AN OVERVIEW OF EXPERT COMPUTER SYSTEMS

Dana S. Nau
Associate Professor

Computer Science Dept.
University of Md.
College Park, MD 20742
(301) 454-7932

ABSTRACT

The current popular attention being paid to expert computer systems has led to variety of misconceptions about their nature and capabilities. This paper is intended to "clear the air" by providing introductory tutorial information about expert computer systems. The paper describes their basic features, and points some of their advantages and disadvantages.

This work was supported by a Presidential Young Investigator award to Dana S. Nau, with matching funds provided by General Motors, IBM, and Martin Marietta.

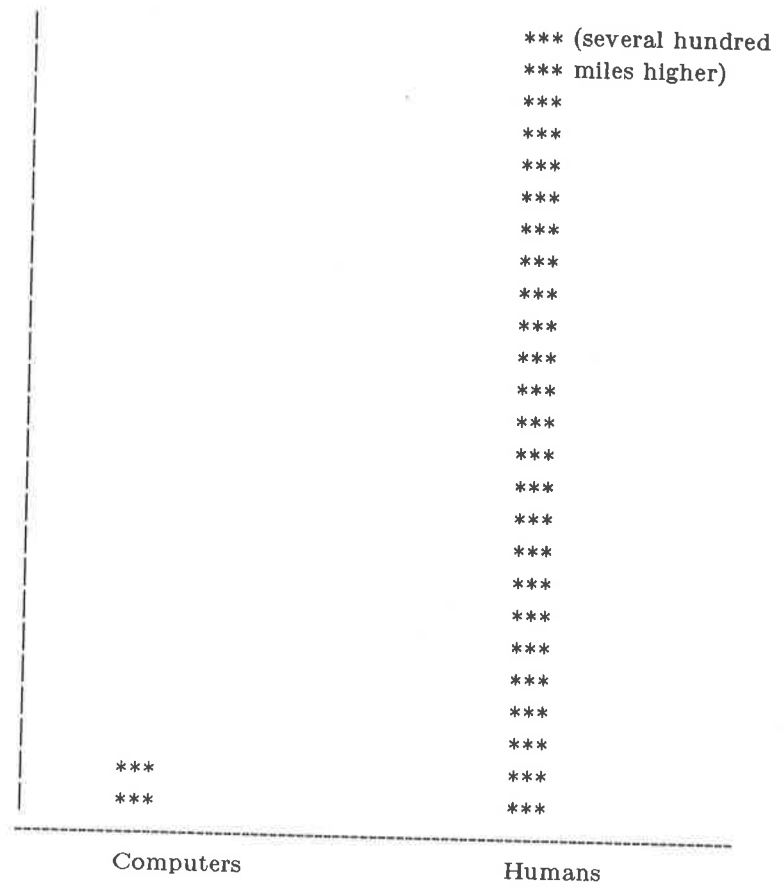


Figure 1.--Current intelligence in humans and machines.

1. INTRODUCTION

1.1. Artificial Intelligence

Artificial Intelligence (or AI) is that area of computer science whose goal is to develop computer systems which exhibit behavior which we would call intelligent. AI has been receiving a great deal of publicity lately, and there are a number of misconceptions about its capabilities and limitations. These misconceptions are manifested in reactions such as

1. "I think computers are getting *too* intelligent, and I'm afraid they might get out of control."
2. "Great! Where can I find an expert computer program to do *x* for me?"
3. "I don't think that computers can ever be intelligent, because they can only do exactly what we tell them to do."

This paper is intended to clear up such misconceptions, by providing a tutorial on a subfield of AI known as knowledge-based computing or expert computer systems.

AI includes several areas of research, some of which are listed below.

1. Research is being done on computer programs which can understand "natural" languages (i.e., human languages, as opposed to computer languages).
2. Work is being done on computer programs for perceptual tasks such as speech understanding and computer vision.
3. Robotics--which overlaps with research in the field of mechanical engineering--finds application in automated manufacturing and assembly. Work is also being done on integrating vision and robotics.
4. There are a number of problem solving methods--some general, some specific--which find application in many of the other areas of AI listed here. Work has been done on problem-solving techniques for game playing, automated program construction, theorem proving, trial-and-error search, and "common-sense" problem solving.
5. Computer learning is one of the most difficult tasks being explored. Various techniques are being explored for getting computers to learn how to recognize something, how to play a game, or how to synthesize concepts or techniques for use in problem solving.
6. Although much AI research is directed towards developing artificial intelligence using whatever techniques seem feasible, some researchers are doing research specifically on emulating the thought processes carried out by humans in solving various problems. This is called cognitive modeling.
7. An expert computer system is a computer system which uses a combination of domain-independent and domain-specific problem solving knowledge to achieve a high level of performance in an area which would normally require a human expert. Because of their potential applicability to a wide variety of problems, expert computer systems are currently of interest to researchers in many different areas of academia, industry, and government.

1.2. Two Myths About AI

Expert computer systems will be discussed in more detail in the following pages, but let us first discuss the three reactions mentioned at the beginning of the paper. These reactions arise from two common myths about AI.

The first myth is that we are already building highly intelligent computers and computer programs. This is simply not true. Although artificial intelligence has made some significant advances lately, it will be a long time before machine approach the thinking capabilities of human beings. The current state of affairs is illustrated in Figure 1.

The second myth is that computers can never be very intelligent, because they can "only do what we tell them to do." This is not true, either. There are ways to program computers to do things we don't explicitly tell them to do, and expert computer system techniques are one such way.

1.3. Expert Computer Systems

As an illustration of the range of applicability of expert computer systems, Table I is a (very incomplete) list of expert computer systems for various problem domains. In addition, several computer systems and languages have been developed to provide tools for creating expert systems for various problem domains. These systems include Age [27] [1], Ars [38], Emycin [22], Expert [43], Hearsay-III [12], KAS [19], KMS [31] [35], OPS5 [15], RLL [16], and Rosie [14] [19]. Such systems typically provide formats or languages for representing both procedural knowledge and declarative knowledge, control structures for manipulating this knowledge, and interfaces for interaction with human users.

Table I.--Some Existing Expert Systems.

System name	Application	Comments	Ref.
Ace	Prediction of problems in telephone lines	In use by AT&T	[39]
AQ11	Diagnosis of plant diseases	Can synthesize its own rules; has sometimes produced rules better than those proposed by experts	[7]
Casnet	Diagnosis and treatment of glaucoma		[44]
Centaur	Interpretation of pulmonary (lung) function tests	Successor to Puff	[2]
Dendral	Hypothesizing molecular structure from mass spectrograms	Has discovered new molecules	[4]
Dipmeter Advisor	Oil exploration	In use by Schlumberger	[8]
EL	Analyzing electrical circuits		[38]
(no name)	Diagnosis of abdominal pain	Outperforms physicians	[10]
E	Diagnosis of strokes	Implemented using KMS; outperforms physicians	[46]
Internist	Diagnosis in internal medicine	One of the largest knowledge bases of any expert computer system	[30]
Macsyma	Mathematical formula manipulation		[24]
MDX	Diagnosis of liver disorders		[6]
Molgen	Planning experiments in molecular genetics		[20]
Mycin	Diagnosis and treatment of infectious diseases		[9]
Prospector	Mineral exploration	Has discovered new ore deposits	[17]
Puff	Interpretation of pulmonary (lung) function tests	Implemented using Emycin	[29]
TIA	Treatment of transient ischemic attacks	Implemented using KMS	[36]
VM	Monitoring a patient's breathing after surgery		[13]
XCON	Configuring computers	Outperforms technicians; in use by Digital Equipment Corporation	[21]

Earlier we characterized an expert computer system as a computer system which achieves a high level of performance in an area that normally requires a human expert. From this description it may not be obvious why a computer system for medical diagnosis qualifies as an expert computer system and a computer system for a task such as payroll accounting does not. After all, both tasks, if done manually, require a highly trained human being. To understand the difference, one must examine the nature of human problem solving.

The problems that human beings can solve can be divided roughly into two classes. The first class consists of the problems that we can solve which we can also explain *how* to solve. Examples include adding columns of numbers, sorting arrays of numbers, and payroll accounting. If we can explain exactly how to solve a problem, an ordinary computer program can be written to solve the problem.

There are also problems which humans can solve but cannot explain how to solve. For example, each of us can recognize the faces of our various colleagues and friends--and yet we cannot describe them to others in such a way that others would be able to recognize them. As another example, physicians go to medical school for many years to learn how to do medical diagnoses, and yet at no time are they taught explicitly how to diagnose medical cases. Instead, they pick it up through experience--and they do internships after completing medical school to gain additional experience.

Problems which humans can solve but cannot explain how to solve are possible candidates for solution using AI techniques. One set of AI techniques which is sometimes useful are the techniques used in expert computer systems. These include ways to capture expert knowledge, put it into a "knowledge base," and make use of it in problem-solving.

1.4. Capturing Expert Knowledge

If an expert is given a specific problem in his or her domain of expertise, he or she will reach a specific conclusion. In some problem domains, the expert may be able to explain how that specific conclusion was reached. For example, the expert may produce an argument of the form

If *a* were true, then this would provide very strong evidence that *b* was true. Thus I checked to see whether *a* was true, and concluded that it probably was, since *c* and *d* were present.

From this argument one may extract the following two rules of inference:

IF *a* THEN conclude *b* (certainty=0.9)
IF *c* and *d* THEN conclude *a* (certainty=0.7)

where "very strong evidence" has been translated into "certainty=0.9" and "probably" has been translated into "certainty=0.7". The following is a similar kind of rule taken from an existing medical expert system [35]:

```
IF      serum K+ = low
        & myotonia = absent
        & serum T4 = high
THEN    paralysis type = thyrotoxic (0.8)
        & paralysis type = hypokalemic (0.2)
```

Useful knowledge cannot always be extracted from experts in this way. For example, talented computer programmers and chess players have worked for many years on computer programs to play chess--and yet the most successful chess-playing computer programs operate in ways very different from the way human chess players play chess [3] [37] [42]. Even when useful knowledge *can* be gotten from experts and encoded as rules, it still does not tell exactly what to do in a given situation, for many of the rules may be applicable at once. Thus some mechanism is necessary to decide which rules to apply and which paths of inference to follow, and this mechanism usually involves a trial-and-error search.

In conventional computer programs, there are two levels of operation: data and program. The program is an encoding of the knowledge necessary to solve problems in some domain, and the data is a representation of the specific problem to be solved. Because of the rules and the necessity of searching around for a solution, expert computer systems are organized in a rather different way.

In an expert computer system, there are three levels of operation: data, knowledge base, and control structure. The data is a representation of the specific problem to be solved. The knowledge base, which usually consists of rules such as those mentioned above, encodes problem-solving knowledge relevant to the domain of expertise of the expert computer system. The control structure is a computer program encoding some fairly general problem-solving knowledge: it decides which rules to apply if more than one rule is applicable at once, when (or whether) to backtrack if the path being explored seems unproductive, and so forth.

1.5. A Simple Example

The following example was developed by James Reggia at the University of Maryland. Suppose we have a personal robot to do our household chores. In looking through our house one morning, we find that it is infested with bugs. The task is to program the robot to get rid of the bugs.

The problem is that there are many different kinds of bugs, and the robot should take different actions depending on what bugs it finds. Ticks and Japanese Beetles, for example, are harmful and should be killed. Spiders, ladybugs, and crickets are benign, and should simply be moved outside. Praying mantises, are beneficial bugs to have in one's garden, because they will eat other bugs which might hurt the plants. Thus they should be moved to the solarium and encouraged to stay there. Table II gives a simple taxonomy of the bugs mentioned above: The following rules can be used to tell what class a bug is in (arachnid or insect):

C1. IF antennae=0 & legs=8 THEN class=arachnid
C2. IF wings=0 THEN class=arachnid
C3. IF antennae=2 & legs=6 THEN class=insect
C4. IF wings≠0 THEN class=insect

If a bug turns out to be an insect, the following rules can be used to tell which type of insect it is (beetle or orthoptera):

T1. IF class=insect & size=small & shape=round THEN type=beetle
T2. IF class=insect & size≠small & shape=elongated THEN type=orthoptera

The following rules can be used to determine the name of the bug:

N1. IF class=arachnid & leg-length=long THEN name=spider
N2. IF class=arachnid & leg-length=short THEN name=tick
N3. IF type=beetle & color=orange-and-black THEN name=ladybug
N4. IF type=beetle & color=green-and-black THEN name=japanese-beetle
N5. IF type=orthoptera & color=black THEN name=cricket
N6. IF type=orthoptera & color=green & size=large THEN name=praying-mantis

Once the identity of the bug has been established, the following rules tell what actions to take:

Table II.--A simple taxonomy.

Class	Type	Name	Characteristics
Arachnid		Spider	benign
		Tick	harmful
Insect	Beetle	Ladybug	benign
		Japanese Beetle	harmful
	Orthoptera	Cricket	benign
		Praying Mantis	beneficial

```
A1.   IF (name=spider | name=cricket | name=ladybug) & location≠outside
      THEN first-action=grasp-bug
           & second-action=move-to-outside
           & third-action=release-bug

A2.   IF name=japanese-beetleg | name=tick
      THEN first-action=swat-bug
           & second-action=grasp-bug
           & third-action=put-in-jar

A3.   IF name=praying-mantis & location≠solarium
      THEN first-action=grasp-bug
           & second-action=move-to-solarium
           & third-action=release-bug

A4.   IF name=praying-mantis & location=solarium
      THEN first-action=get-bug-from-jar
           & second-action=release-bug
           & third-action=none
```

The control strategy, which we call Handle-Bugs, is described below. Handle-Bugs uses what is called a goal-driven (or top-down), depth-first (or backtracking) problem-reduction approach:

Step 1. Let G be the goal Handle-Bugs is trying to achieve, and let S be the set of all rules which might be capable of satisfying G. Let R be the first rule in S.

Note: Each clause in the antecedent (the "IF" part) of R determines a new problem to be solved. For example, in order to determine whether "class=arachnid" is satisfied in rule N1, one must solve the problem of finding out the bug's class. Such a problem is called a *subproblem* or *subgoal.*

Step 2. Let A be the antecedent of R. For each clause C of A, invoke Handle-Bugs recursively with the corresponding problem P as its goal; and if it is found that A cannot be satisfied, then go to Step 3. Otherwise, once A has been satisfied, return from Handle-Bugs, with the returned value being the consequent (the "THEN" part) of R.

Step 3. If there are any rules left in G, then let R be the next one. Otherwise, return from Handle-Bugs, with the returned value being "failure."

Suppose the robot finds a bug. Then the following actions occur.

1. Handle-Bugs is invoked, with the goal of deciding what action to take. There are four rules that might tell what action to take: A1, A2, A3, and A4.
2. First A1 is tried. To satisfy its antecedent, it is necessary to solve two subproblems: finding out the bug's name and the robot's location. The first subgoal is to find out the bug's name. There are six rules that might tell the name: N1 through N6.
3. First N1 is tried. Its antecedent requires finding out the bug's class and its leg length. First the bug's class is checked. There are four rules which might tell the class: C1 through C4.
4. C1 is tried. Its antecedent requires finding out the number of antennae and the number of legs. First subgoal the number of antennae is checked. Suppose it is 2. Then the antecedent of C1 cannot be satisfied, since it requires that the number of antennae be 0. Thus C1 is inapplicable.
5. C2 is tried. Its antecedent requires finding out the number of wings the bug has. Suppose the bug has 2 wings. Then the antecedent of C2 is not satisfied, since it requires that the number of wings be 0. Thus C2 is inapplicable.
6. C3 is tried. Its antecedent requires finding out the number of antennae and the number of legs. The number of antennae is already known to be 2. The antecedent is satisfied so far, so the number of legs is checked. Suppose the number is 6. Then the antecedent for C3 is totally satisfied.

7. C3 concludes that the bug is an insect. This solves the subgoal (see #3 above) of finding out the bug's class.

8. The antecedent of N1 cannot be satisfied since it requires that the bug be an arachnid. Thus N1 is inapplicable

9. Rule N2 is tried. Its antecedent requires finding out the bug's class and its leg length. The bug's class is already known to be "insect." This means that N2 is inapplicable since it requires that the bug be an arachnid.

10. Rule N3 is tried. Its antecedent requires knowing the bug's type and its color. First the bug's type is checked. Two rules are applicable: T1 and T2.

11. T1 is tried. Its antecedent requires knowing the bug's class, size, and shape. The bug's class is already known to be "insect." Suppose the size is small and the shape is round. Then the antecedent to T1 is satisfied.

12. T1 concludes that the bug is a beetle. This satisfies the subgoal (see #10 above) of finding out the bug's type.

13. The antecedent of N3 is satisfied so far, so work is begun on the second subgoal relevant to N3: finding out the bug's color. Suppose that the color is orange-and-black. Then the antecedent of N3 is totally satisfied.

14. N3 concludes that the bug is a ladybug. This satisfies the subgoal (see #2 above) of finding out the bug's name.

15. So far, the antecedent of A1 is satisfied, so the second subgoal relevant to A1 is investigated: finding out the robot's location. If we suppose that the robot is inside, then the antecedent of A1 is totally satisfied.

16. A1 concludes that the actions to be performed are to pick up the bug, go outside, and release the bug. The ultimate goal is solved, so Handle-Bugs terminates.

This example illustrates the interplay among the three levels of an expert system. In this example, the data consisted of facts such as "color = orange-and-black" or "type = beetle". The knowledge base consisted of the various rules. Often several of them were applicable at once, and the control strategy used a goal-directed backtracking search in order to choose which rules to apply.

There are other types of knowledge bases and control strategies, but the reader should now have a basic idea how some expert computer systems work. For more information, the reader is referred to [11] [25] [26] [28] [45].

2. SUMMARY AND CONCLUSIONS

Expert systems are commonly defined as computer systems which use problem-specific problem solving knowledge to achieve high levels of performance in fields which would normally require human experts. It may not be clear from this definition what distinguishes such a system from an ordinary applications program. Certainly, applications programs make use of specialized problem-solving knowledge, and many of them reach high levels of performance. Probably the main differences are the following:

1. In expert systems, the domain-dependent problem solving information appears explicitly in a *knowledge base* rather than appearing only implicitly as part of the coding of the program. This knowledge base is manipulated by a separate *control structure*, which uses domain-independent techniques to search for and elaborate upon various possible solutions to the problem.

2. Just as with applications programs, expert systems cannot be developed for a problem without the availability of human experts who know how to solve the problem. But expert systems can sometimes be developed in problem domains where the human experts cannot explain how they solve the problems they solve.

During the last several years, expert system techniques have begun to find a number of applications outside of artificial intelligence research laboratories. Dendral has been used by university and industrial chemists on a number of problems. R1 saves Digital Equipment Corporation several million dollars each year as an aid in installing computer systems. AT&T's Ace is routinely used as an aid in telephone line

maintenance. And Prospector has discovered new ore deposits [5]. For the success of expert computer systems in other applications, progress will be required in several areas.

One problem is the amount of effort it takes to build an expert system: some expert systems have taken as many as ten to twenty-five man-years to build, and have cost as much as one to two million dollars. One reason for this is the lack of software tools for implementing expert computer systems. Progress is currently being made in this direction with tools for building expert systems (such as Age, Art, Emycin, Expert, Hearsay-III, KEE, KMS, Loops, and OPS-5).

A second problem is the amount of time necessary to take the knowledge from an expert in some problem domain and encode it in a knowledge base. This is partly because of a lack of adequate tools for this task, and partly because of the large gaps that still remain in our understanding of human problem solving. Some of the knowledge that an expert uses to solve a problem often cannot be made consciously accessible to the expert or to others without a great deal of effort. This has been one of the motivations for some of the recent research on machine learning [23].

A third issue is since expert systems have until recently been largely experimental, it has not been necessary to design such systems for long-term maintenance or to construct them to be "friendly" to a community of users who may not have a sophisticated knowledge of computers. For example, users are less likely to believe the conclusions reached by an expert system unless it can justify or explain its conclusions in an understandable and convincing way [41]. Although a number of researchers are doing work on justification and explanation facilities in expert computer systems [9] [40] [18] [34] [32] [33], more attention will have to be paid to such "real-world details" in order to develop useful expert systems for real-world problems.

Fourth, expert systems are currently something of a fad, and expert computer system technology is being oversold. As an example of this, a few years ago a well-known AI research laboratory sent its managerial personnel to a short course being given by a company on the West Coast which gives short courses on expert systems. The company also was marketing expert systems, and pointed out the features of these systems so enthusiastically that when the managerial people came back from the course, the researchers found it necessary to disabuse the managers of the notion that all the major problems of expert systems had already been solved.

Such hyperbole about expert systems has led many potential users to have overly optimistic expectations about the potential applicability, ease of use, and level of performance of expert systems. Because of the current glamorous image of expert computer systems, some people are using the term "expert system" indiscriminately to describe relatively conventional computer programs, and others are attempting to build expert systems for tasks which could perhaps better be solved using conventional techniques. Expert systems are potentially useful for a variety of problems--but unless potential users take a more cautious view of the potential of expert systems, they run the risk of disappointment with the performance of the systems they buy or build.

3. REFERENCES

[1] Aiello, N. and Nii, H., Building a Knowledge-Based System with AGE, Tech. Memo HPP-79-3, Computer Science Dept., Stanford University, 1979.

[2] Aikins, J. S., Prototypes and Production Rules: A Knowledge Representation for Computer Consultations, Ph.D. Dissertation, Tech. Report STAN-CS-80-814, Dept. of Computer Science, Stanford University, August 1980.

[3] Biermann, A. W., Theoretical Issues Related to Computer Game Playing Programs, *Personal Computing*, pp. 86-88, Sept. 1978.

[4] Buchanan, B. G. and Feigenbaum, E. A., Dendral and Meta-Dendral: Their Applications Dimension, *Artificial Intelligence* **11**, pp. 5-24, 1978.

[5] Campbell, A. N., Hollister, V. F., Duda, R. O., and Hart, P. E., Recognition of a Hidden Mineral Deposit by an Artificial Intelligence Program, *Science* **217**, 3, pp. 927-929, Sept. 1982.

[6] Chandrasekaran, B., Gomez, F., Mittal, S., and Smith, J., An Approach to Medical Diagnosis Based on Conceptual Structures, *Proc. Sixth Internat. Joint Conf. Artif. Intelligence,*, Tokyo, pp. 134-142, Aug. 1979.

[7] Chilausky, R., Jacobsen, B., and Michalski, R. S., An Application of Variable-Valued Logic to Inductive Learning of Plant Disease Diagnostic Rules, *Proc. Sixth Annual Internat. Symp. Multi-Valued Logic,*, Utah, 1976.

[8] Davis, R., Austin, H., Carlbom, I., Frawley, B., Pruchnik, P., Sneiderman, R., and Gilreath, J. A., The Dipmeter Advisor: Interpretation of Geological Signals, *Proc. Seventh Internat. Joint Conf. Artif. Intel.*, Aug. 1981.

[9] Davis, R., Buchanan, B., and Shortliffe, E., Production Rules as a Representation for a Knowledge-Based Consultation Program, *Artificial Intelligence* **8**, 1, pp. 15-45, 1977.

[10] deDombal, F., Computer Assisted Diagnosis of Abdominal Pain, *Advances in Medical Computing*, ed. J. Mitchell, New York, pp. 10-19, Churchill-Livingston, 1975.

[11] Duda, R. O. and Gaschnig, J. G., Knowledge-Based Systems Come of Age, *Byte*, pp. 238-281, Sept. 1981.

[12] Erman, L. D., London, P., and Fickas, S. F., The Design and an Example Use of HEARSAY-III, *Proc. Seventh Internat. Joint Conf. Artificial Intelligence*, pp. 409-415, 1981.

[13] Fagan, J. M., VM: Representing Time-Dependent Relations in a Medical Setting, Ph.D. Dissertation, Computer Science Dept., Stanford University, Stanford, CA, June 1980.

[14] Fain, J., Hayes-Roth, F., Sowizral, H., and Waterman, D., Programming in ROSIE: An introduction by means of examples, Tech. Report N-1646-ARPA, Rand Corp., Santa Monica, CA, 1982.

[15] Forgy, C. L., The OPS5 User's Manual, Tech. Report CMU-CS-81-135, Computer Sci. Dept., Carnegie-Mellon University, 1980.

[16] Greiner, R. and Lenat, D., A Representation Language Language, *Proc. First Annual National Conf. Artif. Intelligence*, 1980.

[17] Hart, P. E., Duda, R. O., and Einaudi, M. T., A Computer-Based Consultation System for Mineral Exploration, SRI International, Menlo Park, CA, 1978.

[18] Hasling, D. W., Clancey, W. J., and Rennels, G., Strategic Explanations for a Diagnostic Consultation System, *Internat. Jour. Man-Machine Studies* **20**, pp. 3-19, 1984.

[19] Hayes-Roth, F., Waterman, D. A., and Lenat, D. B., *Building Expert Systems*, Addison-Wesley, 1983.

[20] Martin, N., Friedland, P., King, J., and Stefik, M. J., Knowledge-Base Management for Experiment Planning in Molecular Genetics, *Proc. Fifth Internat. Joint Conf. Artif. Intell.*, pp. 882-887, 1977.

[21] McDermott, J. and Steele, B., Extending a Knowledge-Based System to Deal with Ad Hoc Constraints, *Proc. Seventh Internat. Joint Conf. Artif. Intel.*, pp. 824-828, Aug. 1981.

[22] Melle, W. van, A Domain-Independent Production Rule System for Consultation Programs, *Proc. Sixth Internat. Joint Conf. Artif. Intell.*, 1979.

[23] Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., *Machine Learning: An Artificial Intelligence Approach*, Tioga Publishing Company, Palo Alto, CA, 1983.

[24] Moses, J., Symbolic Integration: The Stormy Decade, *CACM* **14**, 8, pp. 548-560, 1971.

[25] Nau, D. S., Expert Computer Systems, *Computer* **16**, 2, pp. 63-85, Feb. 1983.

[26] Nau, D. S. and Reggia, J. A., Relationships between Abductive and Deductive Inference in Knowledge-Based Diagnostic Problem Solving, Proc. First Internat. Workshop Expert Database Systems, Kiawah, Island, SC, Oct. 1984.

[27] Nii, H. P. and Aiello, N., AGE (Attempt to Generalize): a Knowledge-Based Program for Building Knowledge-Based Programs, *Proc. Sixth Internat. Joint Conf. Artif. Intell.*, pp. 645-655, 1979.

[28] Nilsson, N. J., *Principles of Artificial Intelligence*, Tioga, Palo Alto, 1980.

[29] Osborn, J., Fagan, L., Fallat, R., McClung, D., and Mitchell, R., Managing the Data from Respiratory Measurements, *Medical instrumentation* **13**, 6, Nov. 1979.

[30] Pople, H. E., The Formation of Composite Hypotheses in Diagnostic Problem Solving: an Exercise in Synthetic Reasoning, *Proc. Fifth Internat. Joint Conf. Artif. Intell.*, pp. 1030-1037, 1977.

[31] Reggia, J. A., Knowledge-Based Decision Support Systems: Development through KMS, Ph.D. Dissertation, Tech. Report TR-1121, Computer Sci. Dept., Univ. of Maryland, Oct. 1981.

[32] Reggia, J. A., Perricone, B., Nau, D. S., and Peng, Y., Answer Justification in Abductive Expert Systems--Part II: Supporting Plausible Justifications, *IEEE Trans. Biomedical Engineering* **BME-32**, 4, pp. 268-272, April 1985.

[33] Reggia, J. A., Perricone, B., Nau, D. S., and Peng, Y., Answer Justification in Abductive Expert Systems--Part I: Abductive Inference and Its Justification, *IEEE Trans. Biomedical Engineering* **BME-32**, 4, pp. 263-267, April 1985.

[34] Reggia, J. A. and Perricone, B. T., Answer Justification in Medical Decision Support Systems Based on Bayesian Classification, Submitted for publication, University of Maryland, College Park, MD, 1983.

[35] Reggia, J. A., Pula, T. P., Price, T. R., and Perricone, B. T., Towards an Intelligent Textbook of Neurology, *Proc. Fourth Annual Symp. Computer Applications in Medical Care*, Washington, DC, pp. 190-199, Nov. 1980.

[36] Reggia, J. A., Tabb, D. R., Price, T. R., Banko, M., and Hebel, R., Computer-Aided Assessment of Transient Ischemic Attacks: A Clinical Evaluation, *Archives of Neurology*, 1984. To appear.

[37] Robinson, A. L., Tournament Competition Fuels Computer Chess, *Science* **204**, pp. 1396-1398, 1979.

[38] Stallman, R. M. and Sussman, G. J., Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis, *Artificial Intelligence* **9**, pp. 135-196, 1977.

[39] Stolfo, S. J. and Vesonder, G. T., ACE: An Expert System Supporting Analysis and Management Decision Making, Tech. Report, Dept. of Computer Science, Columbia University, 1983.

[40] Swartout, W. R., Explaining and Justifying Expert Consulting Programs, *Proc. Seventh Internat. Joint Conf. Artificial Intelligence*, pp. 815-822, August 1981.

[41] Teach, R. and Shortliffe, E., An Analysis of Physician Attitudes Regarding Computer-Based Clinical Consultation Systems, *Computers and Biomedical Research* **14**, pp. 542-558, 1981.

[42] Truscott, T. R., Minimum Variance Tree Searching, *Proc. First Internat. Symposium on Policy Analysis and Information Systems*, Durham, NC, pp. 203-209, June 1979.

[43] Weiss, S. M. and Kulikowski, C. A., EXPERT: a System for Developing Consultation Models, *Proc. Sixth Internat. Joint Conf. Artif. Intell.*, pp. 942-947, 1979.

[44] Weiss, S. M., Kulikowski, C. A., Amarel, S., and Safir, A., A Model-Based Method for Computer-Aided Medical Decision-Making, *Artificial Intelligence* **11**, 2, pp. 145-172, 1978.

[45] Winston, P. H., *Artificial Intelligence.*, Reading, MA, Addison-Wesley, 1977.

[46] Zagoria, R. J and Reggia, J. A., Transferability of Medical Decision Support Systems Based on Bayesian Classification, *Medical Decision Making* **3**, 1983.

Artificial intelligence is no longer science theory. A variety of "thinking" systems are out of the laboratory and successfully solving problems using AI knowledge-representation techniques.

SPECIAL FEATURE

Expert Computer Systems

Dana S. Nau, University of Maryland

Only recently has artificial intelligence advanced to the point that AI projects are accomplishing practical results. Most of these results can be attributed to the design and use of *expert systems*, problem-solving computer programs that can reach a level of performance comparable to that of a human expert in some specialized problem domain.

What distinguishes such a system from an ordinary applications program is not readily apparent in this definition. Certainly, applications programs make use of specialized problem-solving knowledge, and many of them reach high levels of performance. Probably the main difference is that in most expert systems, the model of problem-solving in the application domain is explicitly in view as a separate entity or *knowledge base* rather than appearing only implicitly as part of the coding of the program. This knowledge base is manipulated by a separate, clearly identifiable control strategy. Such a system architecture provides a convenient way to construct sophisticated problem-solving tools for many different domains.

Expert systems have been developed for a number of different problem domains, and Table 1 lists only a few of these systems. In addition, several computer systems and languages are being developed to provide tools for creating expert systems, including AGE[1], ARS,[2] Emycin,[3] Expert,[4] KMS,[5] and OPS.[6] Such systems typically provide formats or languages for representing both procedural knowledge and declarative knowledge, control structures for manipulating this knowledge, and user interfaces.

Ordinary computer programs organize knowledge on two levels: data and program. Most expert computer systems, however, organize knowledge on three levels: data, knowledge base, and control. Computers organized in this way are often called knowledge-based systems.

On the data level is declarative knowledge about the particular problem being solved and the current state of affairs in the attempt to solve the problem. On the knowledge-base level is knowledge specific to the particular kind of problem that the system is set up to solve. This knowledge is used by the system in reasoning about the problem and is often given in the form of operators, or "pattern-invoked programs." One, many, or no operators may be applicable to the problem at any one time. If applied, an operator produces changes in the data. In the control structure is a computer program that makes decisions about how to use the specific problem-solving knowledge. Decisions are made, for example, about which operators to apply and how to apply them.

This article discusses the techniques used in expert systems on each of these levels. Since these systems use a combination of AI problem-solving and knowledge-representation techniques, information on these areas is also included.

Table 1.
Some existing expert systems.

SYSTEM	EXPERTISE
AQ11[7]	Diagnosis of plant diseases
Casnet[8]	Medical consulting
Dendral[9]	Hypothesizing molecular structure from mass spectrograms
Dipmeter Advisor[10]	Oil exploration
EL[2]	Analyzing electrical circuits
Internist[11]	Medical consulting
KMS[5]	Medical consulting
Macsyma[12]	Mathematical formula manipulation
MDX[13]	Medical consulting
Molgen[14]	Planning DNA experiments
Mycin[15]	Medical consulting
Prospector[16]	Mineral exploration
Puff[17]	Medical consulting
R1[18]	Computer configuration

The data level: Representing declarative knowledge

One well-known way to represent declarative knowledge is by means of formulas in first-order predicate logic.* Simple declarative facts can often be represented as instantiated predicates. For example, *John gave Mary a book* can for some purposes be adequately represented by *GIVE (John, Mary, book)*. More complicated statements may require a more complicated representation as in the use of

$$(x)\ (y)\ (z)\ (R(x,y)\ \&\ R(y,z) \rightarrow R(x,z))$$

for the statement that the relation R is transitive.

Another way of representing declarative knowledge is in terms of frames.[20-22] Frames are data structures in which all knowledge about a particular object or event is stored together. Such a representation cannot represent any more concepts than first-order predicate logic can, but the organization of knowledge can be useful for modularity and accessibility of the knowledge. In addition, frame systems often allow ways to specify default values for pieces of information about an object when that information is not explicitly given.

Many different variants have been proposed for frame-based knowledge representation, but most of them include the idea of having different types of frames for different types of objects, with fields or *slots* in each frame to contain the information relevant to that type of frame. For example, a frame for a book might be a data structure that has slots for the author, title, and publication date of the book, as well as for the number of pages and color of the cover. To describe a particular book, a copy of this book frame would be created, and the slots would be filled in with the information about the particular book being described.

Semantic networks (or semantic nets) are a third way to represent declarative knowledge. They are like frames in the sense that knowledge is organized around the objects being described, but here the objects are represented by nodes in a graph and the relations among them are represented by labeled arcs.

Example: Representing a Set of Related Facts

Consider the following set of facts.

Bill took the book from Margaret.
Bill is a professor.
Margaret is a doctor.
Margaret lives in Akron, Ohio.

These facts, and some related facts, can be directly represented in first-order predicate logic as

TAKE(Bill, Margaret, book)
OCCUPATION(Bill, professor)
OCCUPATION(Margaret, doctor)
ADDRESS(Margaret, Akron-Ohio)
PERSON(Bill)
PERSON(Margaret)
OBJECT(book)
PROFESSION(professor)
PROFESSION(doctor)

*The techniques and discussion presented in this section are by no means complete. For further details on knowledge representation, see Mylopoulos.[19]

To put this information into frames, we first need to decide what kinds of frames to use. Schank[23,24] has developed a "theory of conceptual dependency" that, among other things, attempts to represent most events in terms of a small number of primitive actions. Each primitive action may be represented by a single kind of frame.

For example, Schank's theory casts "take" and "give" as two examples of the same phenomenon: a transfer of possession. The frame for a transfer of possession is

name of frame: _____
type of frame: transfer of possession
source: _____
destination: _____
agent: _____
object: _____

where the source is the person or thing from which the object is taken, the destination is the person or thing to which the object is given, and the agent is the one who performs the transfer. Thus, for "give" the agent is the same as the source, and for "take" the agent is the same as the destination.

When the above frame is instantiated (a specific instance of it is created) for the sentence "Bill took the book from Margaret," the result is

name of frame: T1
type of frame: transfer of possession
source: Mary
destination: Bill
agent: Bill
object: book

For the other statements, we might create the following frames:

name of frame: OC1
type of frame: occupation
worker: Bill
job: professor

name of frame: OC2
type of frame: occupation
worker: Margaret
job: doctor

name of frame: Bill
type of frame: person
. . . (other information about Bill) . . .

name of frame: Margaret
type of frame: person
. . . (other information about Margaret) . . .

name of frame: ADR1
type of frame: address
person: Margaret
street address: _____
city: Akron
state: Ohio

name of frame: book
type of frame: physical object

As Nilsson[21] pointed out, all this information can be translated directly back into first-order predicate logic, but the formulas will look somewhat different. This time, every predicate is binary, and the first argument to each predicate is the frame name.

ELEMENT-OF(T1, transfer-of-possession-events)
SOURCE(T1, Mary)
DESTINATION(T1, Bill)
AGENT(T1, Bill)
OBJECT(T1, book)

ELEMENT-OF(OC1, occupation-events)
WORKER(OC1, Bill)
JOB(OC1, professor)

ELEMENT-OF(OC2, occupation-events)
WORKER(OC2, Margaret)
JOB(OC2, doctor)

ELEMENT-OF(Bill, persons)
. . . (other information about Bill) . . .

ELEMENT-OF(Margaret, persons)
. . . (other information about Margaret) . . .

ELEMENT-OF(ADR1, address-events)
PERSON(ADR1, Margaret)
CITY(ADR1, Akron)
STATE(ADR1, Ohio)

ELEMENT-OF(book, physical-objects)

Putting this information into a semantic net would create a structure similar to that shown in Figure 1.

Although collections of primitive facts can be represented quite nicely using frames and semantic nets, adequately representing complex facts, such as the transitivity formula

$$(x)\ (y)\ (z)\ (R(x,y)\ \&\ R(y,z) \rightarrow R(x,z))$$

mentioned earlier, is more difficult. For more information on how this can be done, see Nilsson[21] or Schubert.[25]

The main advantage of frames or semantic nets over logical representation is that for each object, event, or concept, all the relevant information is collected together. Accessing and manipulating the information is then easier, and default values can be created when information about an object or event is not explicitly given. For example, in the frame for a book, we might have a slot to indicate whether the book is hardbound or paperback. If we are not given a value for this slot, we might want to put in the value "hardbound," with a flag indicating a guessed value. This value could later be changed if new information is given.

Several computer languages have been or are being developed to provide ways to manipulate frames and semantic nets. Examples are KRL,[26] FRL,[27] NETL,[28] and Klone.[29]

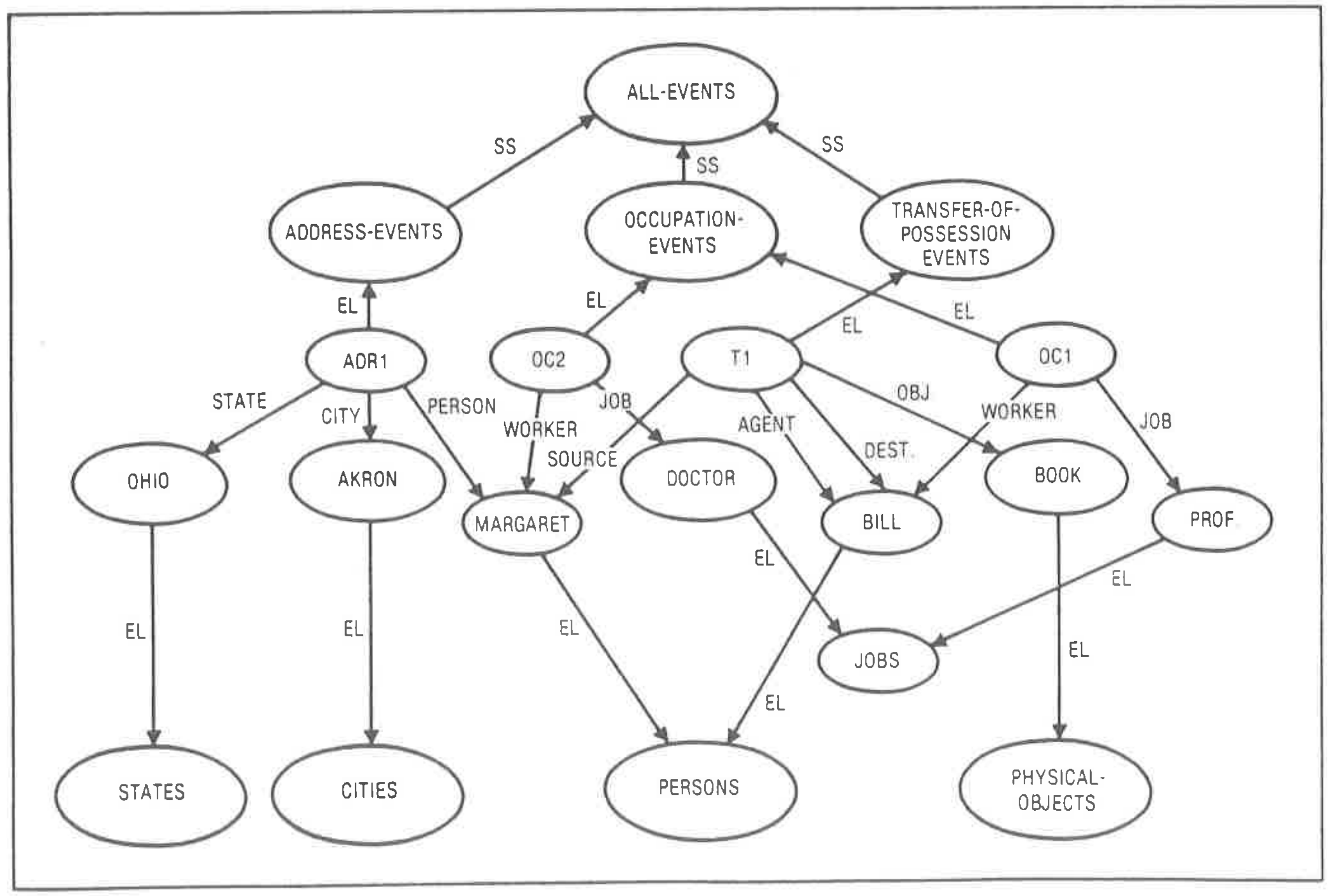


Figure 1. A simple semantic net.

The knowledge-base level: Domain-specific problem-solving knowledge

The domain-specific problem-solving level contains knowledge that is usually procedural in the sense that it tells how the data for a problem can be manipulated in order to go about solving the problem.

Such knowledge could obviously be represented as a conventional computer program, especially when the procedure is well understood, and in fact some expert computer systems have been developed using conventional programming procedures.[30] However, in artificial intelligence problems the precise series of steps necessary to solve the problem may not be known. Consequently, we may have to search through a space containing many alternative paths, not all of which lead to solutions.

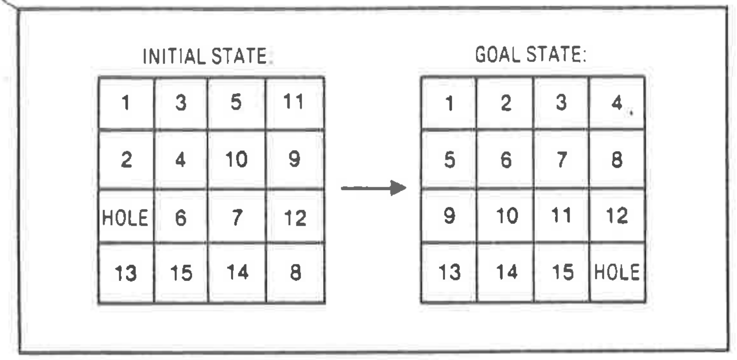


Figure 2. The 15-puzzle. This child's game, the goal of which is to put the tiles in numerical order, is an example of a state-search problem. We are searching for paths that will eventually lead to the goal state.

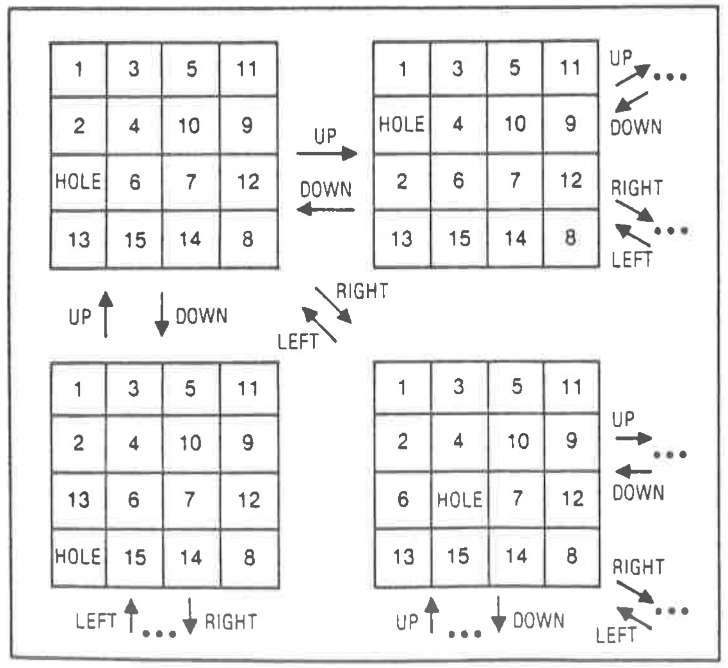


Figure 3. A portion of the state space for the 15-puzzle. The directions to move the tiles constitute the operators that can transform states into other states. More than one operator may be applicable to a state, but not all four are always applicable.

Pattern-invoked programs. In such situations, we may want to encode domain-dependent knowledge in the form of operators or *pattern-invoked programs.* These programs are not called by other programs in the ordinary way but are instead activated by the control structure whenever certain conditions hold in the data. A pattern-invoked program can range from a single statement (as in Mycin[15]) to several hundred lines of coding (some of the "knowledge sources" used in Hearsay II[31] are this large).

Several programming languages, such as Planner,[32] Conniver,[33] Prolog,[34,35] and ARS,[12] allow for pattern-driven invocation of programs in some form. These languages often include features such as automatic backtracking if an attempted solution fails, and ways to schedule the pattern-invoked programs if several are applicable at the same time.

An example of a problem suitable for the use of pattern-invoked programs is a well-known children's game called the 15-puzzle. This puzzle consists of a square frame containing 15 square tiles and a square space, or hole, that can hold any of the tiles (Figure 2). The tiles are numbered from one to 15, and may be moved around by moving a tile into the hole if the tile and hole are adjacent (thus, in effect, moving the hole to where the tile used to be). The objective is to get the squares in numerical order with the hole at the end.

As Nilsson[36] points out, the 15-puzzle is an example of a *state-space search* problem. In this kind of problem, we need to find a path from some *initial state* to any (one or more) *goal state* by applying operators to transform states into other states. In the 15-puzzle, the initial state is the state of the puzzle before any work has been done, and the goal state is the state in which all tiles are in numerical order. The objective is to find any path to this state. The following operators are available:

UP:	if the hole is not at the top of the board then move it up;
DOWN:	if the hole is not at the bottom of the board then move it down;
RIGHT:	if the hole is not at the far right, then move it right;
LEFT:	if the hole is not at the far left, then move it left.

More than one operator may be applicable to a state, but not all four operators are always applicable. Which applicable operator is applied to a state is determined by the control strategy, which is described later.

The state space corresponds to a directed graph in which the states are the nodes of the graph, and in which there is a directed arc from some node A to another node B if an operator exists that can transform state A into state B. Part of the state space for the 15-puzzle is illustrated in Figure 3.

Figure 3 is only one of several possible state-space representations for the 15-puzzle. For example, we may instead choose the set of operators L_i, R_i, U_i, D_i, $i = 1,2, \ldots, 15$, with the following meanings:

L_i:	if tile i is not at the far left, then move it left;
R_i:	if tile i is not at the far right, then move it right;
U_i:	if tile i is not at the top, then move it up;
D_i:	if tile i is not at the bottom, then move it down.

Here, we have 60 operators rather than four, but at most four are applicable at any one time.

Production rules. One type of pattern-invoked program of particular interest is the *production rule,* a degenerate program of the form

IF condition THEN primitive action.

The condition is usually a conjunction of predicates that test properties about the current state, and the primitive action is some simple action that changes the current state. For example, the operators given in the 15-puzzle example are production rules. A problem-solving system in which the domain-dependent procedural knowledge is represented using production rules is known as a production system.

Logical representation. Procedural knowledge can also be represented in first-order predicate logic, if the logical formulas are suitably interpreted. The programming language Prolog[35,37] is an example of such an approach. In Prolog, the formula $B_1 \,\&\, B_2 \,\&\, \ldots \,\&\, B_n - A$ can be thought of either as the logical statement that A is true whenever $B_1, B_2, \ldots, B_n$ are true, or as a procedure for producing a state satisfying condition A. The three basic statements in Prolog and their meanings are

:- A.	A is a goal
A.	A is an assertion
A :- B1, . . . , Bn.	$B_1 \,\&\, \ldots \,\&\, B_n - A$

A and all of the B's must be predicates, and all variables are considered to be universally quantified; that is, the statement is taken to be true for all possible values of the variables.

A Prolog program may contain several different ways to establish a predicate A; for example,

```
A :- B1, B2, . . . , Bi.
A :- C1, C2, . . . , Cj.
```

These statements correspond to the AND/OR graph shown in Figure 4. The solution of a problem presented as a set of Prolog statements is found by doing a depth-first search of the corresponding AND/OR graph until an instantiation of a set of assertions is found that provides a solution graph.

As an example (from McDermott[35]), a Prolog program to append items to a list can be written as

```
append([ ], L, L).
append([X | L1], L2, [X | L3]) :- append(L1, L2, L3).
```

In the above statements, the square brackets denote lists: $[a,b,c]$ is the list containing a, b, and c; [ ] is the null list; and $[a \mid [b,c]] = [a,b,c]$. L, $L1$, $L2$, and $L3$ are variables representing lists, X is a variable representing a list element, and "append(U,V,W)" is a predicate saying that W is the concatenation of U and V. The meaning of the statements is thus (1) the concatenation of [ ] with L is L and (2) if the concatenation of $L1$ with $L2$ is $L3$, then the concatenation of $[X \mid L1]$ with $L2$ is $[X \mid L3]$.

To use the above Prolog program to set the variable A to the concatenation of $[a,b]$ with $[c,d]$, we would write

```
:- append([a,b], [c,d], A).
```

Prolog would then try to instantiate A to whatever value would make the predicate true. The first statement cannot be invoked, since $[a,b]$ is not equal to [ ], but by making the instantiations $X = a$, $L1 = [b]$, $L2 = [c,d]$, and $A = [a,L3]$, the second statement applies. Thus the recursive call

```
:- append([b],[c,d],L3)
```

is evaluated. Again the second statement applies, with the instantiations $X = b$, $L1 = [\ ]$, $L2 = [c,d]$, and $L3 = [b,L3]$ (a different X, $L1$, $L2$, and $L3$, of course), so the recursive call

```
:- append([ ][c,d],L3)
```

is evaluated. Clause 1 applies, giving $L3 = [c,d]$. Returning from the recursive calls, we have $A = [a \mid [b \mid [c,d]]] = [a,b,c,d]$, as desired.

Other techniques. Not all expert systems represent their domain-dependent problem-solving knowledge as pattern-invoked programs, although the majority do. Sometimes, the problem-solving knowledge is represented in terms of conditional probabilities that various events will occur if other events have occurred. This representation is used, for example, to diagnose diseases by repeatedly applying Bayes' theorem[38] to compute the probabilities that certain diseases are present, given that certain symptoms have been observed. This technique is applied to valvular heart disease by Hockstra and Miller[39] and is also available in KMS,[5] a computer system that provides facilities for implementing expert systems using several different ways to represent knowledge bases.

Another way in which problem-solving knowledge is sometimes organized is as static descriptions of phenomena. This representation is used in Internist[11] and is one

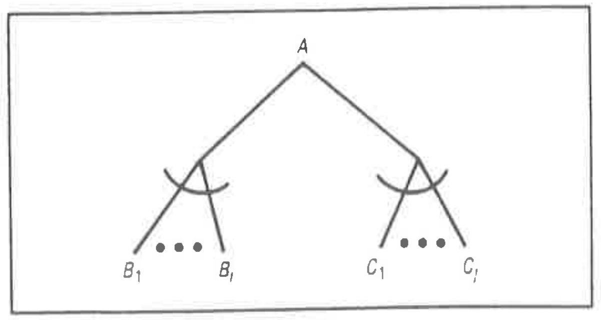


Figure 4. An AND/OR graph corresponding to the Prolog statements A: – B1, B2, . . ., Bi and A: – C1, C2, . . . Cj.

of the representation techniques available in KMS. When using this technique in KMS, we can describe a disease (or some other phenomenon) by listing all the conditions under which it is likely or unlikely to occur and all the symptoms (or effects) it is likely to cause or is capable of causing.[40] Making a medical diagnosis, for example, is then a problem of finding the smallest set of diseases such that all their restrictions are satisfied and the union of all the symptoms they can cause includes the symptoms occurring in the patient. For further discussion of these techniques, see Karp[41] and Reggia et al.[42]

The control level: Procedural knowledge control strategies

We have seen how domain-dependent procedural knowledge can be represented in terms of various types of pattern-invoked programs. Such programs must be invoked by some kind of control strategy, a number of which are possible.

State-space search. State spaces can either be searched in a *forward* direction by starting at the initial state and applying the operators to find a path to a goal state (see earlier discussion on 15-puzzle), or in a *backward* direction by starting with the goal state and applying the inverses of the operators to find a path to the initial state. Which approach is more appropriate depends on the particular problem and the nature of the state space.

Whether an operator is applicable in a forward search is determined by the data describing the current problem state and by the antecedent (the "if" part) of the operator. Thus a forward search is often called a *data-driven* or *antecedent-driven* search. In a backward search, applying an inverse operator to some state *S* corresponds to finding all states that would allow the original (noninverse) operator to produce *S*, and to setting up these states as subsidiary goals (or *subgoals*) to be reached from the initial state. Thus a backward search is often called a *goal-driven search*.

We can easily write a nondeterministic procedure to do a forward state-space search:

```
PROCEDURE state-space:
    s := initial state
    path := NIL /* NIL is the empty list */
    WHILE s is not a goal DO
        ops := ;operators applicable to s;
        nondeterministically select an operator r
          from ops
        path := concatenate(path,r)
            /* path contains all operators used so far */
        s := r(s) /* apply r to s */
    END
    RETURN path
END state-space
```

The nondeterministic selection is done as in a nondeterministic Turing machine: we can visualize the creation of several copies of the program, one for each operator applicable to *s*. Whichever copy of the program finds a path to a goal first returns the path it finds.

Obviously, writing a reasonable deterministic state-space search program requires more thought, and several different techniques have been proposed.

One type of technique is referred to as *backtracking*. Backtracking procedures explore one path as far as possible, ignoring all other paths. If the path dead-ends, or if it otherwise becomes obvious that no paths from the current state will lead to a goal state, the procedure backtracks to a previous state and chooses a new operator to extend the path in a different direction. A backtracking program can thus be written recursively as

```
PROCEDURE backtrack (s, path):
    IF s is a goal THEN RETURN NIL
      /* NIL is the empty list */
    IF decide-to-backtrack(s,path) THEN
      RETURN "fail"
    ops := ;operators applicable to s;
    FOR EVERY r IN ops DO
      /* iterate thru list of applicable operators */
      val = backtrack (r(s), concatenate(path,r))
      IF val is not "fail" THEN RETURN
          concatenate(r,val)
    END
    RETURN "fail"
END backtrack
```

Another type of technique is referred to by Nilsson[21] as graph-searching. Graph-searching procedures explore several paths simultaneously, keeping track of several "current states." Some paths may be explored faster than others, depending on the particular procedure and the particular problem. Examples of such procedures are

- Breadth-first search, in which all paths are searched at the same speed;
- Least-cost-first search (Dijkstra's algorithm[43]), in which at each iteration of the procedure the path that has the least accumulated cost (according to some criterion) is extended;
- Heuristic search, in which various heuristic criteria are used to determine which path or paths to extend next.

Propagation of constraints. In this problem-solving technique, the set of possible solutions becomes further and further constrained by rules or operators that produce "local constraints" on what small pieces of the solution must look like. More and more rule applications are made until no more rules are applicable and only one (or some other small number) possible solution is left. This process can be thought of as a type of state-space search that avoids the necessity of backtracking, since every existing solution must satisfy all the constraints produced by the rule applications.

Example 1: Huffman-Clowes Labeling

As an example, I will consider a technique developed independently by Huffman[44] and Clowes[45] for analyzing two-dimensional line drawings. This technique, which is also discussed by Winston,[22] does not yield "expert"

performance compared with that of human beings, but it is a classic example of propagation of constraints.

The Huffman-Clowes labeling technique applies specifically to 2-D line drawings of 3-D objects composed of flat surfaces. The technique can determine (1) what part of the drawing is the object and what part is the background, (2) which intersections of surfaces are convex and which are concave, and (3) (in some cases) whether the drawing can actually represent a real 3-D object. The technique is explained below, along with an example of how it can be implemented as a state-space search.

The restrictions on the applicability of the Huffman-Clowes technique are

- The drawing must be a simple black-and-white line drawing of a single object, without any indication of coloration, illumination, shadows, cracks, or surface texture.
- Every vertex in the object must be formed by the intersection of exactly three flat surfaces, although not all three of these surfaces need be visible in the drawing.
- No "pathological" points of view of the object are permitted; that is, the point of view given in the drawing must be such that if it is changed by a very small amount, the perceived character of the vertices will remain the same. For example, the point of view shown in Figure 5a cannot be given because the two surfaces that appear to meet in fact do not, as Figure 5b shows.
- If the result of the analysis is a contradiction, then the line drawing cannot possibly represent a real 3-D object. However, if the analysis does not find a contradiction, the drawing may not necessarily be realizable as a 3-D object.

A Huffman-Clowes analysis of such a drawing consists of putting a label on each line to indicate whether it is a convex intersection of surfaces, a concave intersection of surfaces, or a border of the object (Figure 6). Given the restrictions cited above, only four types of vertices are physically possible in any line drawing (Figure 7), and only 16 combinations of labels are physically possible for the lines coming into these vertices (Figure 8).

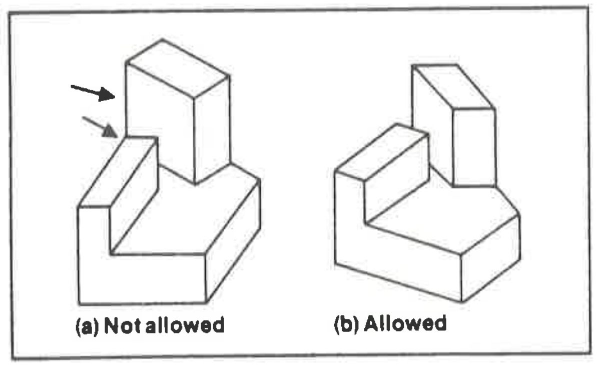


Figure 5. Viewpoints for a line drawing must be chosen so that the perceived character of the vertices is the same as the real character. Viewpoint (a) is not allowed because the indicated line is falsely perceived as going into the indicated vertex. In viewpoint (b), the perception has not been distorted.

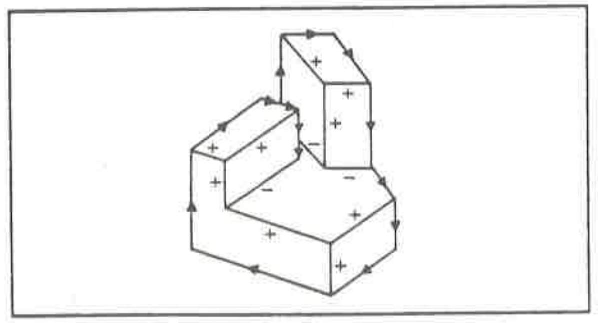


Figure 6. Huffman-Clowes labels on the line drawing of Figure 5. Convex intersections of surfaces are indicated by +'s; concave intersections of surfaces are indicated by −'s; and borders of the object are indicated by arrows directed such that the object is to the right of the arrow and the background is to the left.

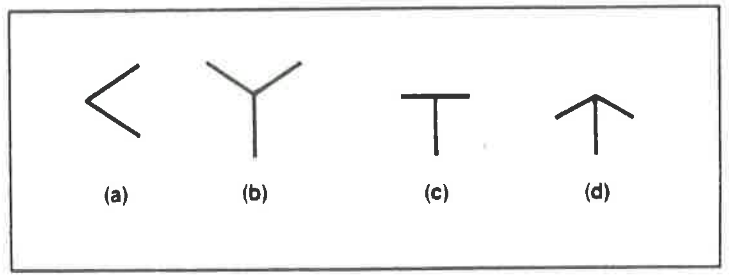


Figure 7. Physically possible types of vertices include (a) an *L* vertex, (b) a *Y* vertex, (c) a *T* vertex, and (d) an arrow vertex.

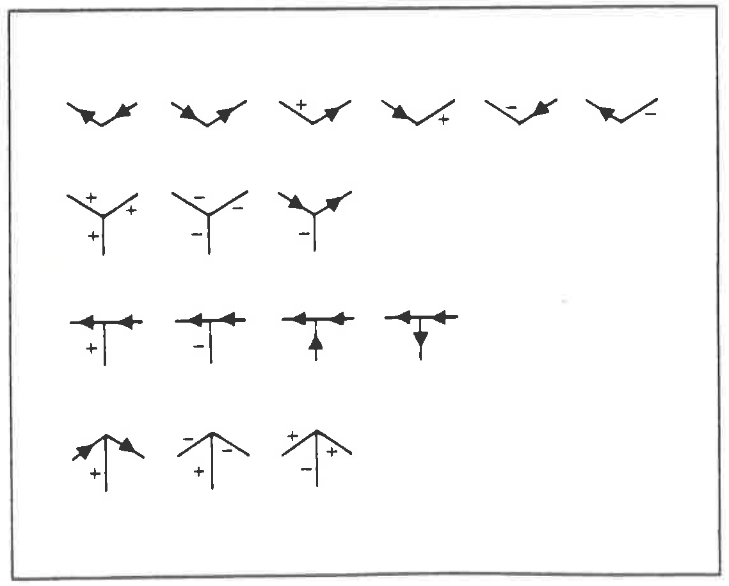


Figure 8. Physically possible vertex labelings. (Used with permission of Addison-Wesley, copyright 1977.[22])

The "propagation of constraints" works as follows. The lines around the perimeter of the object must be labeled with clockwise arrows (Figure 9a). Given the 16 possible label combinations, the number of possible labels for some of the other lines is restricted. For example, the vertex shown in Figure 10a must be labeled as shown in Figure 10b because no other way of labeling yields a legal combination of labels.

Assigning labels to some of the other lines (see Figure 9b) determines the labels on yet more lines, until the entire drawing is labeled (Figure 6). If not all vertices can be assigned legal combinations of labels, then the object cannot be a real 3-D object satisfying the restrictions given above. Figure 11 is an example of this kind of object.

We can construct a state space for the Huffman-Clowes labeling problem as follows. Each state in the space consists of a drawing of the object to be anlayzed, with one or more lines labeled. In the initial state the drawing is unlabeled except for arrows going clockwise around the perimeter of the object. Each of the 16 legal label combinations determines an operator taking unlabeled vertices or partially labeled vertices to fully labeled vertices. For example, the legal label combination shown in Figure 12a corresponds to an operator that applies to any one of the label combinations shown in Figure 12b.

The search procedure may now be written as

```
PROCEDURE label:
    put clockwise labels around the perimeter of the
      object
    WHILE not all lines are labeled DO
        IF there is a vertex with only one applicable
          labeling operator
        THEN apply the operator
        ELSE DO
            /* the object may have more than one
              legal interpretation */
            choose a vertex that is not completely
              labeled
            nondeterministically choose an applicable
              labeling operator
            apply the operator
        END
    END
END label
```

If not all vertices have legal combinations of labels at the end of the procedure's operation, then the object cannot be a real 3-D object. However, the existence of legal labelings for all vertices of the object does not guarantee that the object can be real.[22]

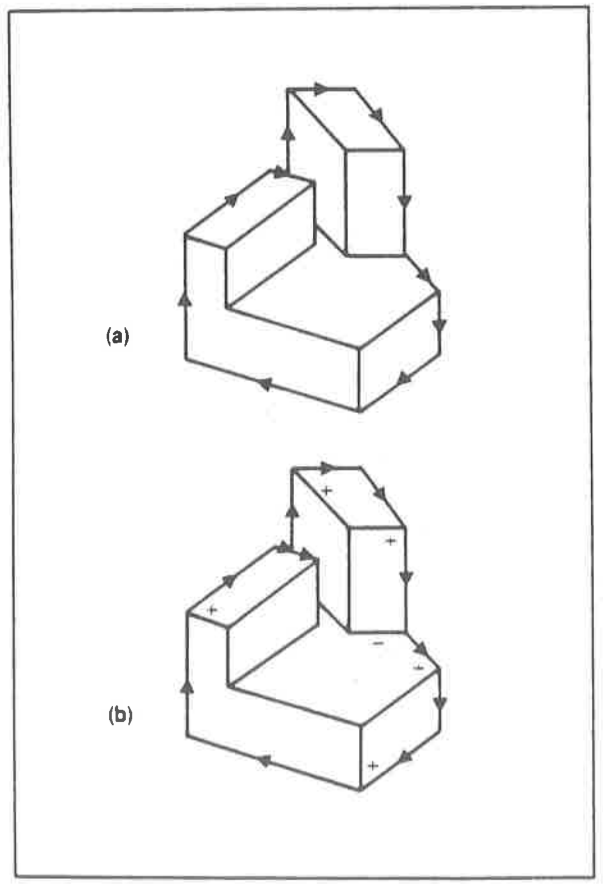


Figure 9. Successive steps in labeling the line drawing of Figure 5.

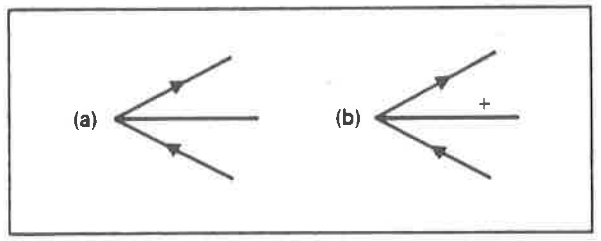


Figure 10. Before (a) and after (b) labeling a line on the basis of the labels of other lines going into the same vertex.

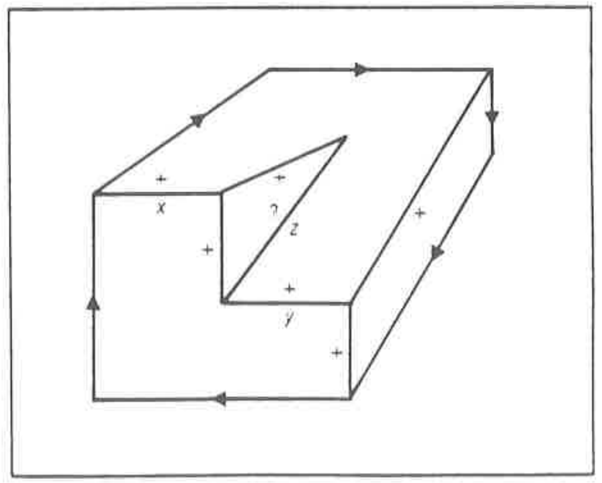


Figure 11. An impossible object. Line z cannot be labeled such that the vertices on both its ends have legal combinations of labels. (Used with permission of Addison-Wesley, copyright 1977 [22])

Example 2: EL

One expert system, called EL,[2] uses propagation of constraints to analyze electrical circuits containing elements such as resistors, transistors, and diodes. The system is written in a language called ARS, for antecedent reasoning system, a language for writing knowledge-based programs that do forward search.

EL's knowledge base exploits many familiar electrical laws to provide propagation of constraint rules. For example, Ohm's law and Kirchhoff's current law give rise to the following rules:

- IF the voltages on both terminals of a resistor are given, THEN compute the current using Ohm's law.
- IF the current and the voltage at one terminal are given, THEN compute the voltage at the other terminal using Ohm's law.
- IF all but one of the currents into a node are given, THEN compute the remaining current using Kirchhoff's current law.

The propagation-of-constraints algorithm for EL, which is similar to the one for Huffman-Clowes labeling, is roughly

```
PROCEDURE propagate:
    Choose an arbitrary node and assign it a potential of
      0.
    WHILE there are nodes whose potentials are not
      known DO
        IF there is an applicable rule
        THEN DO
            apply the rule
            IF two different expressions have now
              been set equal
            THEN solve for a variable and
              eliminate it
        ELSE DO
            choose a node which does not have a
              potential assigned to it
            assign it a potential which is a variable
        END
    END
END propagate
```

As an example, consider the circuit given in Figure 13. If the voltage at the junction of $R3$ and $R4$ is e, then the current going through $R4$ must be $e/10$. But from Kirchhoff's current law, the same current must flow through $R3$, which means that the voltage at the junction of $R1$, $R2$, and $R3$ must be $3e$. Thus, applying Ohm's law again, the current through $R2$ can be determined in terms of e, and so forth. Once the voltage at the junction of $R1$ and the voltage source is determined, the value of e can be determined and the variable e eliminated.

To handle devices such as transistors and diodes, the situation is more complicated. Here, EL uses the "method of assumed states": the operating characteristics of the device are modeled as a number of piecewise-linear regions, and an assumption is made as to which region the device is operating in. If the assumption is wrong, it will eventually lead to a contradiction, and backtracking will be necessary.

One trouble with backtracking is the "combinatorial explosion" that may arise in large search spaces. For example, suppose device A has operating regions $A1$, $A2$, and $A3$, and device B has operating regions $B1$ and $B2$. Six combinations of operating regions are then possible for the two devices. Suppose that (1) a backtrack search first assumes an operating region for device A and then assumes an operating region for device B, as illustrated in Figure 14, and (2) independent of the operating region of device A, device B cannot be operating in region $B1$.

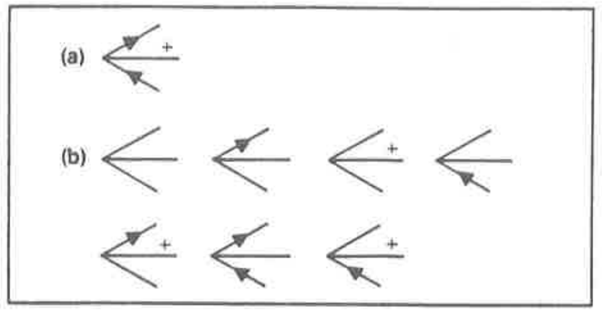


Figure 12. A vertex labeling operator, and the vertices to which it applies. To get the legal label combination (a), we need to evaluate all possible label combinations (b).

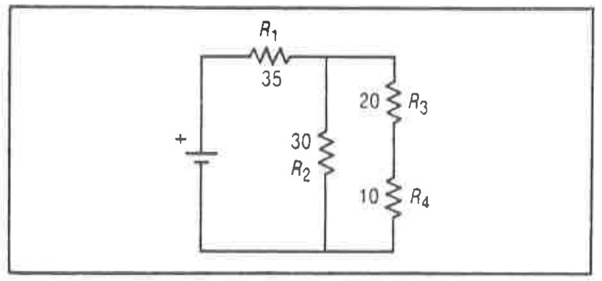


Figure 13. Using Ohm's law and Kirchhoff's current law, EL, an expert system that analyzes electrical circuits, can determine the voltages and currents for resistors R_1 through R_4 such as those in the circuit above.

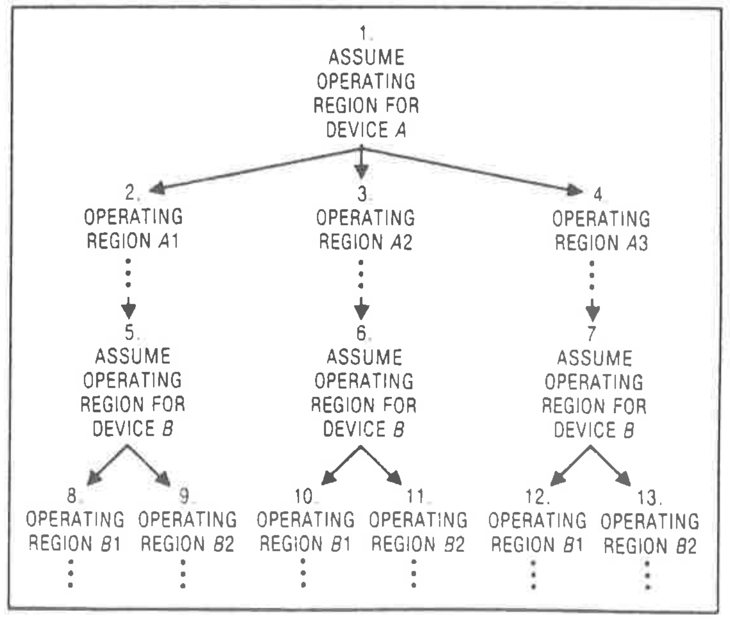


Figure 14. A portion of the state space explored by a backtracking program which first assumes an operating region for device A and then assumes an operating region for device B.

Then, if the search does not turn up a solution along the way, it will eventually try each of the three combinations ($A1$, $B1$) (node 8 of the figure), ($A2$, $B1$) (node 10 of the figure), and ($A3$, $B1$) (node 12 of the figure), even though the failure of ($A1$, $B1$) should have indicated that no other combination involving $B1$ would work. To avoid such redundant searching, EL notes which assumptions were responsible for an error whenever one occurs and never makes that combination of assumptions again. (For a further discussion of EL see Stallman and Sussman.[2])

Problem reduction. One alternative to state-space search is a technique known as *problem reduction*. This strategy is used in some of the early problem-solving systems such as GPS[46] and Strips.[47] Here, the problem to be solved is partitioned or decomposed into subproblems that can be solved separately, in such a way that combining the solutions to the subproblems will yield a solution to the original problem. Each subproblem can be further decomposed into sub-subproblems, which may be even further decomposed, until *primitive* problems, which can be solved directly, are generated.

As an example, consider the 15-puzzle again. As illustrated in Figure 15, the problem of getting from the initial state of the goal state can be decomposed into the following four subproblems:

- Getting the first row in order;
- Getting the first two rows in order, given that the first row is in order;
- Getting the first three rows in order, given that the first two rows are in order;
- Getting all four rows in order, given that the first three rows are in order.

The solutions to these subproblems provide a solution to the original problem if they are simply concatenated together.

Further decomposition is also possible here. The first subproblem, for example, could be decomposed into the sub-subproblems of getting each of the four tiles in the first row into its proper place.

Obviously, more than one way can be used to decompose a problem. For example, the 15-puzzle could have been decomposed into getting the four columns correct, rather than the four rows. We can graphically represent all possible decompositions of a problem in a problem-reduction graph or *AND/OR graph* (Figure 16) in which each OR branch represents a choice of several alternative decompositions, and each AND branch represents a particular way of decomposing a problem.

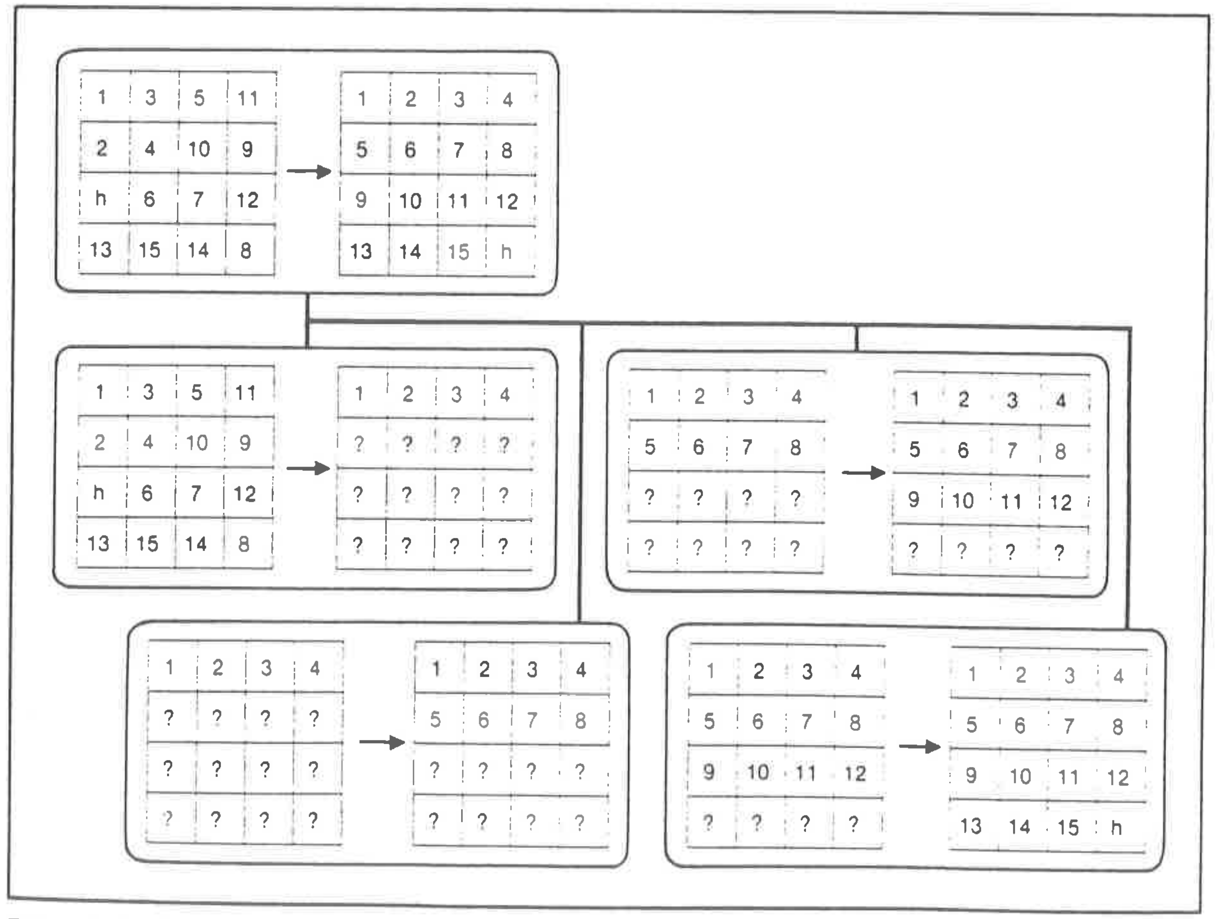


Figure 15. Problem reduction on the 15-puzzle. The 15-puzzle problem can be decomposed into four subproblems, the solutions of which can be concatenated to arrive at the overall solution.

Some decompositions of a problem may lead to solvable subproblems; others may not. To solve a problem using problem reduction, we must choose a decomposition yielding subproblems that can all be solved. To solve each of these subproblems, we must choose decompositions that yield solvable sub-subproblems, and so forth. Thus a problem solution is represented by a *solution graph*, which may be defined recursively as follows.

Let n be a node in an AND/OR graph, and let N be a set of terminal nodes in that graph. We can think of the nodes in N as the set of solvable primitive problems. A solution graph from n to N is defined as

- If n is in N, then the solution graph is simply n.
- If n is not in N and n is terminal, then a solution graph from n to N does not exist.
- If n is not in N, n is not terminal, and the branch from n to its children $n_1, n_2, \ldots, n_k$ is an OR branch, then n is solvable if any one of its children is solvable. If for every i a solution graph G_i exists from n_i to N, then the union of G_i with the node n and the arc (n, n_i) is a solution graph from n to N. Thus several solution graphs from n to N are possible.
- Finally, if n is not in N, n is not terminal, and the branch from n to its children $n_1, n_2, \ldots, n_k$ is an AND branch, then n is solvable only if all its children are solvable. If for $i=1,2,\ldots,k$ a solution graph G_i exists from n_i to N, then the union of all G_i, the node n, and the arcs (n,n_i) $i=1,\ldots,k$ is a solution graph from n to N.

Obviously, a problem solved using problem reduction can have many solution graphs. Depending on the particular problem, we may want all solution graphs, the solution graph of the least (or highest) cost according to some cost criterion, or solution graphs satisfying other criteria.

Suppose Q is a problem that can be solved using either state-space search or problem reduction. Let S be the state-space graph for Q, and R the problem reduction graph. R is often considerably smaller than S (Figure 17). However, since the solution R is a subgraph rather than a path, a typical search procedure for R will usually be much more complicated than a typical search procedure for S. In fact, the number of consecutive problem states

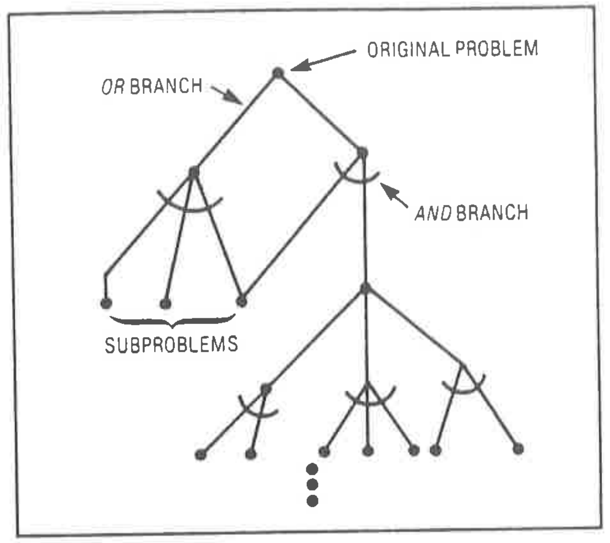
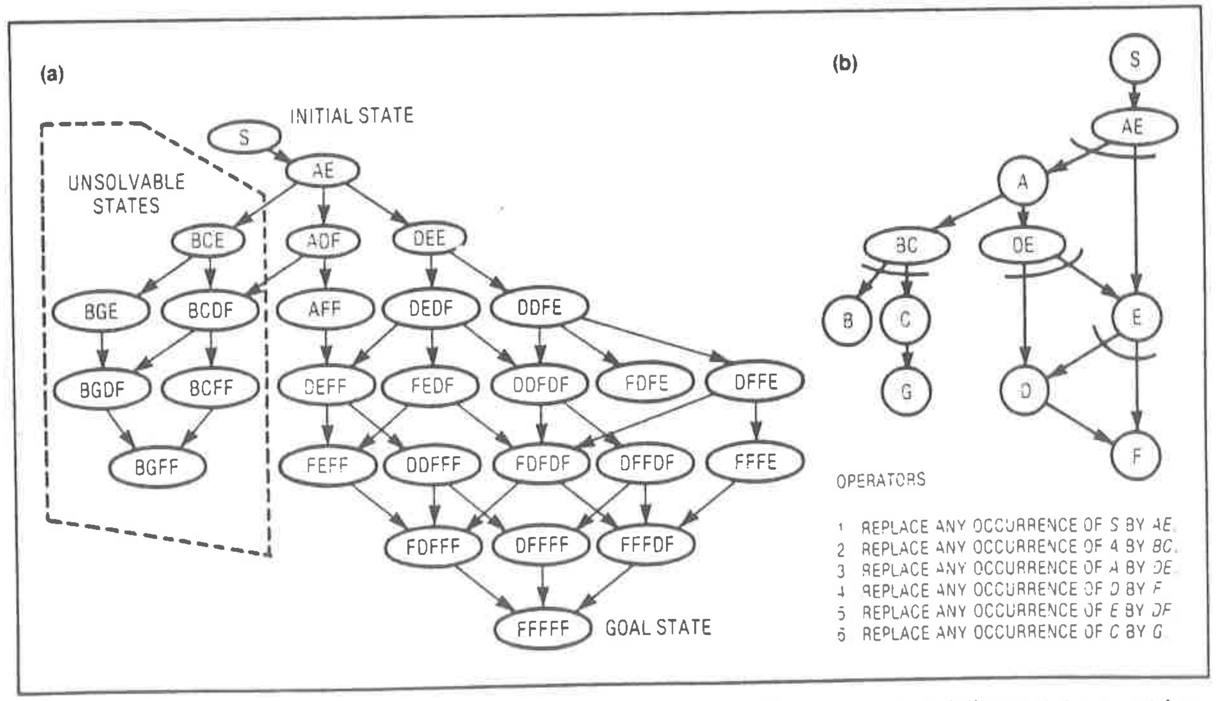


Figure 16. An AND/OR graph. The AND branch is the decomposition chosen in the OR branch, where a choice of several decompositions is made.


(a)
INITIAL STATE
UNSOLVABLE STATES
S
AE
BCE
ADF
DEE
BGE
BCDF
AFF
DEDF
DDFE
BGDF
BCFF
DEFF
FEDF
DDFDF
FDFE
DFFE
BGFF
FEFF
DDFFF
FDFDF
DFFDF
FFFE
FDFFF
DFFFF
FFFDF
FFFFF
GOAL STATE
(b)
S
AE
A
BC
DE
B
C
E
G
D
F
OPERATORS
1 REPLACE ANY OCCURRENCE OF S BY AE.
2 REPLACE ANY OCCURRENCE OF A BY BC.
3 REPLACE ANY OCCURRENCE OF A BY DE.
4 REPLACE ANY OCCURRENCE OF D BY F
5 REPLACE ANY OCCURRENCE OF E BY DF
6 REPLACE ANY OCCURRENCE OF C BY G.
Figure 17. State-space (a) and problem-reduction (b) graphs for a problem The problem is as follows: states are strings of characters, with the initial state being the character S. The goal is to produce any string consisting entirely of F's.

necessary to solve Q using R may be more than the number necessary when using S.

Example 1: Mycin

Consider a search problem in which each operator requires that several conditions hold simultaneously in order for the operator to be applicable. If these conditions can be established independently, problem reduction can be combined with a backward (goal-driven) search in the following manner: Any time a rule is found whose application can achieve a subgoal, the problem of satisfying all of the preconditions of that rule is decomposed into the problems of satisfying each precondition independently, and these problems are set up as subgoals. Many expert systems use this approach; one of the first was Mycin.[15]

Mycin is a program, written in Lisp, that can diagnose infectious diseases and recommend treatment. Mycin interacts with the user, asking questions about the symptoms of the disease, to determine a diagnosis. The state of a diagnosis problem in Mycin is a collection of all known facts about the problem, which is augmented and changed by various operators as the diagnosis proceeds. Mycin uses production rules to represent causal relations among the facts.

Since we cannot be completely certain if some facts are true or certain causal relations hold, each fact and each production rule has associated with it a "certainty factor." The CF indicates the certainty with which each fact or rule is believed to hold and is a number in the interval $[-1,1]$. Positive and negative CFs indicate a predominance of confirming or disconfirming evidence, respectively. CFs of 1 or -1 indicate absolute knowledge.

A typical Mycin production rule is the following:[15]

```
PREMISE  ($AND (SAME CNTXT INFECT PRIMARY-
             BACTEREMIA)
                (MEMBF CNTXT SITE STERILESITES)
                (SAME CNTXT PROTAL GI))
ACTION   CONCLUDE CNTXT IDENT BACTEROIDES
             TALLY .7)
```

If (1) the infection is primary-bacteremia,
(2) the site of the culture is one of the sterilesites, and
(3) the suspected portal of entry of the organism is the gastro-intestinal tract,
then there is suggestive evidence (.7)
that the identify of the organism is bacteroides.

```
Procedure FINDVALUEOF (item GOAL)
  begin item X; list L; rule R; premise__clauseP;
  if (X—UNITYPATH(GOAL)) then return (X);
  If LABDATA(GOAL) and DEFINITE__ANSWER(X—ASKUSER(GOAL)) then return(X);
  L—RULES__ABOUT(GOAL);
  L—APPLY__METARULES(GOAL,L,0);
  for R ∈ L DO
    unless PREVIEW(R) = false do
      begin ''evaluate-rule''
      for P ∈ PREMISES__OF(R) do
        begin ''test-each-premise-clause''
        if not TRACED(ATTRIBUTE__IN(P)) then FINDVALUEOF(ATTRIBUTE__IN(P));
        if EVALUATION__OF(P) < .2 then next (R);
        end ''test-each-premise-clause'';
      CONCLUDE(CONCLUSION__IN(R));
      if VALUE__KNOWN__WITH__CERTAINTY(GOAL) then
        begin MARK__AS__TRACED(GOAL); return (VALUEOF(GOAL)); end;
      end ''evaluate-rule'';
  MARK__AS__TRACED(GOAL);
  if VALUEOF(GOAL) = unknown and NOT__ALREADY__ASKED(GOAL)
      then return(ASKUSER(GOAL))
      else return(VALUEOF(GOAL));
end;

Procedure APPLY__METARULES(item GOAL; list L; integer LEVEL);
  begin list M; rule Q;
  if (M—METARULES__ABOUT(GOAL,LEVEL+1))
    then APPLY__METARULES(GOAL,M,LEVEL+1);
  for Q ∈ M do USE__METARULE__TO__ORDER__LIST(Q,L);
  return(L);
  end;

Procedure CONCLUDE(action__clause CONCLUSION);
  begin rule T; list l;
  UPDATE__VALUE__OF(ATTRIBUTE__IN(CONCLUSION),VALUE__IN(CONCLUSION));
  l—ANTECEDENTRULES__ASSOCIATED__WITH(CONCLUSION);
  l—APPLY__METARULES(ATTRIBUTE__IN(CONCLUSION),l,0);
  for T ∈ l do CONCLUDE(CONCLUSION__IN(T));
  end;
```

Figure 18. The control structure for Mycin, a program written in Lisp that can diagnose infectious diseases and recommend treatment. (Used with permission of North-Holland, copyright 1977.[10])

$AND, a multivalue AND operation, is used to manipulate CFs as described below.

In Mycin, facts are represented in the form of four tuples such as the following:[15]

```
(SITE CULTURE-1 BLOOD 1.0)
(IDENT ORGANISM-2 KLEBSIELLA .25)
(IDENT ORGANISM-2 E.COLI .73)
(SENSITIVS ORGANISM-1 PENICILLIN-1)
```

To evaluate Mycin's production rules, we take the following steps.

- The CF of a conjunction of several facts is taken to be the minimum of the CFs of the individual facts.
- The CF for the conclusion produced by a rule is the CF of its premise multiplied by the CF of the rule.
- The CF for a fact produced as the conclusion of one or more rules is the maximum of the CFs produced by the rules yielding that conclusion.

For example, suppose Mycin is trying to establish fact D, and the only rules concluding anything about D are

IF A and B and C, THEN CONCLUDE D (CF = .8)
IF H and I and J, THEN CONCLUDE D (CF = .7)

Suppose further that facts A, B, C, H, I, and J are known with CFs of .7, .3, .5, .8, .7, and .9, respectively. Then the following computation produces a CF of .49 for D.

```
IF A and B and C,           |
THEN D(CG = .8)             |
  CF(A) = .7 |              | → .8 × .3 = .24   |
  CF(B) = .3 | → min = .3   |                   |
  CF(C) = .5 |              |                   |
                                                | → max = .49
IF H and I and J,           |                   |
THEN D (CF = .7)            |                   |
  CF(H) = .8 |              | → .7 × .7 = .49   |
  CF(I) = .7 | → min = .7   |
  CF(J) = .9 |              |
```

In this example, facts A, B, C, H, I, and J would typically be established by other production rules bearing on them. The chaining of these rules together to establish D corresponds to searching an AND/OR graph. Mycin diagnoses diseases by setting up the diagnosis problem as a goal and then doing a depth-first search of the resulting AND/OR graph. The search strategy of Mycin is fairly simple, as shown in Figure 18. For example, suppose Mycin's goal is to find the value of A, and some of its rules are

1. IF F = f THEN CONCLUDE C = c (CF = .5)
2. IF G = g and H = h THEN CONCLUDE C = c (CF = .6)
3. IF H = h and I = i THEN CONCLUDE C = c' (CF = .7)
4. IF B = b and C = c THEN CONCLUDE A = a (CF = .8)

. . . other rules making conclusions about A . . .

Suppose it also knows that the values of B, F, G, H, I, and E are laboratory data, determined by asking the user for their values. The AND/OR graph corresponding to these rules is shown in Figure 19. Mycin searches this graph depth-first from left to right, determining the values of B, F, G, H, I, C, and A in turn.

Note that when a rule such as

IF B = b and C = c THEN CONCLUDE A = a (CF = .8)

is invoked, the subgoals Mycin creates are not to prove that $B = b$ and $C = c$ but rather to find the values of B and C. The system can then focus on a particular topic when interacting with the user, rather than jumping around from topic to topic. Also, since the information accumulated about the subgoals is saved, B and C need not be reevaluated if another rule is ever encountered that requires information about them.

Note also that every rule relevant to a particular goal must be invoked unless the value of the goal can be established with a CF of 1 or -1. However, if one of the premises of a rule is already known to have a CF of -1, then that rule need not be invoked, since it cannot possibly conclude anything.

One major problem with medical diagnosis is that the requisite knowledge is generally accumulated experientially by a physician and cannot be regurgitated on demand to be put into a computer system. Mycin handles this problem by means of the following simple learning mechanism.

A user who already knows the answer to some diagnostic problem can give the problem to Mycin to solve. If Mycin reaches an incorrect conclusion, the user can invoke a question-answering system to find out which rules were invoked and why. If the user decides that one of the rules is incorrect or that a new rule needs to be added, he can make the appropriate change or addition. Since the rules are invoked automatically whenever

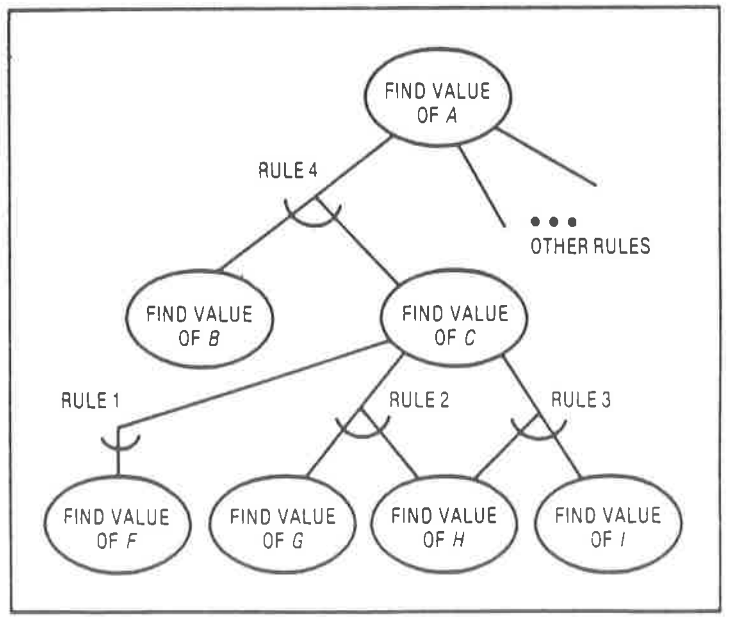


Figure 19. An AND/OR graph for Mycin. Mycin searches this graph depth-first from left to right, determining the values of *B*, *F*, *G*, *H*, *I*, *C*, and *A* in turn.

they have bearing on a goal, no other change need be made to the system to ensure that a rule is used.

The advantages of the production system architecture of Mycin are

- Each production rule is completely modular and independent of the other rules. Thus changing or adding to Mycin's knowledge base is easy.

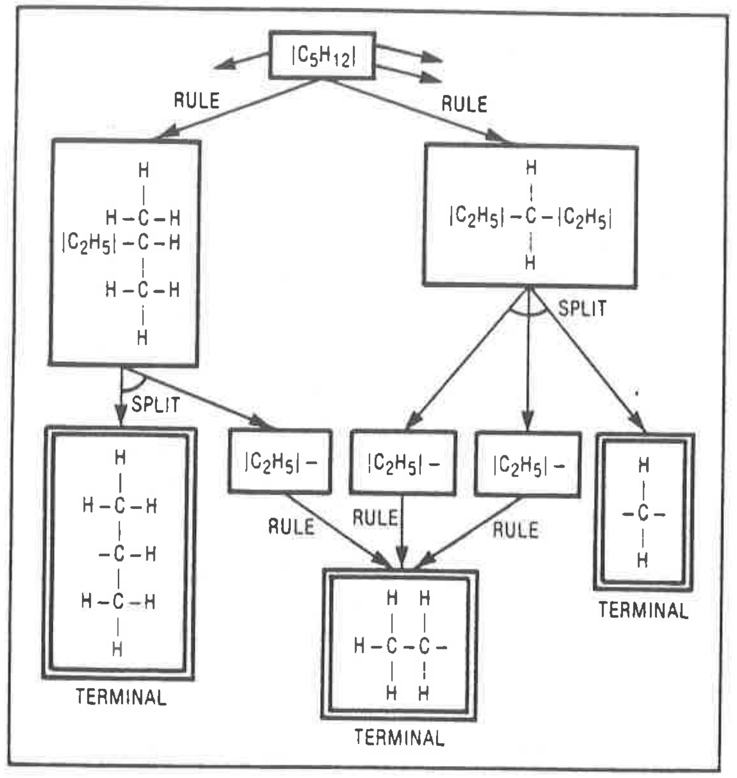


Figure 20. An AND/OR graph for a sample problem for Dendral, a system that proposes plausible chemical structures for molecules given their mass spectrograms. (Used with permission of Tioga Publishing, copyright 1980.[21])

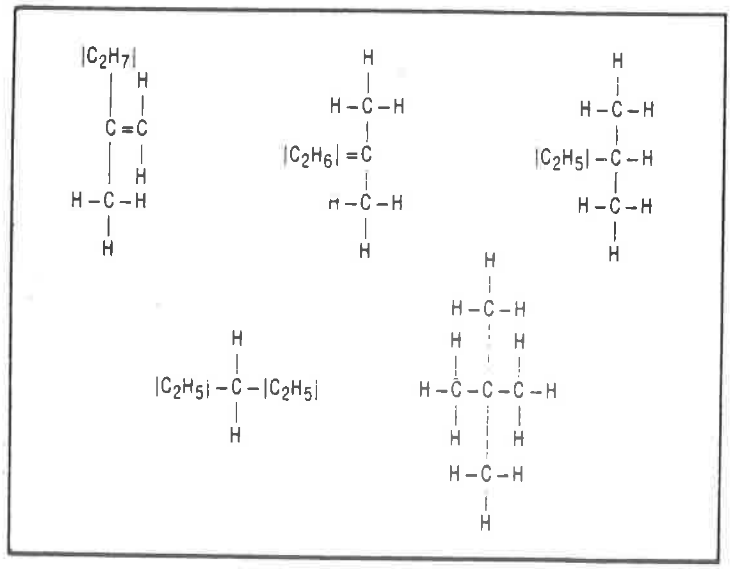


Figure 21. Partial structures proposed by the "generate" part of Dendral for C_2H_5. (Used with permission of Tioga Publishing, copyright 1980.[21])

- The stylized nature of the production rules makes the coding easy to examine. Thus the question-answering system can supply clear answers, in most cases, to questions about how and why it made its diagnosis.
- Each rule represents a small, isolated chunk of knowledge. Thus a physician familiar with the system may be able to formulate new rules if necessary.

The disadvantages are

- Since the rules are not called explicitly by other rules, we are not sure what side-effects may occur from adding or changing a rule. Some impact may arise from unexpected interactions with other rules.
- Sometimes we cannot easily represent a piece of knowledge about a disease as a production rule.
- Since Mycin searches backwards from the goal of diagnosing and treating a disease to the known data about the patient, a sequence of desired actions or tests cannot always be mapped into a set of production rules whose invocation will provide that sequence.

Sometimes rules can be written in a more general format than that given earlier. For example, we might want a rule saying "for each organism such that . . . , conclude . . . ." A few of these rules have been put into the system, but they cause other problems. For example, the question-answering system cannot adequately explain the invocation of these rules.

As of the date of the reference material,[15] Mycin could perform diagnoses with an agreement of about 72 percent with medical experts. This figure has probably improved since.

The knowledge base of Mycin can be removed and a set of rules from another domain substituted. This approach has been used for the problem of diagnosing lung diseases, yielding a program called Puff.[17]

Example 2: Dendral

Problem reduction can also be used in conjunction with a forward (data-driven) search. For example, suppose that applying an operator yields a state that contains several independent items. The state can then be decomposed into each of these items, which can be worked on independently. This technique is used in Dendral,[9] a computer system that proposes plausible chemical structures for molecules, given their mass spectrograms.

Dendral uses a "plan, generate, and test" technique. First, in the planning phase, constraints on the problem solution are inferred from mass spectrometry data. Second, in the generation phase, the program generates all molecular structures satisfying these constraints, as well as general chemical constraints. Third, in the testing phase, the proposed structures are tested in a more sophisticated manner for compatibility with the mass spectrometry data.

The generation phase of Dendral's operation is a data-driven problem-reduction search. Dendral contains a number of operators, each of which takes some of the

atoms in a chemical formula and replaces them with one or more pieces of molecular structure. The problem is decomposed by considering each of these pieces of structure independently and applying other operators to it to fill in more details. Figures 20 and 21 are examples of this approach.

A difficulty that many problem-reduction systems encounter is that the subproblems into which a problem is decomposed may not be independent of each other, and their solutions cannot be combined into a solution to the original problem. Dendral has this problem and uses a testing phase to eliminate the solutions proposed in the generation phase that do not work.

For a further discussion of the operation of Dendral see Feigenbaum et al.[9]

Large-scale expert systems

Hearsay-II. The pattern-invoked programs we have seen so far are relatively simple programs, such as production rules. Hearsay-II, a computer system for speech understanding, uses pattern-invoked programs that are much larger. In addition, its architecture allows both data-driven and goal-driven invocation of these programs. Hearsay-II is not really an expert system because its performance is not as good as that of any reasonably good speaker of English, but I include it here because it represents the latest in automated speech recognition. Also, the techniques used in its construction should be of interest to designers of expert systems.

The performance of Hearsay-II is given in Table 2,[31] and its architecture is shown in Figure 22. The KSs, or *knowledge sources,* indicated in the figure are pattern-invoked computer programs. Each KS consists of a condition program that evaluates whether the KS is applicable and an action program to accomplish whatever

Table 2.
The Performance of Hearsay-II.*

Number of speakers:	One
Environment:	Computer terminal room (>65 dB)
Microphone:	Medium quality, close talking
System speaker-tuning:	20-30 training utterances
Speaker adaptation:	None required
Task:	Document retrieval
Language constraints:	Context-free semantic grammar; other restrictions
Test data:	23 utterances, brand new to the system, run blind; 7 words/utterance (average); 2.6 seconds/utterance (average)
Accuracy:	9% sentences misunderstood; 10% sentences not word-for-word correct but meaning understood anyway
Computing resources:	60M instructions/second of speech on a PDP-10
Time required to comprehend speech:	On the order of 10 times the length of the utterance

*Adapted with permission of the ACM, copyright 1980.[31]

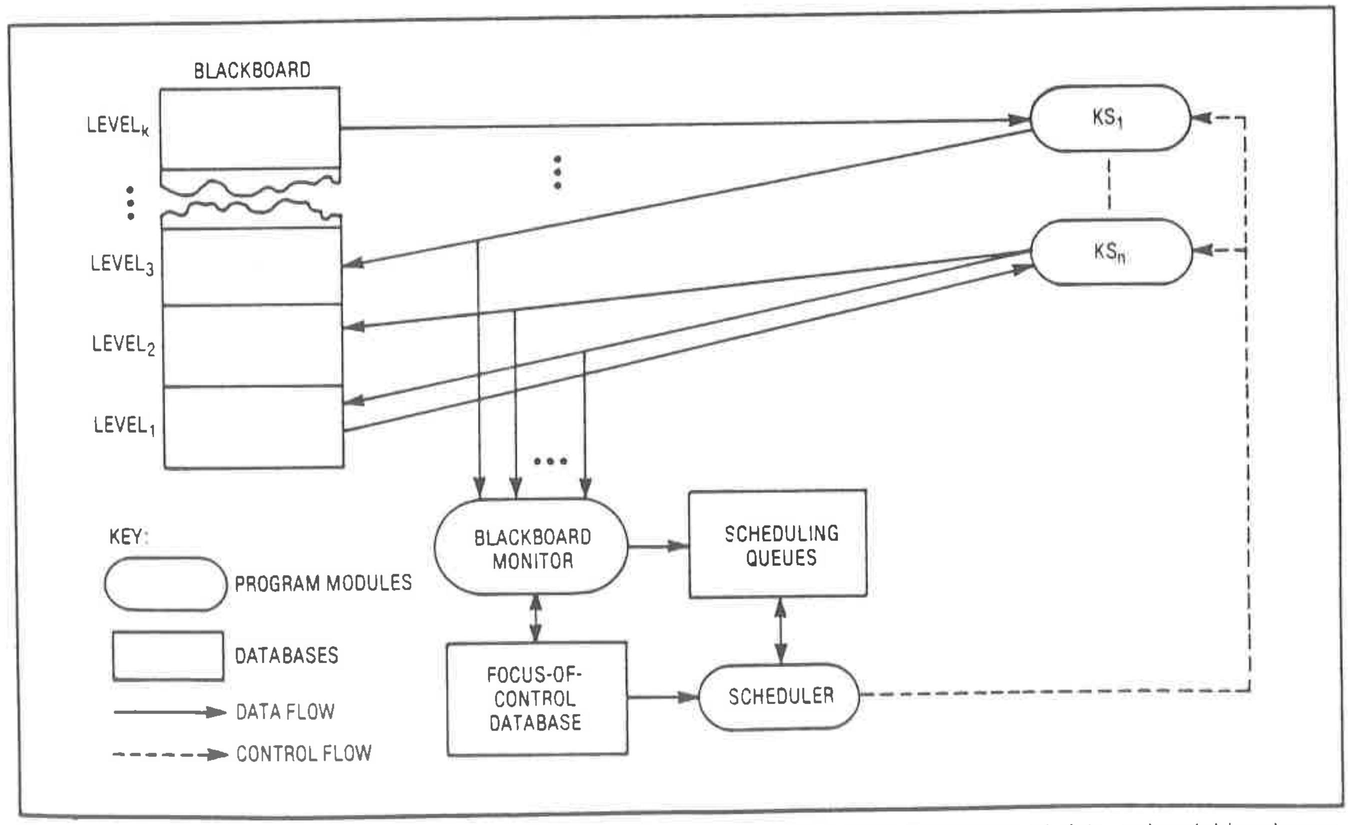


Figure 22. Schematic of the architecture of Hearsay-II, a system for speech understanding that allows both data- and goal-driven invocation of pattern-invoked programs. (Used with permission of the ACM, copyright 1980.[31])

they have bearing on a goal, no other change need be made to the system to ensure that a rule is used.

The advantages of the production system architecture of Mycin are

- Each production rule is completely modular and independent of the other rules. Thus changing or adding to Mycin's knowledge base is easy.

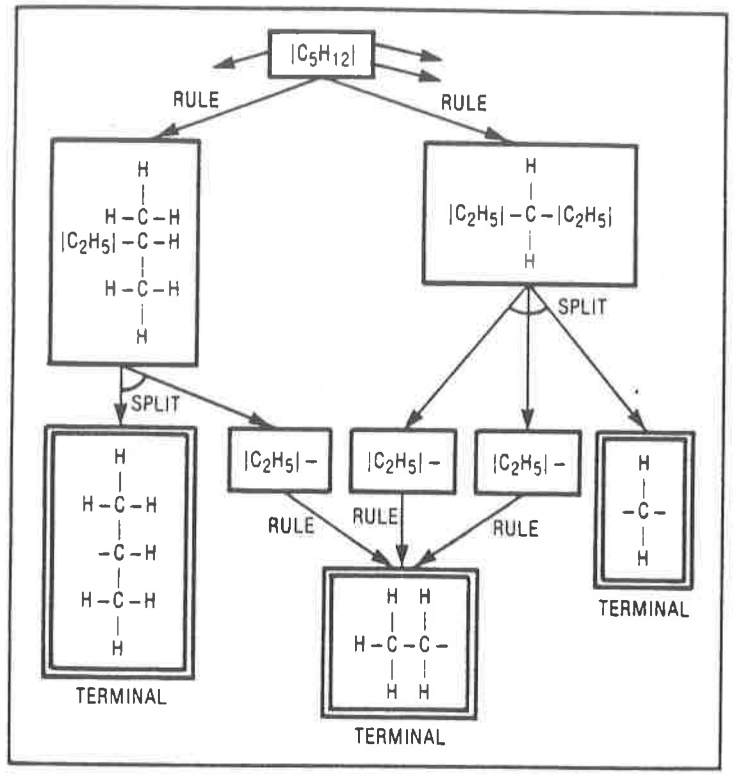


Figure 20. An AND/OR graph for a sample problem for Dendral, a system that proposes plausible chemical structures for molecules given their mass spectrograms. (Used with permission of Tioga Publishing, copyright 1980.[21])

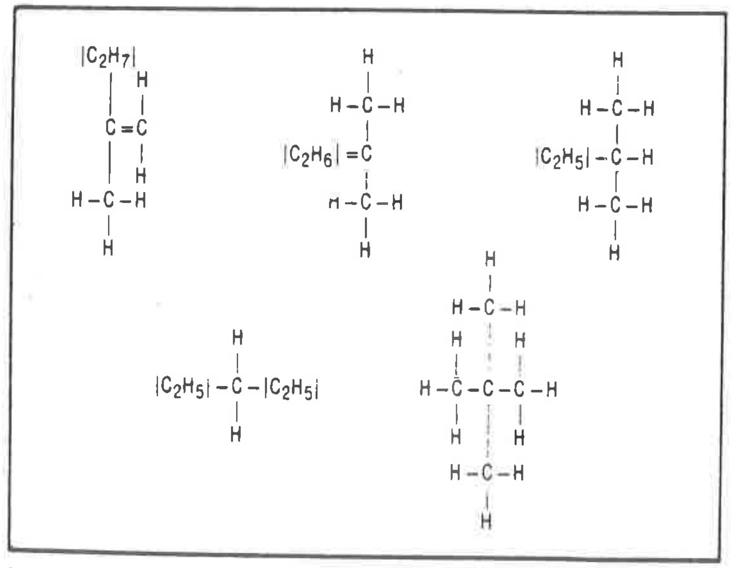


Figure 21. Partial structures proposed by the "generate" part of Dendral for C_2H_5. (Used with permission of Tioga Publishing, copyright 1980.[21])

- The stylized nature of the production rules makes the coding easy to examine. Thus the question-answering system can supply clear answers, in most cases, to questions about how and why it made its diagnosis.
- Each rule represents a small, isolated chunk of knowledge. Thus a physician familiar with the system may be able to formulate new rules if necessary.

The disadvantages are

- Since the rules are not called explicitly by other rules, we are not sure what side-effects may occur from adding or changing a rule. Some impact may arise from unexpected interactions with other rules.
- Sometimes we cannot easily represent a piece of knowledge about a disease as a production rule.
- Since Mycin searches backwards from the goal of diagnosing and treating a disease to the known data about the patient, a sequence of desired actions or tests cannot always be mapped into a set of production rules whose invocation will provide that sequence.

Sometimes rules can be written in a more general format than that given earlier. For example, we might want a rule saying "for each organism such that . . . , conclude . . . ." A few of these rules have been put into the system, but they cause other problems. For example, the question-answering system cannot adequately explain the invocation of these rules.

As of the date of the reference material,[15] Mycin could perform diagnoses with an agreement of about 72 percent with medical experts. This figure has probably improved since.

The knowledge base of Mycin can be removed and a set of rules from another domain substituted. This approach has been used for the problem of diagnosing lung diseases, yielding a program called Puff.[17]

Example 2: Dendral

Problem reduction can also be used in conjunction with a forward (data-driven) search. For example, suppose that applying an operator yields a state that contains several independent items. The state can then be decomposed into each of these items, which can be worked on independently. This technique is used in Dendral,[9] a computer system that proposes plausible chemical structures for molecules, given their mass spectrograms.

Dendral uses a "plan, generate, and test" technique. First, in the planning phase, constraints on the problem solution are inferred from mass spectrometry data. Second, in the generation phase, the program generates all molecular structures satisfying these constraints, as well as general chemical constraints. Third, in the testing phase, the proposed structures are tested in a more sophisticated manner for compatibility with the mass spectrometry data.

The generation phase of Dendral's operation is a data-driven problem-reduction search. Dendral contains a number of operators, each of which takes some of the

atoms in a chemical formula and replaces them with one or more pieces of molecular structure. The problem is decomposed by considering each of these pieces of structure independently and applying other operators to it to fill in more details. Figures 20 and 21 are examples of this approach.

A difficulty that many problem-reduction systems encounter is that the subproblems into which a problem is decomposed may not be independent of each other, and their solutions cannot be combined into a solution to the original problem. Dendral has this problem and uses a testing phase to eliminate the solutions proposed in the generation phase that do not work.

For a further discussion of the operation of Dendral see Feigenbaum et al.[9]

Large-scale expert systems

Hearsay-II. The pattern-invoked programs we have seen so far are relatively simple programs, such as production rules. Hearsay-II, a computer system for speech understanding, uses pattern-invoked programs that are much larger. In addition, its architecture allows both data-driven and goal-driven invocation of these programs. Hearsay-II is not really an expert system because its performance is not as good as that of any reasonably good speaker of English, but I include it here because it represents the latest in automated speech recognition. Also, the techniques used in its construction should be of interest to designers of expert systems.

The performance of Hearsay-II is given in Table 2,[31] and its architecture is shown in Figure 22. The KSs, or *knowledge sources,* indicated in the figure are pattern-invoked computer programs. Each KS consists of a condition program that evaluates whether the KS is applicable and an action program to accomplish whatever

Table 2.
The Performance of Hearsay-II.*

Number of speakers:	One
Environment:	Computer terminal room (>65 dB)
Microphone:	Medium quality, close talking
System speaker-tuning:	20-30 training utterances
Speaker adaptation:	None required
Task:	Document retrieval
Language constraints:	Context-free semantic grammar; other restrictions
Test data:	23 utterances, brand new to the system, run blind; 7 words/utterance (average); 2.6 seconds/utterance (average)
Accuracy:	9% sentences misunderstood; 10% sentences not word-for-word correct but meaning understood anyway
Computing resources:	60M instructions/second of speech on a PDP-10
Time required to comprehend speech:	On the order of 10 times the length of the utterance

*Adapted with permission of the ACM copyright 1980.[31]

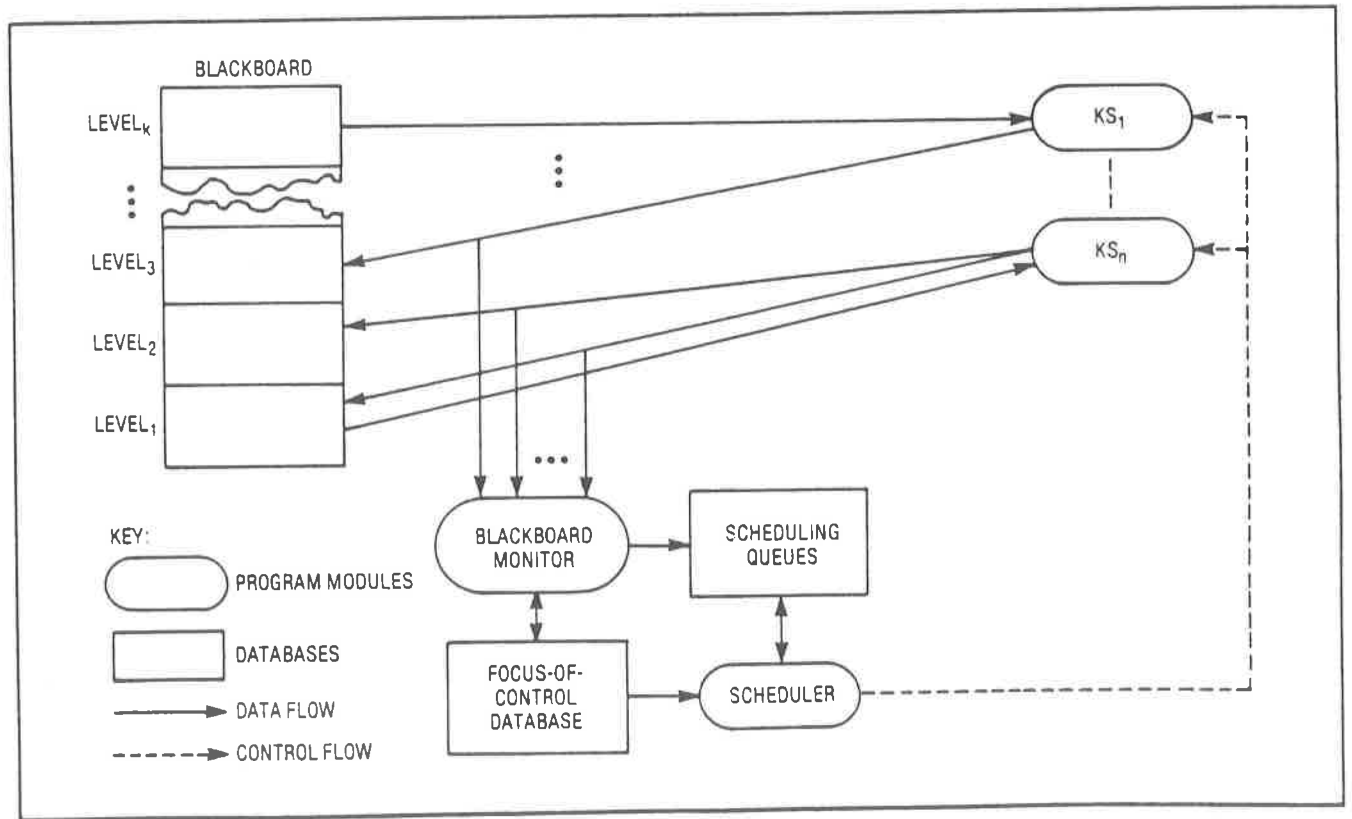


Figure 22. Schematic of the architecture of Hearsay-II, a system for speech understanding that allows both data- and goal-driven invocation of pattern-invoked programs. (Used with permission of the ACM, copyright 1980.[31])

results the KS is to produce. The system contains approximately 40 KSs, which are from five to 100 pages of source code apiece. Thirty pages is a typical KS size. Each KS has up to 50K bytes of its own local data storage.

The KSs communicate with each other by posting messages on a global data structure called the blackboard. Messages posted on the blackboard are noted by the blackboard monitor, which creates entries on the scheduling queues for any KS whose applicability conditions might be satisfied. For each KS condition program or action program on the queues, the scheduler creates a priority. The highest priority activity is removed from the queues and executed.

The blackboard is divided into several levels, which can be thought of as the various levels in a problem-reduction tree at which subproblems are located. The condition program of each KS tests events occurring at a particular level or levels of the blackboard, and the action program of the KS puts hypotheses at a particular level or levels of the blackboard (Figures 23 and 24).

Examples of KSs are

- SEG divides the input signal up into segments and assigns to each segment several alternative possibilities for the phoneme it might be.
- PARSE is a KS that takes sequences of words and parses them into phrases. PARSE consists of an encoding of the grammar for the task language as a network, and procedures for searching the network to parse a sequence of words.

Considered as a tool for designing speech understanding systems, Hearsay-II provides ways to define blackboard levels, configure KS groups, access and modify blackboard hypotheses, activate and schedule KSs, and debug and analyze KS performance. It also provides ways to specify which KSs should have their condition programs invoked when new hypotheses appear on the blackboard, to read hypotheses from the blackboard, and to put new hypotheses onto the blackboard.

The main features that distinguish the Hearsay-II architecture from that of systems such as Mycin are the use of arbitrary pattern-invoked programs as units of knowledge rather than production rules and the flexibility of the scheduler (as opposed to the strict goal-driven invocation used in Mycin).

For a large, complex problem such as speech understanding, these features offer several advantages. Since the KSs can be arbitrarily complex—and arbitrarily different in their internal operation—the most appropriate problem-solving approach can be implemented at each level of processing. Each KS may itself be a small knowledge-based problem solver, and its internal processes have only local effects, rather than causing potential interactions with the rest of the system. This quality alleviates the "combinatorial explosion" that often occurs when search techniques are used on very large problems. In fact, when portions of Hearsay-II were experimentally rewritten as a production system, the system ran approximately 100 times slower.[48]

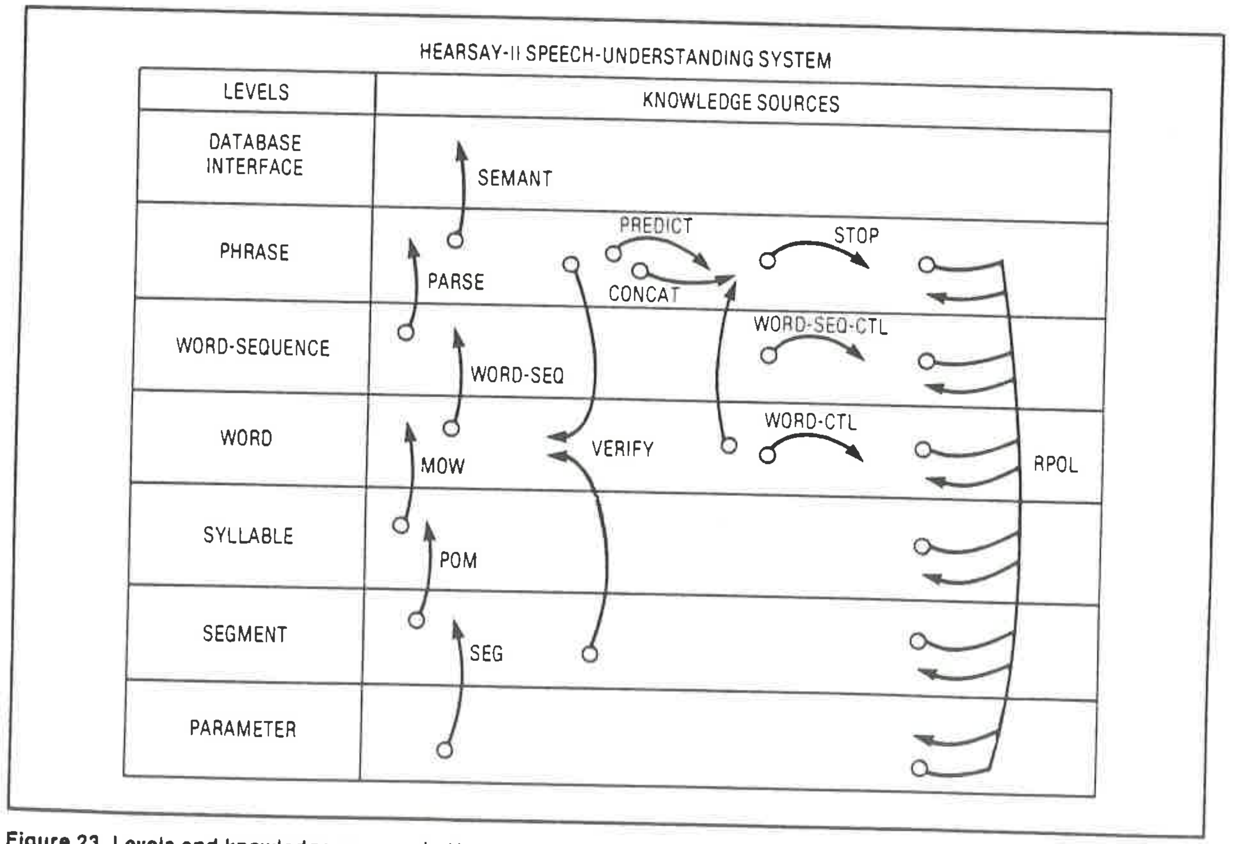


Figure 23. Levels and knowledge sources in Hearsay-II, as of September 1976. KSs are indicated by vertical arcs with the circled ends indicating output levels. (Used with permission of the ACM, copyright 1980.[31])

Casnet. An expert system can be set up to operate at several different levels of knowledge. Hart[49] has characterized a *surface system* as "one having no knowledge of such fundamental concepts as causality, intent, or basic physical principles," and a *deep system* as "one that does attempt to represent physical concepts at this level." Mycin, for example, is a surface system: its rules linking observations with diseases do not include any knowledge of how the diseases arise or progress or how their symptoms are caused. The Casnet system is a deep system because it uses a model of the causal relations among a detailed set of disease states in a family of diseases known as glaucoma.[8] Casnet uses production rules, semantic nets, and other techniques, as well as the following three levels of disease description:

- Observations (or tests). These consist of disease symptoms and laboratory tests and form the direct evidence that a disease is present.
- Pathophysiological states. These are the internal condtions assumed to occur in the patient. "States" as used here are different from the problem states mentioned earlier. Here, each state is a condition that may or may not be present, and the states are not mutually exclusive.
- Categories of disease. Each category consists of a pattern of states and observations.

Signal Acquisition, Parameter Extraction, Segmentation, and Labeling:
- SEG: Digitizes the signal, measures parameters, and produces a labeled segmentation.

Word Spotting:
- POM: Creates syllable-class hypotheses from segments.
- MOW: Creates word hypotheses from syllable classes.
- WORD-CTL: Controls the number of word hypotheses that MOW creates.

Phrase-Island Generation:
- WORD-SEQ: Creates word-sequence hypotheses that represent potential phrases from word hypotheses and weak grammatical knowledge.
- WORD-SEQ-CTL: Controls the number of hypotheses that WORD-SEQ creates.
- PARSE: Attempts to parse a word sequence and, if successful, creates a phrase hypothesis from it.

Phrase Extending:
- PREDICT: Predicts all possible words that might syntactically precede or follow a given phrase.
- VERIFY: Rates the consistency between segment hypotheses and a contiguous word-phrase pair.
- CONCAT: Creates a phrase hypothesis from a verified contiguous word-phrase pair.

Rating, Halting, and Interpretation:
- RPOL: Rates the credibility of each new or modified hypothesis, using information placed on the hypothesis by other KSs.
- STOP: Decides to halt processing (detects a complete sentence with a sufficiently high rating, or notes the system has exhausted its available resources and selects the best phrase hypothesis or set of complementary phrase hypotheses as the output.)
- SEMANT: Generates an unambiguous interpretation for the information-retrieval system that the user has queried.

Figure 24. Functional descriptions of a few of the Hearsay-II KSs. (Used with permission of the ACM, copyright 1980.[31])

For an example involving glaucoma, see Figure 25. The therapy recommendations made by Casnet are determined by all three levels of description of a disease.

In Casnet, the pathophysiological states of a disease are causally related by rules of the form

$$n_i \xrightarrow{a_{i,j}} n_j$$

where n_i and n_j are states, and $a_{i,j}$ is the causal frequency with which state n_i, when present in a patient, leads to state n_j. Since the states are not mutually exclusive, the sum

$$\sum_j a_{i,j}$$

may exceed one. The rules are combined into a causal network, a semantic net that shows the courses a disease can take.

Starting states, or states in the network without any predecessors, are taken to be conditions that can arise spontaneously in the patient. Each other state is assumed to occur only as a result of the occurrences of a state immediately preceding it in the network (Figure 26).

Rules for associating tests (or observations) with states also exist in the form

$$t_i \xrightarrow{Q_{i,j}} n_j$$

where t_i is a test (or observation) or Boolean combination of tests, n_j is a state, and $Q_{i,j}$ is a number in the interval $[-1,1]$, representing the confidence with which t_i is believed to be associated with n_j. A positive result for test t_i is taken to indicate that state n_j is probably present or not present in the patient, as $Q_{i,j}$ is greater or less than zero, respectively.

In evaluating the rules, if

$$t_i \xrightarrow{Q_{i,j}} n_j \quad \text{and} \quad t_k \xrightarrow{Q_{k,j}} n_j$$

are two rules about n_j, and both tests t_i and t_k are positive, then (heuristically) the rule taken to be applicable is the one believed "most reliable," i.e., the one with the highest absolute value of Q. This value of Q is then used as the certainty factor, or CF. The CF indicates how certain the belief is that state n_j has occurred in the patient. If the CF for n_j is above an arbitrary threshold H, then state n_j is assumed to be confirmed. If the CF is below $-H$, then state n_j is assumed to be denied. Otherwise, n_j is taken to be undetermined.

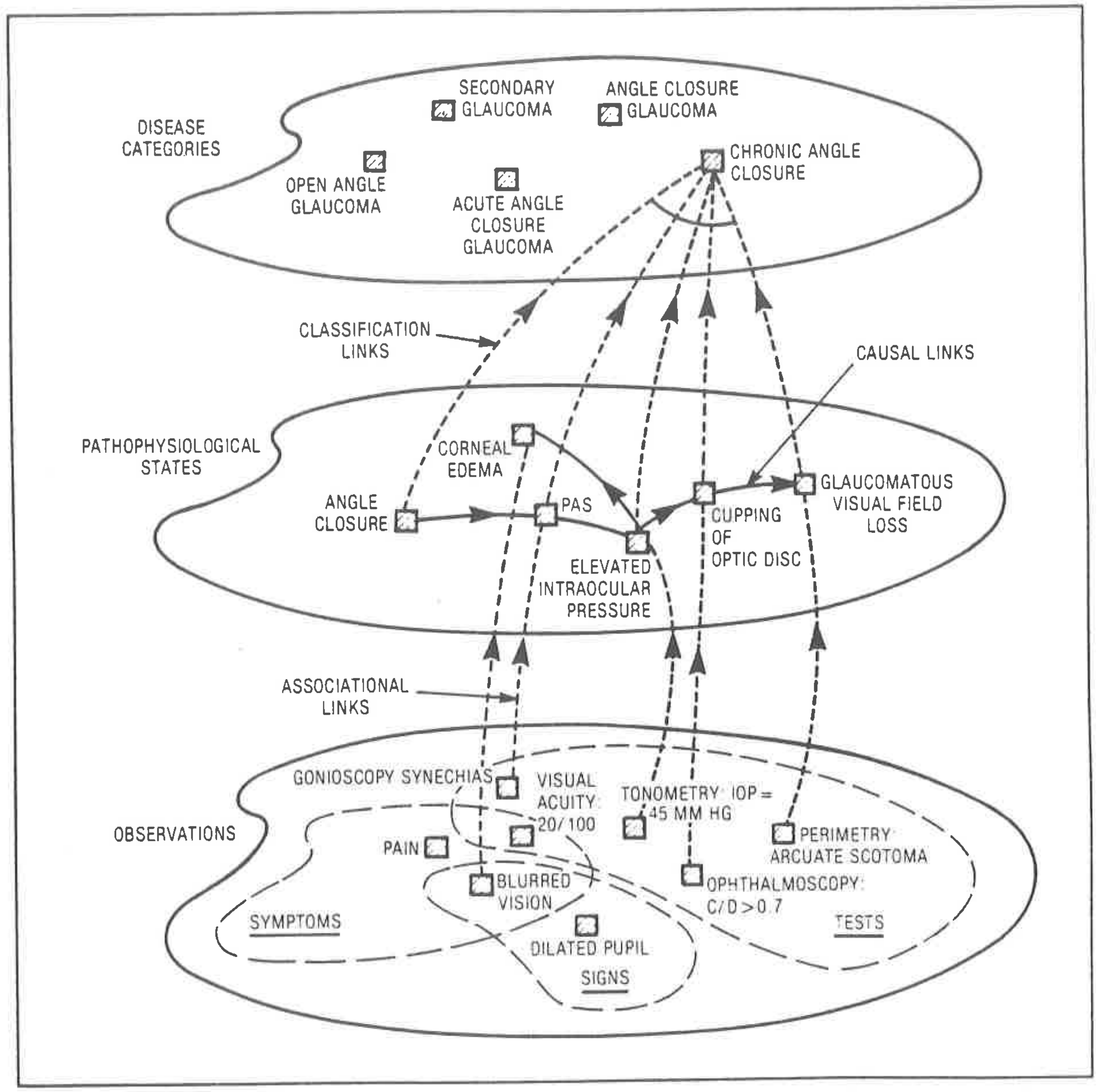
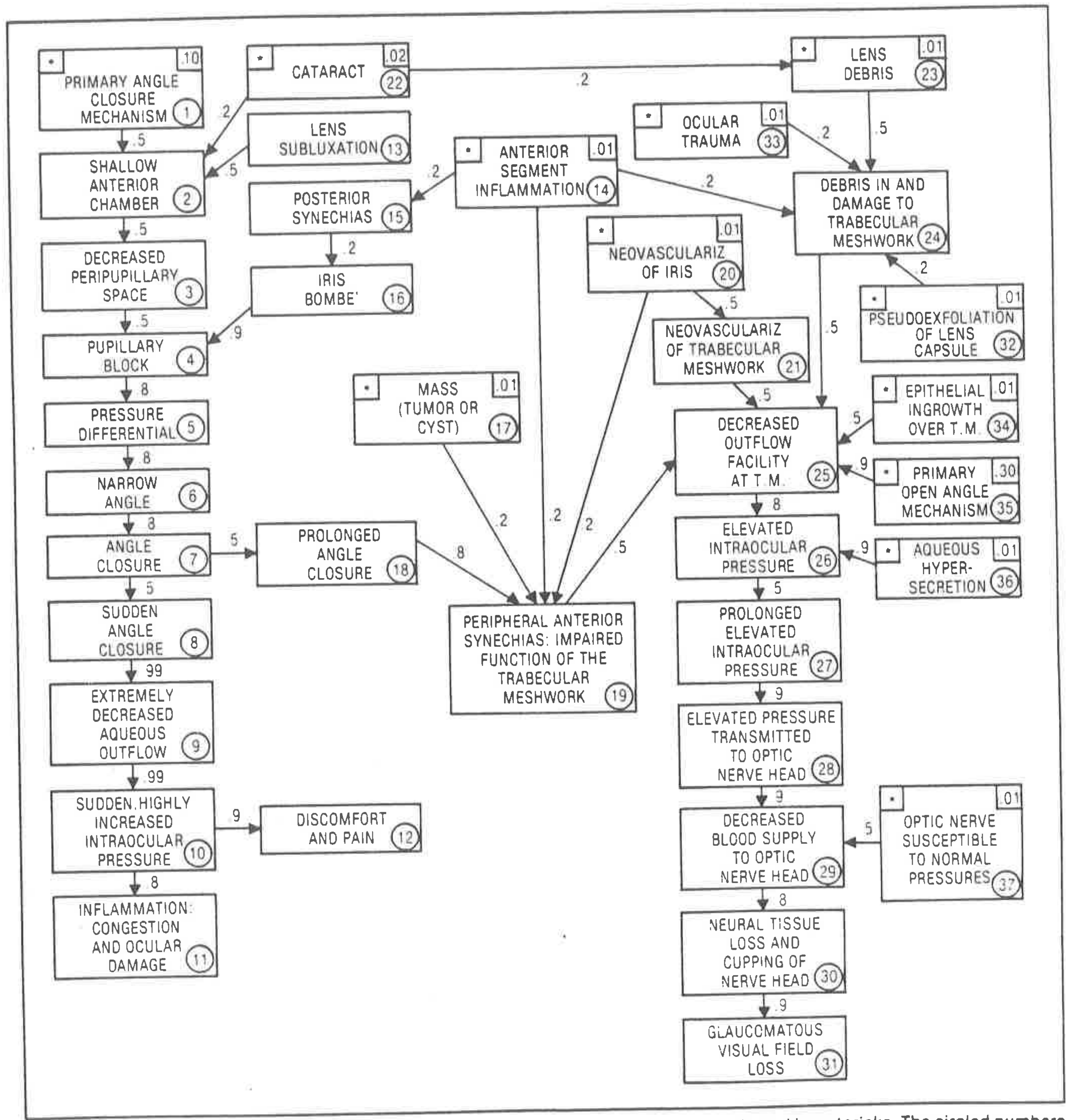


Figure 25. Three-level descriptions of a disease process used by Casnet, an expert diagnostic system, to make therapy recommendations. (Used with permission of North-Holland, copyright 1978.[3])



PRIMARY ANGLE CLOSURE MECHANISM 1 .10
CATARACT 22 .02
LENS DEBRIS 23 .01
OCULAR TRAUMA 33 .01
LENS SUBLUXATION 13
ANTERIOR SEGMENT INFLAMMATION 14 .01
SHALLOW ANTERIOR CHAMBER 2
POSTERIOR SYNECHIAS 15
DEBRIS IN AND DAMAGE TO TRABECULAR MESHWORK 24
DECREASED PERIPUPILLARY SPACE 3
IRIS BOMBE' 16
NEOVASCULARIZ OF IRIS 20 .01
PSEUDOEXFOLIATION OF LENS CAPSULE 32 .01
PUPILLARY BLOCK 4
NEOVASCULARIZ OF TRABECULAR MESHWORK 21
PRESSURE DIFFERENTIAL 5
MASS (TUMOR OR CYST) 17 .01
EPITHELIAL INGROWTH OVER T.M. 34 .01
DECREASED OUTFLOW FACILITY AT T.M. 25
NARROW ANGLE 6
PRIMARY OPEN ANGLE MECHANISM 35 .30
ANGLE CLOSURE 7
PROLONGED ANGLE CLOSURE 18
ELEVATED INTRAOCULAR PRESSURE 26
AQUEOUS HYPER-SECRETION 36 .01
SUDDEN ANGLE CLOSURE 8
PERIPHERAL ANTERIOR SYNECHIAS: IMPAIRED FUNCTION OF THE TRABECULAR MESHWORK 19
PROLONGED ELEVATED INTRAOCULAR PRESSURE 27
EXTREMELY DECREASED AQUEOUS OUTFLOW 9
ELEVATED PRESSURE TRANSMITTED TO OPTIC NERVE HEAD 28
SUDDEN, HIGHLY INCREASED INTRAOCULAR PRESSURE 10
DISCOMFORT AND PAIN 12
DECREASED BLOOD SUPPLY TO OPTIC NERVE HEAD 29
OPTIC NERVE SUSCEPTIBLE TO NORMAL PRESSURES 37 .01
INFLAMMATION: CONGESTION AND OCULAR DAMAGE 11
NEURAL TISSUE LOSS AND CUPPING OF NERVE HEAD 30
GLAUCOMATOUS VISUAL FIELD LOSS 31

Figure 26. Partial causal network for glaucoma. States with no antecedent causes are indicated by asterisks. The circled numbers correspond to the state labels n_i used in the text. (Used with permission of North-Holland, copyright 1978.[3])

Casnet allows the use of three main strategies for selecting tests for the user to perform on the patient:

- Small decision trees containing questions on related topics. Thus the questions are asked only if certain conditions are detected to hold.
- Likelihood measures over the pathophysiological states. These are computed by assuming the occurrence (or nonoccurrence) of whatever states in the network have already been confirmed (or denied), and using the causal frequency values in the network—as if they were probabilities—in the usual conditional probability formulas, to compute for each state n_j a "conditional likelihood" $W(n_j)$ that n_j can be confirmed. To use these likelihood values for test selection, each test t_i is assumed to have a cost $C(t_i)$. One approach is to select a state n_j and test t_i to maximize the ratio $W(n_j)/C(t_i)$, given that test t_i tests for state n_j. Another approach is to look only at tests t_i for which $C(t_i)$ is less than some cutoff, and select n_j and t_i to maximize $W(n_j)$, given that t_i tests for n_j.
- Calculation of the most likely cause of the disease. For this approach, the most likely cause of a disease is taken to be the starting state capable of explaining the largest number of confirmed states in the network without contradicting the denial of any denied states in the network. This is the starting state from which one can produce paths traversing the greatest number of confirmed states without traversing any denied states (if two such starting states exist, the one with the greatest likelihood measure is selected). If this state does not explain all confirmed states in the network, a second starting state is selected in the same manner to explain the remaining confirmed states, and so forth, until all confirmed states have been covered. Tests are then selected that have bearing on the selected starting states.

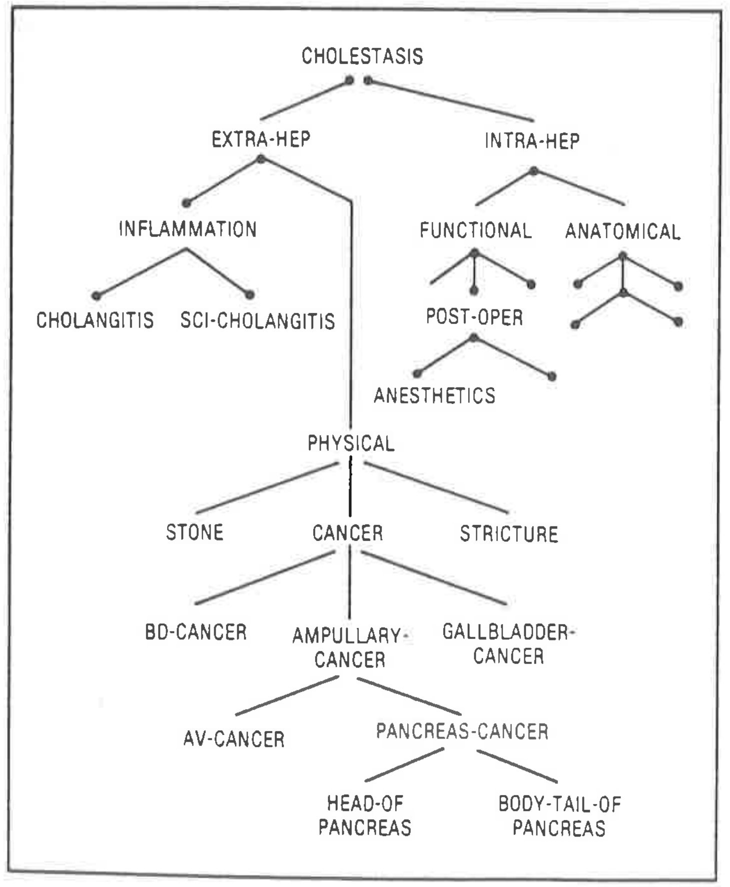


Figure 27. The conceptual structure of cholestasis used by MDX, a medical diagnostic system still under development. The MDX comprises a series of expert subsystems, one for each node of the tree. (Used with permission of the American Association for Artificial Intelligence, copyright 1979[13])

Sometimes the test results may contradict or conflict with the model, for example, when all paths into a confirmed state contain a denied state or an undetermined state ($|CF| < H$) for which $CF < 0$. Casnet includes some ways of dealing with such situations, which are described elsewhere.[8]

Pathophysiological states are associated with disease categories as follows. Let $n_1, n_2, \ldots, n_k$ be the states in a causal pathway in the network, with starting state n_1. Then we can build a classification table of the form

First unconfirmed state in the pathway	Disease category
n_2	D_1
n_3	D_2
.	.
.	.
.	.
n_k	D_{k-1}
----	D_k

where each D_i is a disease category. (The implementation in Casnet is somewhat different, because a single classification table may include information about several causal pathways. However, the use of these tables corresponds to the description given here.)

Each starting state has pointers to the classification tables relevant to the diseases caused by that starting state. These tables are used, once the most likely starting states are found, to determine the disease categories of the patient.

The classification tables can be augmented to include treatment recommendations:

First unconfirmed state in the pathway	Disease category	Recommended treatment
n_2	D_1	T_1
n_3	D_2	T_2
.	.	.
.	.	.
.	.	.
n_k	D_{k-1}	T_{k-1}
----	D_k	T_k

However, the problem of recommending treatment is usually more complicated. Instead of a single treatment T_j for each category D_j, a set $T_{j,1}, T_{j,2}, \ldots, T_{j,n}$ of several possible treatments may exist. The treatment chosen is determined by rules of the form

$$t_i \xrightarrow{P_{i,j,k}} T_{j,k}$$

which assigns preference to treatments on the basis of test results. Preferences are computed for various treatments associated with a disease category in the same way as certainty factors were computed for pathophysiological states earlier, and the treatment of highest preference is recommended.

As of 1978, Casnet (as set up to handle glaucoma) had more than 100 states, 75 classification tables, and 200 diagnostic and treatment statements. Casnet's rules must be invoked once for each eye, and special rules are used for various types of binocular comparisons. According to Kulikowski,[50] knowledge from medical textbooks alone allowed Casnet to perform at 70- to 75-percent accuracy. To get the performance above 90-percent accuracy, information from human experts was incorporated.

MDX. A medical diagnostic system still under development, MDX has an architecture designed to handle the "combinatorial explosion" problems discussed earlier. It comprises a collection of small, expert subsystems that are similar to Hearsay-II's KSs. These subsystems are organized in a hierarchy that corresponds to the taxonomy of a disease. For example, consider the classification of a medical condition called cholestasis (Figure 27). A complete implementation of MDX for cholestasis (not yet finished) would include an expert subsystem corresponding to each node of the classification tree. The communication and transfer of control among these expert subsystems, instead of being handled by global mechanisms such as Hearsay-II's blackboard and scheduler, are handled by the experts themselves, and are constricted to flow along the lines of the hierarchy.

The hierarchy of experts corresponding to the tree in Figure 27 might be entered at the root of the tree by a request for a diagnosis. When invoked, the expert at each node does some processing involving its specialized knowledge and may request information from other experts to make a diagnosis. Communication and transfer of control among experts is restricted to follow the arcs of the tree. Thus if the cholestasis expert were to send a message or request to the cholangitis expert, it would pass through the extra-hepatic and inflammation experts along the way, which would perhaps modify the message so that the cholangitis expert could understand it. If an expert is not implemented, a human at a computer terminal can take the part of that expert without affecting the rest of the system. For more information about MDX, see Chandrasekaran.[13]

Conclusions

Expert system techniques are now finding their first applications outside artificial intelligence research laboratories. Dendral[9] has been used by university and industrial chemists on a number of problems; R1[18] is being used by Digital Equipment Corporation to aid in installing computer systems; and Schlumberger's Dipmeter Advisor is being adapted for field use after several years of development in the laboratory. For the continued success of this transition from research laboratories to the real world, more progress is needed in resolving several problems.

One problem is the amount of effort it takes to build an expert system; construction sometimes takes as many as 10 to 25 man-years and costs as much as $1 to $2 million. One reason is the lack of software tools for implementing expert computer systems. Progress is currently being made in this direction with tools such as AGE, Emycin, Expert, KMS, and OPS-5, but more work is needed.

A second problem is the amount of time needed to take the knowledge from an expert in some problem domain and encode it in a knowledge base. Again this problem is due to a lack of tools for the task, but it is also due to the large gaps that still remain in our understanding of human problem solving. Some of the knowledge that an expert uses to solve a problem often cannot be made consciously accessible to him or to others without a great deal of effort.

A third issue is that since expert systems have until recently been largely experimental, we have not had to consider the need for long-term maintenance or the ability to be "friendly" to a community of users who may not have a sophisticated knowledge of computers. More attention will have to be paid to these "real-world details" if expert systems are to be useful in the long run. ■

Acknowledgments

I thank the following people who were willing to discuss problems, provide source material, and read rough drafts during the preparation of this article: Howard Bloom, Ted Chang, Bill Gevarter, and Ted Hopp.

Most of the material in this article is adapted from a report I prepared during a summer research appointment at the NBS in 1981. This work was also supported in part by NSF grant MCS-8117391 to the Laboratory for Pattern Analysis at the University of Maryland.

References

1. H. P. Nii and N. Aiello, "AGE (Attempt to Generalize): A Knowledge-Based Program for Building Knowledge-Based Programs, *Proc. Sixth Int'l Joint Conf. Artificial Intelligence,* 1979, pp. 645-655.
2. R. M. Stallman and G. J. Sussman, "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis," Vol. 9, *Artificial Intelligence,* 1977, pp. 135-196.
3. W. van Melle, "A Domain-Independent Production Rule System for Consultation Programs," *Proc. Sixth Int'l Joint Conf. Artificial Intelligence,* 1979.
4. S. M. Weiss and C. A. Kulikowski, "EXPERT: A System for Developing Consultation Models," *Proc. Sixth Int'l Joint Conf. Artificial Intelligence,* 1979, pp. 942-947.
5. J. Reggia et al., "Towards an Intelligent Textbook of Neurology," *Proc. Fourth Annual Symp. Computer Applications in Medical Care,* 1980, pp. 190-199.
6. C. L. Forgy, *The OPS5 User's Manual,* Tech. Report, Carnegie-Mellon University, 1980.
7. R. Chilausky, B. Jacobsen, and R. S. Michalski, "An Application of Variable-Valued Logic to Inductive Learning of Plant Disease Diagnostic Rules," *Proc. Sixth Annual Int'l Symp. Multiple-Valued Logic,* 1976.
8. S. M. Weiss et al., "A Model-Based Method for Computer-Aided Medical Decision-Making," *Artificial Intelligence,* Vol. 11, No. 2, 1978, pp. 145-172.
9. E. Feigenbaum, G. Buchanan, and J. Lederberg, "Generality and Problem Solving: A Case Study Using the DENDRAL Program," *Machine Intelligence 6,* D. Meltzer and D. Michie, eds., Edinburgh University Press, 1971, pp. 165-190.
10. R. Davis et al., "The Dipmeter Advisor: Interpretation of Geological Signals," *Proc. Seventh Int'l Joint Conf. Artificial Intelligence,* Aug. 1981.
11. H. E. Pople, "The Formation of Composite Hypotheses in Diagnostic Problem Solving: An Exercise in Synthetic Reasoning," *Proc. Fifth Int'l Joint Conf. Artificial Intelligence,* 1977, pp. 1030-1037.
12. J. Moses, "Symbolic Integration: The Stormy Decade," *Comm. ACM,* Vol. 14, No. 8, 1971, pp. 548-560.
13. B. Chandrasekaran et al., "An Approach to Medical Diagnosis Based on Conceptual Structures," *Proc. Sixth Int'l Joint Conf. Artificial Intelligence,* 1979, pp. 134-142.
14. N. Martin et al., "Knowledge-Base Management for Experiment Planning in Molecular Genetics," *Proc. Fifth Int'l Joint Conf. Artificial Intelligence,* 1977, pp. 882-887.
15. R. Davis, B. Buchanan, and E. Shortliffe, "Production Rules as a Representation for a Knowledge-Based Consultation Program," *Artificial Intelligence,* Vol. 8, No. 1, 1977, pp. 15-45.
16. P. E. Hart, R. O. Duda, and M. T. Einaudi, *A Computer-Based Consultation System for Mineral Exploration,* Tech. Report, SRI International, Menlo Park, Calif., 1978.
17. J. Osborn et al., "Managing the Data from Respiratory Measurements," *Medical Instrumentation,* Vol. 13, No. 6, Nov. 1979.
18. J. McDermott and B. Steele, "Extending a Knowledge-Based System to Deal with Ad Hoc Constraints," *Proc. Seventh Int'l Joint Conf. Artificial Intelligence,* 1981, pp. 824-828.
19. J. Mylopoulos, "An Overview of Knowledge Representation," *Proc. Workshop Data Abstraction, Databases, and Conceptual Modeling,* June 1980, pp. 5-12.
20. M. Minsky, "A Framework for Representing Knowledge," *The Psychology of Computer Vision,* P. H. Winston, ed., McGraw-Hill, New York, 1975, pp. 211-277.
21. N. J. Nilsson, *Principles of Artificial Intelligence,* Tioga, Palo Alto, Calif., 1980.
22. P. H. Winston, *Artificial Intelligence,* Addison-Wesley, Reading, Mass., 1977.
23. R. C. Schank, "Conceptual Dependency: A Theory of Natural Language Understanding," *Cognitive Psychology,* Vol. 3, No. 4, 1972.
24. R. C. Schank, *Conceptual Information Processing,* North-Holland, New York, 1975.
25. L. K. Schubert, "Extending the Expressive Power of Semantic Nets," *Artificial Intelligence,* Vol. 7, No. 2, 1976, pp. 163-198.
26. D. G. Bobrow and T. Winograd, "An Overview of KRL, a Knowledge Representation Language," *Cognitive Science,* Vol. 1, No. 1, 1977, pp. 3-46.
27. I. Goldstein and S. Papert, "Artificial Intelligence, Language and the Study of Knowledge," *Cognitive Science,* Vol. 1, No. 1, 1977, pp. 84-123.
28. S. E. Fahlman, *NETL: A System for Representing and Using Real-World Knowledge,* MIT Press, Cambridge, Mass., 1979.
29. R. Brachman, "On the Epistemological Status of Semantic Networks," *Associative Networks: Representation and Use of Knowledge by Computer,* N. V. Findler, ed., Academic Press, 1979, pp. 3-50.
30. Bleich, "Computer-Based Consultation," *American J. Medicine,* Vol. 53, 1972, pp. 285-291.
31. L. D. Erman et al., "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty," *Computing Surveys,* Vol. 12, No. 2, June 1980, pp. 213-253.
32. C. Hewitt, *Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot,* Tech. Report 258, MIT AI Laboratory, 1972.
33. G. J. Sussman and D. McDermott, "From PLANNER to CONNIVER—a Genetic Approach," *AFIPS Conf. Proc.,* Vol. 41-II, 1972 FJCC, pp. 1171-1179.
34. M. H. van Emden and R. A. Kowalski, "The Semantics of Predicate Logic as a Programming Language," *J. ACM,* Vol. 23, No. 4, 1976.
35. D. McDermott, "The PROLOG Phenomenon," *Sigart Newsletter,* No. 72, July 1980, pp. 16-20.
36. N. J. Nilsson, *Problem-Solving Methods in Artificial Intelligence,* McGraw-Hill, New York, 1971.
37. R. A. Kowalski, *Logic for Problem Solving,* North-Holland, New York, 1979.
38. R. V. Hogg and A. T. Craig, *Introduction to Mathematical Statistics,* third ed., Macmillan, New York, 1970.
39. Hockstra and Miller, "Sequential Games and Medical Diagnosis," *Computers and Biomedical Research,* Vol. 9, pp. 205-215.

40. J. A. Reggia, P. Y. Wang, and D. S. Nau, "Minimal Set Covers as a Model for Diagnostic Problem Solving," *Proc. Medcomp 82,* Sept. 1982, pp. 340-347.
41. R. Karp, "Reducibility Among Combinatorial Problems," *Complexity of Computer Computations,* J. W. Thatcher, ed., Plenum Press, New York, 1972, pp. 85-103.
42. J. Reggia, D. Nau, and P. Wang, *Diagnostic Expert Systems Based on a Set Covering Model,* tech. report, Computer Science Dept., University of Maryland, Oct. 1982.
43. E. W. Dijkstra, "A Note on Two Problems in Connection with Graphs," *Numerical Mathematics,* Vol. 1, 1959, pp. 269-271.
44. D. Huffman, "Impossible Objects as Nonsense Sentences," *Machine Intelligence 6,* D. Meltzer and D. Michie, eds., Edinburgh University Press, 1971, pp. 295-323.
45. M. Clowes, "On Seeing Things," *Artificial Intelligence,* Vol. 2, No. 1, 1971, pp. 79-116.
46. A. Newell and H. A. Simon, "GPS, a Program that Simulates Human Thought," *Computers and Thought,* E. A. Feigenbaum and J. A. Feldman, eds., McGraw-Hill, New York, 1963.
47. R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence,* Vol. 2, No. 3, 1971, pp. 189-208.
48. D. L. McCracken, "Representation and Efficiency in a Production System for Speech Understanding," *Proc. Sixth Int'l Joint Conf. Artificial Intelligence,* 1979, pp. 556-561.
49. P. E. Hart, "Directions for AI in the Eighties," *Sigart Newsletter,* No. 79, Jan. 1982, pp. 11-16.
50. C. Kulikowski, personal communication, 1981.

Dana S. Nau is an assistant professor of computer science at the University of Maryland. His previous experience includes research appointments at IBM Research in New York and at the National Bureau of Standards in Maryland. During the latter appointment, he made recommendations for the use of expert computer system techniques in automated manufacturing.

Nau has published papers in both biomathematics and artificial intelligence. His current research interests include searching and problem-solving methods in artificial intelligence, and expert computer systems.

Nau received a BS in applied mathematics from the University of Missouri at Rolla in 1974, and received an AM and PhD in computer science from Duke University in 1976 and 1979, respectively, where he attended on NSF and James B. Duke graduate fellowships.