑used YES)                      ;   mark x as used, and
   (modify 3 ↑value (compute <n> + 1))       ;   increment the output counter.
)

(p sort-done                                  ; Production named sort-done
   (task    ↑taskname SORT)                   ; If current task is to sort
 - (number  ↑used NO)                        ;   but no unused number remains
   (counter ↑value <total>)                  ;   and the output counter is total
  -->                                         ; Then
   (write <total> items sorted)              ;   write the total number of items
   (remove 1)                                ;   and terminate the sorting task
)
```

This is a pair of rules written in the OPS5 language. The first rule is a complete description of a sort. It operates by repeatedly finding the smallest number, printing it, and discarding it. In addition, the first rule counts the number of items sorted. The second rule notices when the sort has finished, prints the total number of items sorted, and also removes from the working memory the context element that establishes the sorting task.

Figure 2: Example small-rule productions.

This example illustrates the slots of a frame-based knowledge source.

Knowledge-source name: Downshift control

Description:
The downshift control knowledge source examines information from sensors on an automobile and decides whether to shift the transmission to a lower gear.

Trigger Condition:
is a declarative indication that the antecedent should be executed every 1/10 second, and whenever the accelerator pedal is depressed further.

Screen Test:
is a small procedure that checks prerequisite conditions to determine whether the antecedent should be executed. An example of such a condition would be that the driver has not selected Neutral or Park for the transmission.

Antecedent:
is a procedure that collects sensor data such as the current gear, combustion chamber peak pressure, engine output torque, vehicle velocity, vehicle acceleration, accelerator pedal position, carburettor throttle position, and carburettor choke valve position. These data are added to the Private State History slot. The procedure also performs a simple calculation on the velocity, acceleration, and pedal position to obtain a prediction of whether the consequent would try to shift gears. The result of this calculation is stored in the Potential Action slot.

Potential Action:
is a simple declarative indication of the likely result of executing the consequent. The value of this slot is set by the antecedent, and is examined by the scheduler to assist in determining whether to allocate execution time to the consequent during this inferencing cycle.

Consequent:
is a procedure that performs a calculation based on the current sensor data collected by the antecedent, together with the private state information, to maximize an objective function such as: Maximize fuel economy subject to (1) a minimum acceptable acceleration depending on the difference between accelerator pedal position and current velocity, (2) suppression of oscillation between gears, (3) protection of the engine and drive train from overload or overspeed. The result of the computation is an updating of the private state information, together with the possible activation of a servo mechanism to change gear.

Private State History:
is a data structure to store the activity of the gear selection servo, and the history of sensor data over the past several time intervals. The antecedent and consequent store data here, and the consequent examines it.

Development History:
is notes indicating the development history for this knowledge source, the rationale for implementation conditions, and warnings concerning modification or use.

Figure 3: Example frame-based knowledge source.

This is an example of forward-chaining rule execution, showing a rule base and initial database, with a trace of six cycles of (match, select, act), and the final state of working memory.

```
Rule base:
  1.  A and B and C -> P
  2.  D and E -> P
  3.  F and G and H -> Q
  4.  P and Q -> X

Initial data base:
  A1  A2  A3  C1  D1  D2  E1  F1  G1  H1

Cycle 1.  Match rules with data, obtain rule instantiations:
            rule 2:  (D1, E1)
                     (D2, E1)
            rule 3:  (F1, G1, H1)
          Select, say, (F1, G1, H1).
          Execute, assert Q(F1+G1+H1) into the data base.

Cycle 2.  Match, (note that executed instantiations are not reused).
            rule 2:  (D1, E1)
                     (D2, E1)
          Select, say, (D2, E1).
          Execute, assert P(D2+E1).

Cycle 3.  Match.
            rule 2:  (D1, E1)
            rule 4:  (P(D2+E1), Q(F1+G1+H1)).
          Select, say, (P(D2+E1), Q(F1+G1+H1)).
          Execute, assert X(D2+E1+F1+G1+H1).

Cycle 4.  Match.
            rule 2:  (D1, E1)
          Select (D1, E1).
          Execute, assert P(D1+E1).

Cycle 5.  Match.
            rule 4:  (P(D1+E1), Q(F1+G1+H1)).
          Select (P(D1+E1), Q(F1+G1+H1)).
          Execute, assert X(D1+E1+F1+G1+H1).

Cycle 6.  Match, no new instantiations.
          Selection fails.
          Execution halts.

Final working memory:
  A1  A2  A3  C1  D1  D2  E1  F1  G1  H1  Q(F1+G1+H1)  P(D2+E1)
  X(D2+E1+F1+G1+H1)   P(D1+E1)   X(D1+E1+F1+G1+H1)
```

Figure 4: Example forward-chaining rule execution.

This example shows a backward-chaining execution of the same rule base as the previous figure. Given an initial goal, the example shows cycles of backchaining to subgoals, and backtracking when a subgoal fails.

```
Rule base:
  1.  A and B and C -> P
  2.  D and E -> P
  3.  F and G and H -> Q
  4.  P and Q -> X

Initial goal stack:  X

Cycle 1.  Backchain rule 4, new subgoals P, Q.
          Goal stack (P, Q, X(4)).

Cycle 2.  Backchain rule 1, new subgoals A, B, C.
          Goal stack (A, B, C, P(1), Q, X(4)).

Cycle 3.  No rules for A; ask user.  User provides A1.
          Goal A satisfied.
          Goal stack (B, C, P(1), Q, X(4)).

Cycle 4.  No rules for B, user doesn't provide a value.
          Goal P(1) fails; back up to previous state.
          Goal stack (P, Q, X(4)).

Cycle 5.  Backchain rule 2, new subgoals D, E.
          Goal stack (D, E, P(2), Q, X(4)).

Cycle 6.  No rules for D; ask user.  User provides D1.
          Goal D satisfied.
          Goal stack (E, P(2), Q, X(4)).

Cycle 7.  No rules for E; ask user.  User provides E1.
          Goal E satisfied.
          Goal P(2) satisfied.
          Goal stack (Q, X(4)).

Cycle 8.  Backchain rule 3, new subgoals F, G, H.
          Goal stack (F, G, H, Q(3), X(4)).

Cycle 9.  No rules for F; ask user.  User provides F1.
          Goal F satisfied.
          Goal stack (G, H, Q(3), X(4)).

Cycle 10. No rules for G; ask user.  User provides G1.
          Goal G satisfied.
          Goal stack (H, Q(3), X(4)).

Cycle 11. No rules for H; ask user.  User provides H1.
          Goal H satisfied.
          Goal Q(3) satisfied.
          Goal X(4) satisfied.
          Goal stack empty; execution terminates.
```

Figure 5: Example backward-chaining rule execution.

This example shows two rules and two input observations, with associated confidence figures. Techniques for combining uncertain evidence are described, and the evaluation of the rules is performed.

```
Rule 1:  transportation type sleigh
         power source reindeer
         cargo container sack
         cargo type toys
          --> driver Santa Claus (0.8)

Rule 2:  weight class obese
         beard color white
         beard length long
         eye emotion twinkle
         suit color red
          --> name Santa Claus (0.6)

Observation:
  transportation type sleigh 1.0        weight class obese 0.7
  power source reindeer 0.9             beard color white 0.8
  cargo container sack 0.9              beard length long 0.6
  cargo_type toys 0.7                   eye emotion twinkle 0.6
                                        suit color red 1.0

The following example of the calculation of a measure of belief for the
conclusion that the individual is Santa Claus, is based on the following
three simple techniques for combining evidence.

  1. The measure of belief of an antecedent (a conjunction of clauses) is
     the minimum of the measures of belief of the clauses.  This is the
     fuzzy set rule for conjunctions.  ('A chain is as weak as its weakest
     link.')  Note that this is symmetric with respect to the clauses.

  2. The measure of belief of a rule consequent is defined to be the product
     of the measure of belief of the antecedent and the belief factor of the
     rule.

  3. Evidence from consequents X and Y is combined as follows.
       Let P = belief(X),  Q = belief(Y).
       Define aggregate belief A = P + (1-P)Q.
     That is, the belief resulting from P is augmented by a fraction
     proportional to the belief of Q.  This can also be interpreted as 'A is
     true because P is true, or (if P is false) Q is true.'
     Note that A is symmetric in P and Q:  A = P + Q - PQ.


Evaluation for rule 1:
  Measure of antecedent = 0.7
  Measure of consequent = 0.56

Evaluation for rule 2:
  Measure of antecedent = 0.6
  Measure of consequent = 0.36

Aggregate belief that the individual is Santa Claus:
  0.56 + (0.44) x 0.36 = 0.56 + 0.16 = 0.72
```

Figure 6: Example reasoning with uncertainty.

Index