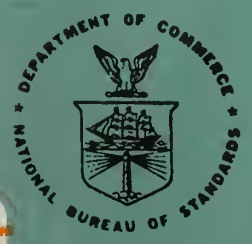
NBSIR 81-2466

Expert Computer Systems, and Their Applicability to Automated Manufacturing

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
National Engineering Laboratory
Center for Manufacturing Engineering
Industrial Systems Division
Metrology Building, Room A127
Washington, DC 20234

February 1982

Issued October 1982



U.S. DEPARTMENT OF COMMERCE
NATIONAL BUREAU OF STANDARDS

NBSIR 81-2466

EXPERT COMPUTER SYSTEMS, AND THEIR APPLICABILITY TO AUTOMATED MANUFACTURING

Dr. Dana S. Nau*

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
National Engineering Laboratory
Center for Manufacturing Engineering
Industrial Systems Division
Metrology Building, Room A127
Washington, DC 20234

February 1982

Issued October 1982

*Faculty Appointee from the University of Maryland.

U.S. DEPARTMENT OF COMMERCE, Malcolm Baldrige, *Secretary*
NATIONAL BUREAU OF STANDARDS, Ernest Ambler, *Director*

EXPERT COMPUTER SYSTEMS, AND THEIR APPLICABILITY TO AUTOMATED MANUFACTURING

by

Dana S. Nau

Computer Science Department
University of Maryland
College Park, MD 20742

ABSTRACT

This paper contains two main parts: a tutorial on techniques used in expert systems, and some recommendations for an automated process planning system for the Automated Manufacturing Research Facility at the National Bureau of Standards (NBS).

The tutorial portion of the paper consists of Sections 2, 3, and 4. Sections 2 and 3 discuss AI problem solving and knowledge representation techniques. Section 4 describes ways in which these techniques have been used to build computer systems which achieve a high level of performance on problems which normally require significant human expertise for their solution.

Section 5 contains a summary of the activities required for process planning in the Automated Manufacturing Research Facility (AMRF) at NBS, and recommendations for how to how to accomplish these activities. Section 6 contains recommendations for how an expert system could be designed to perform a process planning activity called process selection.

TABLE OF CONTENTS

Page

Abstract 1

1. Introduction. 3

2. Problem Solving 5
   2.1. State-Space Search 5
   2.2. Problem Reduction 9
   2.3. Examples 12

3. Knowledge Representation 19
   3.1. Declarative knowledge 19
   3.2. Procedural Knowledge 24

4. Expert Systems 28
   4.1. MYCIN 29
   4.2. CASNET 34
   4.3. Hearsay-II 40
   4.4. DENDRAL and MDX 44

5. Process Planning Requirements for the AMRF 47
   5.1. Process Planning Activities 47
   5.2. Geometric Modelling Requirements 58

6. Recommendations for Process Selection 62
   6.1. Existing Systems 62
   6.2. The Recommended System 64
   6.3. Scenario for an Automated Process Selection System . 67

Acknowledgements 78

References 79

Figures 83

EXPERT COMPUTER SYSTEMS, AND THEIR APPLICABILITY TO AUTOMATED MANUFACTURING

by

Dana S. Nau

Computer Science Department
University of Maryland
College Park, MD 20742

ABSTRACT

This paper contains two main parts: a tutorial on techniques used in expert systems, and some recommendations for an automated process planning system for the Automated Manufacturing Research Facility at the National Bureau of Standards (NBS).

The tutorial portion of the paper consists of Sections 2, 3, and 4. Sections 2 and 3 discuss AI problem solving and knowledge representation techniques. Section 4 describes ways in which these techniques have been used to build computer systems which achieve a high level of performance on problems which normally require significant human expertise for their solution.

Section 5 contains a summary of the activities required for process planning in the Automated Manufacturing Research Facility (AMRF) at NBS, and recommendations for how to how to accomplish these activities. Section 6 contains recommendations for how an expert system could be designed to perform a process planning activity called process selection.

TABLE OF CONTENTS

Page

Abstract 1

1. Introduction. 3

2. Problem Solving 5
   2.1. State-Space Search 5
   2.2. Problem Reduction 9
   2.3. Examples 12

3. Knowledge Representation 19
   3.1. Declarative knowledge 19
   3.2. Procedural Knowledge 24

4. Expert Systems 28
   4.1. MYCIN 29
   4.2. CASNET 34
   4.3. Hearsay-II 40
   4.4. DENDRAL and MDX 44

5. Process Planning Requirements for the AMRF 47
   5.1. Process Planning Activities 47
   5.2. Geometric Modelling Requirements 58

6. Recommendations for Process Selection 62
   6.1. Existing Systems 62
   6.2. The Recommended System 64
   6.3. Scenario for an Automated Process Selection System . 67

Acknowledgements 78

References 79

Figures 83

1. INTRODUCTION

The field of Artificial Intelligence (AI) is in an exciting position at present. It is only recently that AI has advanced to the point that AI projects are accomplishing useful practical results. Most of these results involve the design and utilization of expert systems, problem-solving computer systems which can reach a level of performance comparable to that of an expert human being in some specialized problem domain.

All expert systems known to this author are knowledge-based systems: their high level of performance is based on the use not only of declarative knowledge about the problem domain (i.e., the data to be manipulated), but also of knowledge about problem solving that is tailored specifically to the problem domain. An expert system typically consists of (1) a "knowledge base" of declarative knowledge, (2) an encoding of problem-specific problem solving knowledge, (3) a control strategy for manipulating the problem solving knowledge and the knowledge base, and (4) interfaces which allow a human user to conult the system, get it to solve problems, and explain its solutions.

Sections 2, 3, and 4 comprise a tutorial on techniques used in expert systems. Since expert systems are a combination of AI problem solving techniques and AI knowledge representation techniques, the tutorial includes expositions of these areas as well. Section 2 discusses problem solving techniques, Section 3 discusses knowledge representation, and Section 4 gives several examples of the workings of expert systems.

Sections 5 and 6 are a discussion of process planning in the Automated Manufacturing Research Facility (AMRF) at NBS, and the possible applicability of expert system techniques to this task. Section 5 discusses the activities necessary for process planning, and makes suggestions for how these activities might be accomplished. Section 6 focuses specifically on a process planning activity called process selection, and describes how an interactive expert system might be designed for this task.

Section 4, as an exposition of various expert systems techniques, has somewhat the flavor of a survey of current expert systems. However, this is mainly for tutorial purposes: considered as a survey, the section is not at all complete. I apologise in advance to those researchers whose projects and ideas have been omitted from this section, either inadvertently or due to space and time constraints.

2. PROBLEM SOLVING

2.1. State-Space Search

Many problems in artificial intelligence are solved by searching through a space of "states of the world" to find a path from some initial state (the state of the world before any work has been done to solve the problem) to any of one or more goal states (states in which the problem has been solved). This search is done by applying operators (which depend on the particular problem domain) to transform states into other states.

In a forward search, one starts with the initial state and applies operators to try to find a solution path to a goal state. In a backward search, one starts with the goal state (or states), and applies the inverses of the operators to try to find a solution path. The state space corresponds to a directed graph in which the states are the nodes of the graph, and in which there is an arc from some node A to another node B if there is an operator which transforms state A into state B.

The possible objectives of searching through the state space might be (1) to find any goal, (2) to find the shortest path to a goal, (3) to find the path of least cost or highest cost, according to some cost criterion, or possibly other objectives.

The organization of state-space search procedures is frequently as shown below:

CONTROL STRUCTURE: knowledge about how to do what can be done.
Keeps track of states.
Decides which operators to apply.
Decides whether to backtrack.

OPERATORS: knowledge about what can be done.
They govern what the search space looks like.

STATES: knowledge about what has been done.
The basic elements of the search space.

2.1.1. Example

As an example, let us consider a well-known children's game called the 15-puzzle. This puzzle consists of a square frame containing fifteen square tiles and a square hole. The tiles are numbered from 1 to 15, and may be moved around by moving a tile into the hole if the tile and hole are adjacent (thus creating a new hole). The goal of the game is to get the squares in order (see Figure 2.1).

The 15-puzzle may be modeled as a state-space search problem as follows (Nilsson [1971], pp. 4-7).

1. Every board configuration is a state;
2. the operators are

   UP: move the hole up

   DOWN: move the hole down

   RIGHT: move the hole to the right

   LEFT: move the hole to the left;
3. the initial state is whatever state the puzzle is received in;
4. the goal state is as shown in Figure 2.1.

   The most likely objective would be to find any path to a goal.

Note that more than one operator may be applicable to a state, but not all four operators are always applicable. Part of the state space is illustrated in Figure 2.2.

The above is only one of several possible state-space representations for the 15-puzzle. For example, one may instead choose the set of operators Li, Ri, Ui, Di, i=1,2,...,15, where

Li means "move tile i left",

Ri means "move tile i right",

Ui means "move tile i up",

and Di means "move tile i down".

In this case, there are sixty operators rather than four, but at most four are applicable at any one time.

2.1.2. Searching the State Space

One can easily write a nondeterministic state-space search algorithm:

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

The nondeterministic selection is done as in a nondeterminstic Turing machine: one may think of several copies of the program being created, 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. Several different techniques have been proposed.

One class of techniques 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  /* 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 ops */
        val = backtrack (r(s), concatenate(path,r))
        IF val is not "fail" THEN RETURN concatenate(r,val)
    END
    RETURN "fail"
END backtrack
```

Another class of procedures is referred to by Nilsson [1980] as graph-searching procedures. Such procedures explore several paths simultaneously, keeping track of several "current states". Some paths may be explored faster than others, depending on the particular procedure. Examples of such procedures are

1. breadth-first search, in which all paths are searched at the same speed;
2. least-cost-first search (e.g. Dijkstra's algorithm [###]), in which at each iteration of the procedure, the path which

currently has the least accumulated cost (according to some criterion) is extended;

3. heuristic search, in which various heuristic criteria are used to determine which path or paths to extend next.

2.2. Problem Reduction

An alternative to using state space search for problem solving is to use a technique known as problem reduction. Here the problem to be solved is partitioned or decomposed into subproblems, each of which can be solved separately, in such a way that combining together the solutions to the subproblems will yield a solution to the original problem. The subproblems may each be decomposed into sub-subproblems, which may be even further decomposed, until primitive problems are generated which can be solved directly.

As an example, consider the 15-puzzle again. As illustrated in Figure 2.3, the problem of getting from the initial state to the goal state may be decomposed into the following four subproblems:

1. the problem of getting the first row in order;
2. the problem of getting the first two rows in order, given that the first row is in order;
3. the problem of getting the first three rows in order, given that the first two rows are in order;
4. the problem of getting all four rows in order, given that the first three rows are in order.

The solutions to these four subproblems provide a solution to the original problem simply by concatenating them together.

Subproblem 1, 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, there may be more than one way to decompose a problem. For example, the 15-puzzle could have been decomposed into the subproblems of getting the four columns correct, rather than the four rows. We may graphically represent all possible decompositions of a problem in an problem reduction graph or AND/OR graph (see Figure 2.4), in which each OR branch represents a choice of several alternate decompositions, and each AND branch represents a particular way of decomposing a problem.

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 which yields subproblems all of which are solvable. To solve each of these subproblems, we must choose decompositions which 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. One may think of the nodes in N as the set of solvable primitive problems. A solution graph from n to N is defined as follows.

1. Suppose n is in N. Then the solution graph is simply n itself.
2. Suppose n is not in N, and n is terminal. Then there is no solution graph from n to N.
3. Suppose n is not in N, n is not terminal, and the branch from

n to its children n[1], n[2], ..., n[k] is an OR branch. Then n is solvable if any one of its children are solvable. For every i such that there is a solution graph G[i] from n[i] to N, the union of G[i] with the node n and the arc (n, n[i]) is a solution graph from n to N. Thus there may be several solution graphs from n to N.

4. Finally, suppose n is not in N, n is not terminal, and the branch from n to its children n[1], n[2], ..., n[k] is an AND branch. Then n is solvable only if every one of its children is solvable. If for i=1,2,...,k, there is a solution graph G[i] from n[i] to N, then the union of all of the G[i], the node n, and the arcs (n, n[i]), i=1,...,k, is a solution graph from n to N.

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

Suppose Q is a problem which may be solved using either state-space search or problem reduction. Let S be the state-space graph for Q, and let R be the problem reduction graph for Q. R is often considerably smaller than S (see Figure 2.5). However, since the solution to R is a subgraph rather than a path, a typical search procedure P(R) for R will usually be much more complicated than a typical search procedure P(S) for S. In fact, the number of consecutive process states necessary for P(R) to solve Q may be more than the number necessary for P(S).

2.3. Examples

We now discuss two examples of the application of search techniques to problem solving.

2.3.1. Huffman-Clowes Labeling

Propagation of constraints is a state-space problem solving technique in which the set of possible solutions becomes further and further constrained by the application of local constraints on the structure of pieces of the solution, until only one (or some small number) of possible solutions is left.

One example of propagation of constraints is Huffman-Clowes labeling [Huffman 1971] [Clowes 1971] [Winston 1977]. This is a technique for analyzing two-dimensional line drawings of three-dimensional objects composed of flat surfaces. The technique can determine what part of the drawing is the object and what part is the background, which intersections of surfaces are convex and which are concave, and (in some cases) whether the drawing represents a real three-dimensional object. The Huffman-Clowes labeling technique is explained below, along with an explanation of how it can be implemented as a state space search.

The restrictions on the applicability of the technique are--

1. The drawing must be a simple black-and-white line drawing of a single object, without any indication of coloration, illumination, shadows, cracks, or texture of surfaces.
2. 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).

3. No "pathological" points of view of the object are permitted; i.e., the point of view given in the drawing must be such that if the point of view is changed by a very small amount, the percieved character of the vertices will remain the same. This prohibits, for example, the point of view shown in Figure 2.6a, because in this drawing two surfaces appear to meet which in fact do not meet (see Figure 2.6b).
4. If the result of the analysis is a contradiction, then it is certain that the line drawing cannot represent a real three-dimensional object. However, if the analysis does not find a contradiction, it does not necessarily mean that the drawing cannot be realized as a three-dimensional object.

A Huffman-Clowes analysis of such a drawing consists of putting a label on each line to indicate whether it is a convex or concave intersection of surfaces, or a border of the object (see Figure 2.7). Given the restrictions cited above, it turns out that only four types of vertices are physically possible in any line drawing (see Figure 2.8), and that only sixteen combinations of labels are physically possible for the lines coming into these vertices (Figure 2.9).

The "propagation of constraints" idea works as follows: The lines around an perimeter of the object must be labeled with clockwise arrows (see Figure 2.10a), and given the sixteen possible label combinations, this restricts the possible labels on some of the other lines. For example, the vertex shown in Figure 2.11a must be labeled as shown in Figure 2.11b, because there is no other way to label the unlabeled line that yields a

legal combination of labels.

Once the labels have been assigned to some of the other lines (see Figure 2.10b), this determines the labels on yet more lines, until the entire drawing is labeled (Figure 2.6). If not all vertices can be assigned legal combinations of labels, then the object cannot be a real three-dimensional object satisfying the restrictions given above. For an example, see Figure 2.12.

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 analyzed, with one or more lines labeled. The initial state consists of the drawing in which no lines are labeled except for clockwise arrows going around the perimeter of the object. Each of the sixteen 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 2.13a corresponds to an operator which applies to any one of the label combinations shown in Figure 2.13b, and whose application produces the label combination shown in Figure 2.13a.

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
```

At the end of the algorithm's operation, if not all vertices have legal combinations, then the object is not a real three-dimensional object. However, a legal labeling for the object does not guarantee that the object can be real (Winston [1977], p. 60).

2.3.2. STRIPS

STRIPS (Stanford Research Institute Problem Solver) is a problem-solving system which was originally developed to produce plans of action for a robot. STRIPS works by representing the "current state of the world" as a conjunction of predicates (statements containing variables), and maintaining a stack of goals and subgoals to be solved. The actions that STRIPS can perform are represented by operators that transform states into other states by adding and removing predicates. The search performed by STRIPS can be thought of as a combination of state space search and problem reduction.

We will discuss STRIPS in the context of an example found in Nilsson [1980], Chapter 7. Suppose STRIPS is working in a

world consisting of three blocks and a robot arm which can pick up blocks and move them around. One might represent the state of the world given in Figure 2.14 by the set of predicates given below:

Predicate	Meaning
CLEAR(B)	The top of block B is clear.
CLEAR(C)	The top of block C is clear.
ON(C,A)	Block C is on top of block A.
HANDEMPTY	The robot's hand is empty.
ONTABLE(A)	Block A is on the table.
ONTABLE(B)	Block B is on the table.

Obviously, the predicates above do not provide a complete description of the world (nor is a complete description even possible). However, they provide all the information about the current state of the world relevant to the problem to be solved.

Suppose we give STRIPS the goal of creating a stack of three blocks: ON(A,C) & ON(C,B). We must also give STRIPS operators that will allow it to achieve this goal

Each STRIPS operator consists of three parts: a list of preconditions, a delete list, and an add list. The preconditions are predicates, all of which must be satisfied before the operator can be applied. The predicates may contain variables, in which case the predicates must hold for some instantiation (possible set of values) of the variables before the operator can be applied. The delete and add lists, respectively, are sets of predicates to be deleted from and added to the current state when the operator is applied. In applying the operator, the same instantiations of variables are made in the precondition list,

delete list, and add list.

There are four operators in our example, as given below.

1. pickup(x)
   preconditions: ONTABLE(x), CLEAR(x), HANDEMPTY
   delete list: ONTABLE(x), CLEAR(x), HANDEMPTY
   add list: HOLDING(x)

2. unstack(x,y)
   preconditions: ON(x,y), CLEAR(x), HANDEMPTY
   delete list: ON(x,y), CLEAR(x), HANDEMPTY
   add list: HOLDING(x), CLEAR(y)

3. putdown(x)
   preconditions: HOLDING(x)
   delete list: HOLDING(x)
   add list: ONTABLE(x), CLEAR(x), HANDEMPTY

4. stack(x,y)
   preconditions: HOLDING(x), CLEAR(y)
   delete list: HOLDING(x), CLEAR(y)
   add list: ON(x,y), ClEAR(x), HANDEMPTY

For further explanation of the operation of STRIPS, we quote from Nilsson [1980], pp. 298 and 302.

> STRIPS maintains a "stack" of goals and focuses its problem-solving effort on the top goal of the stack. Initially, the goal stack contains just the main goal. Whenever the top goal in the goal stack matches the current state description, it is eliminated from the stack ... . Otherwise, if the top goal in the goal stack is a compound goal, STRIPS adds each of the component goal [predicates], in some order, above the compound goal in the goal stack. The idea is that STRIPS works on each of these component goals in the order in which they appear on the stack. When all of the component goals are solved, it reconsiders the compound goal again, re-listing the components on the top of the stack if the compoound goal does not match the current state description. This reconsideration of the compound goal is the (rather primitive) safety feature that STRIPS uses to deal with the interacting goal problem. If solving one component goal undoes an already solved component, the undone goal is reconsidered and solved again if needed.
>
> When the top (unsolved) goal on the stack is a [single predicate], STRIPS looks for an [operator] whose add list contains a [predicate that can be instantiated to match it. The instantiated name of this operator] then replaces the [predicate] at the top of the stack. On top of the [name of the operator] is then added the [instantiation] of its

precondition formula, P. If P is compound and does not match the current state description, its components are added above it, in some order, on the stack.

When the top item on the stack is an [operator], it is because the precondition formula of this [operator] was matched by the current state description and removed from the stack. Thus, the [operator] is applicable, and it is applied to the current state description and removed from the top of the stack. The new state description is now used in place of the original one, and the system keeps track of the [operator] that has been applied for later use in composing a solution sequence.

. . .

Several decisions must be made by the control component of the STRIPS system. We'll mention some of these briefly. First, it must decide how to order the components of a compound goal above the compound goal in the goal stack.

. . .

When (existentially quantified) variables occur in the goal stack, the control component may need to make a choice from among several possible instantiations. We can assume that a different successor can be created for each possible instantiation.

When more that one STRIPS [operator] would achieve the top goal on the goal stack, we are again faced with a choice. Each relevant rule can produce a different successor node.

The operation of STRIPS on our example is illustrated in Figures 2.15 and 2.16, which are taken from Nilsson [1980], pp. 300-301.

3. KNOWLEDGE REPRESENTATION

The knowledge needed by a problem solving system can be partitioned into two types of knowledge: declarative knowledge (as of facts, or of the current state of the problem domain), and procedural knowledge (knowledge of how to solve the problem). We briefly discuss several types of representation of each of these kinds of knowledge. For a further discussion of issues in knowledge representation, the reader is referred to Mylopoulos [1980].

3.1. Declarative knowledge

3.1.1. First-Order Predicate Calculus

Declarative knowledge may be thought of as the knowledge of facts about the state of the world in which a problem is being solved. One well-known way to represent such knowledge is by means of formulas in first-order predicate calculus (FOPC). 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) -> R(x,z))

for the statement that the relation R is transitive.

3.1.2. Frames

Another way of representing knowledge is in terms of frames [Minsky 1975] [Winston 1977, Chapter 7] [Nilsson 1980, Chapter 9], in which all the knowledge about a particular object or event is stored together. Such a representation cannot represent any more concepts than FOPC can, but the organization of the 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. For example, a frame for a book might be a data structure including fields or slots for the author, title, and publication date of the book, as well as the number of pages, color of the cover, etc. 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.

3.1.3. Semantic Networks

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

3.1.4. Example

As an example, 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 FOPC 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)
```

To put this information into frames, we first need to decide what kinds of frames to use. Using Shank's notion of conceptual dependency [Schank 1975] [Winston 1977, pp. 190-191], we can consider "take" to be a transfer of possession in which Bill transfers the book from Margaret to himself. The frame for a transfer of possession might be

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

Constructing similar frames for the other objects and filling in all the slot values might yield the following:

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

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
```

All of the above information can be translated directly back into FOPC, but the formulas look somewhat different than before. 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 3.1.

As can be seen, collections of primitive facts can be represented quite nicely using frames and semantic nets. However, the representation of 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 to do this, see Nilsson [1980], Chapter 9, or Schubert [1976].

The main advantage of frames or semantic nets over logical representation is that for each object, event, or concept, the relevant information is all collected together. This is convenient for accessing and manipulating the information, and also allows for a convenient way to create default values when information about an object or event is not explicitly given. For example, in the frame for a book, one 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 that this is a guess. 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 [Bobrow et al. 1977], FRL [Goldstein et al. 1977], NETL [Fahlman 1979], and KLONE [Brachman 1979].

3.2. Procedural Knowledge

3.2.1. Programs and Pattern-Invoked Programs

Procedural knowledge is the knowledge of what can be done to change the problem states in order to solve a problem. Obviously, such knowledge can be represented as computer programs in the usual sense of the term, and this is undoubtedly the best way to represent procedural knowledge when the procedure is well-understood. For some problems, however, the precise way to solve a problem may not be known. For example, the search procedures discussed in Section 2 typically must try several alternate paths before finding one which leads to a goal node.

In such situations, it is sometimes useful to encode knowledge in the form of pattern-invoked programs, which are invoked automatically whenever certain conditions are found to hold in the data representing the current problem state. The STRIPS operators in Section 2.3.2 are simple examples of such programs, and the invoking mechanism for them is the STRIPS control strategy.

Several different programming languages (such as PLANNER [Hewitt 1972], CONNIVER [Sussman et al. 1971], and PROLOG [van

Emden et al. 1976] [McDermott 1980] have been written which allow for pattern-directed invocation in one form or another, and which often include features such as automatic backtracking if an attempted solution fails.

3.2.2. Production Rules

One type of pattern-invoked program which is of particular interest is the production rule, which is a degenerate program of the form

IF condition THEN primitive action

where the condition is usually a conjunction of predicates testing properties about the current state, and the primitive action is some simple action which changes the current state. Both the labeling operators of Section 2.3.1 and the STRIPS operators of Section 2.3.2 can be thought of as production rules.

A problem solving system in which the domain-dependent procedural knowledge is represented using production rules is known as a production system. The organization for production systems is similar to the organization of state space search procedures given in Section 2.1, with the operators replaced by production rules:

CONTROL STRUCTURE: knowledge about how to do what can be done.
Keeps track of problem states.
Decides which rules to apply.

PRODUCTION RULES: determine about what can be done.
If applied, they produce changes to the database.

DATABASE: the declarative knowledge about the problem.

3.2.3. Logical Representation and PROLOG

Procedural knowledge can also be represented in FOPC, provided suitable interpretations are made of the FOPC formulas. The programming language PROLOG [McDermott 1980] is an example of such an approach. PROLOG makes use of the fact that a formula such as

B1 & B2 & ... & Bn -> A

can be thought of either as the logical statement that A is true whenever B1, B2, ..., Bn are true, or as a procedure for producing a state satisfying condition A. The three basic statements in PROLOG are as follows:

statement	meaning
:- A.	A is a goal
A.	A is an assertion
A :- B1, ..., Bn.	B1 & ... & Bn -> A

A and all of the B's must be predicates, and all variables are considered to be universally quantified (i.e., 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.

This corresponds to the AND/OR graph shown in Figure 3.2. The solution of a problem presented as a set of PROLOG statements proceeds by doing a depth-first search of the corresponding AND/OR graph, until an instantiation of a set of assertions is found which provides a solution graph.

As an example (due to McDermott [1980]), 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 | [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.
2. If the concatenation of L1 with L2 is L3, then the concatenation of [X | L1] with L2 is [X | L3].

To use the above PROLOG program to "set" the variable A to the concatenation of [a,b] with [c,d], one 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 recursive call ":- append([],[c,d],L3)" is evaluated. Clause 1 applies, giving L3 = [c,d]. Returning from the recursive calls, we have A = [a | [b | [c,d]]] = [a,b,c,d], as desired.

4. EXPERT SYSTEMS

Feigenbaum [1980, p. 1], describes an expert system as

> an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution. The knowledge necessary to perform at such a level, plus the inference procedures used, can be thought of as a model of the expertise of the best practitioners of that field.

Expert systems have been developed for a number of different problem domains. Below is an incomplete list of such systems.

system name	expertise	reference
DENDRAL	hypothesizing molecular structure from mass spectrograms	Feigenbaum et al. [1980]
MACSYMA	mathematical formula manipulation	Moses [1975]
MYCIN	medical consulting	Davis et al. [1977]
CASNET	" "	Weiss et al. [1978]
INTERNIST	" "	Pople [1977]
MDX	" "	Chandrasekaran et al. [1979]
KMS	" "	Reggia et al. [1981]
PUFF	" "	Osborn et al. [1979]
AQ11	diagnosis of plant diseases	Chilausky et al. [1976]
PROSPECTOR	predicting likely ore deposit locations	Hart et al. [1978]
MOLGEN	planning DNA experiments	Martin et al. [1977]

In addition, there are several computer systems being developed to provide tools for creating expert systems for various problem domains. These systems include EMYCIN [van Melle 1979], EXPERT

[Weiss et al. 1979], KMS [Reggia et al. 1981], and AGE [Nii et al. 1979]. Such systems typically provide formats for representing both procedural knowledge and declarative knowledge, a control structure or control structures for manipulating this knowledge, and an interface for interaction with a human user.

This section contains overviews of several expert systems. The systems described were chosen to illustrate the variety of approaches useful in building expert systems. Section 4.1 is a description of MYCIN. Section 4.2 describes CASNET. Section 4.3 describes Hearsay-II, a speech understanding system (Erman, et al. [1980]). Section 4.4 contains a few comments about DENDRAL and MDX.

4.1. MYCIN

MYCIN [Davis et al. 1977] is a production system, written in LISP, to diagnose infectious diseases and recommend treatment for them. MYCIN interacts with the user, asking questions about the symptoms of the disease, to determine a diagnosis.

In medical diagnosis, the knowledge of relevant facts and the causal associations between symptoms and diseases may not be completely certain. MYCIN handles this problem by associating with each fact and production rule a "certainty factor" (CF) indicating the certainty with which it is believed to be true. The CF is a number in the interval [-1,1]. Positive and negative CF's indicate a predominance of confirming or disconfirming evidence, respectively. CF's of 1 or -1 indicate absolute knowledge.

A typical MYCIN production rule is the following [Davis et al. 1977, p. 21]:

```
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 identity of the organism is bacteroides.
```

$AND, a multi-valued AND operation, is used to manipulate CFs as described below.

In MYCIN, declarative knowledge is represented in the form of 4-tuples such as the following (Davis et al. [1972], p. 22):

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

When MYCIN evaluates its production rules, it does so as follows.

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

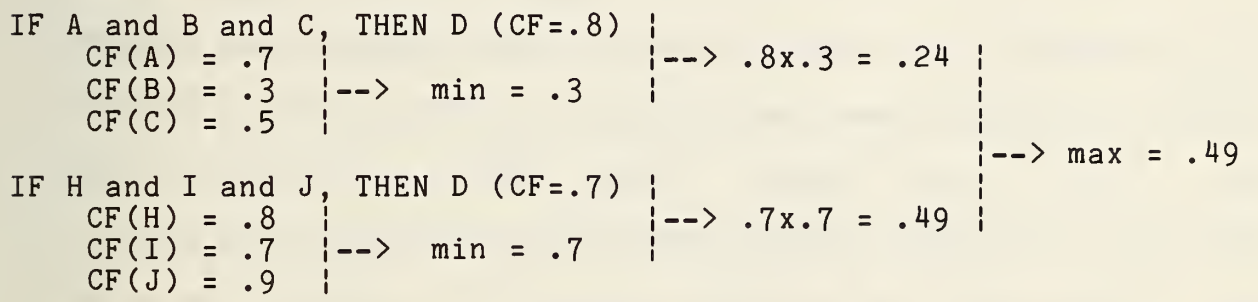
This is illustrated in the following example.

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 CF's .7, .3, .5, .8, .7, and .9, respectively. Thus the computation, as shown in the diagram below, produces a CF of .49 for D.



In the above example, the 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 4.1. 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 furthermore that it knows that the values of B, F, G, H, I, and E are laboratory data which are determined by asking the user for their values. The AND/OR graph corresponding to these rules is shown in Figure 4.2. 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 "prove B=b" and "prove C=c" but rather "find the value of B" and "find the value of C". This enables the system to maintain 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 evaluated again if another rule is ever encountered which requires information about B or C in its premises.

Note also that every rule relevant to a particular goal must be invoked unless the value of the goal can be established with CF 1 or -1. However, if one of the premises of a rule is already known to have CF -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 all 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 diagnosis 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 to MYCIN, he can make the appropriate change or addition. Since the rules are invoked automatically whenever they have bearing on a goal, no other change need be made to the system to assure that a rule will be used.

The advantages of the architecture of MYCIN are the following:

1. Each production rule is completely modular and independent of the other rules. Thus it is easy to change or add to MYCIN's knowledge base.
2. 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.
3. Each rule represents a small, isolated chunk of knowledge. Thus a physician familiar with the system may be able to formulate new rules for the system if necessary.

The disadvantages are as follows:

1. Sometimes it is not completely natural to represent a piece of knowledge about a disease as a production rule.
2. Since MYCIN searches backwards from the goal of diagnosing and treating a disease to the known data about the patient, it is not always easy to map a sequence of desired actions or tests into a set of production rules whose invocation will provide that sequence.
3. Sometimes it is desirable to write rules in a more general

format than that given earlier. For example, one might want a rule saying "for each organism such that ..., conclude ... ." A few such rules have been put into the system, but this causes other problems. For example, the question-answering system cannot explain the invocation of these rules very well.

As of the date of our reference on MYCIN (Davis et al. [1977]), MYCIN could perform diagnoses with an agreement of about 72% with medical experts. This figure has probably improved since.

It is possible to remove the knowledge base of MYCIN and substitue a set of rules from another domain. This has been done, for example, for the problem of diagnosing lung diseases, yielding a program called PUFF [Osborn et al. 1979].

4.2. CASNET

CASNET [Weiss et al. 1978] is a computer system for medical decision-making which uses production rules, semantic nets, and some other features as well. CASNET is constructed using the following three levels of description of a disease:

1. Observations (or tests). These consist of disease symptoms, laboratory tests, etc., and form the direct evidence that a disease is present.
2. Pathophysiological states. These are the internal conditions assumed to occur in the patient. "States" as used here are different than the problem states of Section 2. Here, each state is a condition which may or may not be present, and the states are not mutually exclusive.

3. Categories of disease. Each category consists of a pattern of states and observations.

For an example involving glaucoma, see Figure 4.3. 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 1. The rules are combined together into a causal network which shows the courses that a disease can take.

Starting states, or states in the network without any predecessors, are taken to be conditions which can arise spontaneously in the patient. Each other state is assumed to occur only as a result of the occurrence of a state immediately preceding it in the network. For example, see Figure 4.4.

There are also rules for associating tests (or observations) with states. These rules are of the form

```
          Q[i,j]
t[i] --------> n[j]
```

where t[i] is a test (or observation) or Boolean combination of tests, n[j] is a state, and Q[i,j], a number in the interval [-1,1], is 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.

Evaluation of the rules is as follows: if

```
          Q[i,j]                              Q[k,j]
t[i] --------> n[j]    and    t[k] --------> 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 which is 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 (CF) with which it is believed that state n[j] has occurred in the patient. If the CF for n[j] is above an arbitrary threshhold 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.

CASNET allows for three main strategies for selecting tests for the user to perform on the patient: (1) local logical constraints among questions CASNET can ask the user, (2) likelihood measures over the pathophysiological states, and (3) calculation of the most likely cause of the disease. Each approach is explained briefly below.

(1) For the local logical constraint approach, questions on related topics are organized into small tree structures and are asked only if certain logical conditions are detected to hold.

(2) Likelihood measures over the states in the network are computed by assuming the occurrence (or nonoccurrence) of whatever states in the network have 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].

(3) 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 which have bearing on the selected starting states.

Sometimes the test results may contradict or conflict with the model, as in the case where 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, but these will not be described here.

Pathophysiological states are associated with disease categories as follows. Let n[1], n[2], ..., n[k] be the states in a causal pathway in the network, with starting state n[1]. Then one 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 actual implementation in CASNET is somewhat different, in that a single classification table may include information about several causal pathways. However, the usage 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 may 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], there may be a set T[j,1], T[j,2], ..., T[j,n] of several possible treatments. Which treatment is actually chosen is determined by rules of the form

```
        p[i,j,k]
t[i] ----------> T[j,k]
```

which assign preferences to treatments based on 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 diagnosis and treatment statements. CASNET's rules must be invoked once for each eye, and in addition, there are special rules for various types of binocular comparisons. According to Kulikowski [1981], knowledge from medical textbooks alone allowed CASNET to perform at 70% to 75% accuracy. To get the performance above

90%, it was necessary to incorporate information obtained directly from human experts.

4.3. Hearsay-II

Hearsay-II is a computer system, written in a computer language called SAIL, for speech understanding. Strictly speaking, it is not an expert system according to Feigenbaum's definition quoted earlier, because its performance is not as good as that of any reasonably good speaker of English. However, it is representative of the state of the art in automated speech recognition, and the techniques used in its construction are of interest to designers of expert systems.

The performance of Hearsay-II is as follows (as adapted from Erman et al. [1980], p. 239):

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 avg. 2.6 seconds/utterance avg.
Accuracy	9% sentences misunderstood; 10% sentences not word-for-word correct but meaning understood anyway.
Computing resources:	60 million instructions per second of speech on a PDP-10.
Time required to comprehend speech:	On the order of 10 times the length of the utterance.

The architecture of Hearsay-II is shown in Figure 4.5. The KS's, or knowledge sources, indicated in the figure are pattern-invoked computer programs (as described in Section 3). Each KS consists of a condition program (which evaluates whether the KS is applicable) and an action program (to accomplish whatever results the KS is to produce). The system contains approximately 40 KS's, which range in size from five to one hundred pages of source code apiece. Thirty pages is a typical KS size. Each KS has up to 50,000 bytes of its own local data storage.

The KS's 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. This is illustrated in Figures 4.6 and 4.7. As can be seen, Hearsay-II accomodates both KS's which operate top-down and KS's which operate bottom-up.

Examples of KS's are the following:

1. SEG divides the input signal up into segments, and assigns to each segment several alternate possiblilites for the phoneme it might be.
2. PARSE is a KS which 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

1. ways to define blackboard levels, configure KS groups, access

and modify blackboard hypotheses, activate and schedule KS's, and debug and analyse KS performance. It also provides ways to specify which KS's should have their condition programs invoked when new hypotheses appear on the blackboard, how to read hypotheses from the blackboard, and how to put new hypotheses onto the blackboard.

Probably the main features that distinguish the Hearsay-II architecture from that of MYCIN are the the use of arbitrary pattern-invoked programs as units of knowledge, rather than production rules, and the flexibility of the scheduler (as opposed to strict top-down invocation).

For a large, complex problem such as speech understanding, this offers several advantages. Since the KS's can be arbitrarily complex--and arbitrarily different in their internal operation--this provides a way to implement whatever problem-solving approach is most appropriate 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 entire rest of the system. This alleviates the problems often encountered with "combinatorial explosion" when production systems are used on very large problems. In fact, when portions of Hearsay-II were experimentally rewritten as a production system, the production system version was found to run approximately 100 times as slow [McCracken 1979].

4.4. DENDRAL and MDX

This section contains a few comments about DENDRAL and MDX. Rather than discussing these systems in detail, this section merely outlines some of the similarities and differences between them and the systems described earlier.

DENDRAL is a computer system which proposes plausible chemical structures for molecules, given their mass spectrograms. DENDRAL uses a "plan, generate and test" technique. First, constraints on the problem solution are inferred from mass spectrometry data. Second, the program generates all molecular structures satisfying these constraints, as well as general chemical constraints. Finally, the proposed structures are tested in a more sophisticated manner for compatibility with the mass spectrometry data.

According to Nilsson [1980, p. 41], the "generate" part of DENDRAL can be viewed as a production system. However, its operation is different from MYCIN.

MYCIN operates top-down. When MYCIN sets up a goal, it uses its rules to decompose the goal further and further, until primitive problems are reached which are answered in the data for the problem. The "generate" portion of DENDRAL, however, starts with the data and operates bottom-up. It has a number of rules corresponding various quantities of molecules with various pieces of molecular structure, and it applies these rules to the data to hypothesize structures for more and more pieces of the molecule. Examples of the operation of the "generate" part of DENDRAL and the types of molecular structures it proposes are shown in Figures 4.8 and 4.9.

For more information about DENDRAL, the reader is referred to Feigenbaum et al. [1971].

MDX is a medical diagnosis system, still under development, whose architecture is designed to try to handle the "combinatorial explosion" (mentioned at the end of Section 4.3) which may interfere with efforts to solve large problems.

MDX consists of a collection of small experts which can be thought of as being somewhat similar to Hearsay-II's KS's. These expert systems are organized in a hierarchy which corresponds to the taxonomy of a disease. For example, consider the classification in Figure 4.10 of various types of a medical condition called cholestasis. A complete implementation of MDX for cholestasis (not yet finished) would include an expert system corresponding to each node of the classification tree. The communication and transfer of control among these experts, instead of being handled by global mechanisms such as Hearsay-II's blackboard and scheduler, is handled by the experts themselves, and is constricted to flow along the lines of the hierarchy.

The hierarchy of experts corresponding to the tree of Figure 4.10 might be entered at the root of the tree, by a request for a diagnosis. The expert at each node, when invoked, does some processing involving its specialized knowledge, and may request information from other experts in order to make a diagnosis. Communication and transfer of control among experts is restricted to go along 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, and would perhaps be modified by them to put it into a form understandable to the cholangitis expert. If an expert is not implemented, a human being at a computer terminal can take the part of that expert, and the rest of the system will not know the difference.

For more information about MDX, see Chandrasekaran [1979].

5. PROCESS PLANNING REQUIREMENTS FOR THE AMRF

Process planning is the task of determining a plan of action for manufacturing a part. This activity is distinct from production planning, which involves taking the process plans for all parts to be produced and scheduling the factory's resources to perform the activities specified by the process plans.

This section contains a summary of the requirements for a process planning system for the Automated Manufacturing Research Facility (AMRF). Section 5.1 briefly describes the activities that the process planning system must perform, and makes some suggestions for how these activities might be done. Section 5.2 describes some of the three-dimensional modeling and reasoning capabilities required for process planning.

5.1. Process Planning Activities

Manufacturing a metal part consists of taking a piece of stock and performing various processes or operations on it to transform it into the desired part. The piece of material being transformed is called the workpiece. In the AMRF, the processes to be considered are those performed by machine tools (lathes, mills, etc.), which are equipped with cutting tools to remove material from the workpiece. A machined surface on the workpiece is any geometric surface or combination of geometric surfaces produced by a single machine tool operation. For example, a hole is a single machined surfaces, although it consists of a cylinder and a cone (see Figure 5.1). For more information on machining operations, see Boothroyd [1975], Chapter 1.

In rough outline, process planning in the AMRF will consist of the following activities.

1. Reception of a request for a process plan for a part.
2. Selection of a piece of stock.
3. Selection of appropriate machining processes.
4. Selection of an appropriate type of machine tool and cutting tool for each process.
5. Fixture selection for each process.
6. Robot gripper selection for each process.
7. Selection of a sequence (or several possible sequences) for the processes.
8. Selection of the work station to be used for each process.
9. Determination of machine tool parameters (feed rate, etc.).
10. Determination of tool trajectories for each process.
11. Creation of instructions to the machine tool, robot arm, robot cart, work station, cell, and other levels of the factory.
12. Verification of the process plan.

The above list is incomplete. Robot sensors, probes, reference surfaces (surfaces to be inspected), and inspection techniques must also be selected for use in monitoring the manufacture of the part. Instructions must be created for the sensors and probes. To handle the errors that the sensors may detect, error recovery instructions must be created for the machine tool, robot arm, robot cart, and other levels of the factory. To simplify the discussion, and because of lack of time, these activities are not discussed here.

There are many problems involved with automating the activities listed above. Complex reasoning about the properties of the part is required for practically every phase of process planning. The representation of the part used in the computer systems which do the process planning, and the capabilities for manipulating this representation and reasoning about it, must be quite sophisticated.

The author knows of no geometric modeling system which is currently capable of handling the problems involved. These problems are discussed further in Section 5.2. However, partial automation of process planning can be done without requiring the full capabilities of such a system. Some approaches for doing this are suggested below.

5.1.1. Receiving the Process Planning Request

In an automated manufacturing system, handling the reception of a request to do process planning will require constructing a computer representation of the part. This could conceivably be done either by having an Computer Aided Design (CAD) system on which a user designs the part, or by having an optical scanning system capable of reading engineering drawings. The former approach seems more reasonable. Engineering drawings are produced with the idea that a human will be reading them, and some design details may not explicitly be represented since they would be obvious to an experienced human. A CAD system, however, could be designed to request explicitly all information necessary to insure an unambiguous specification of the part.

Creating such a CAD system presupposes a solution to the problems of adequately representing and reasoning about the part automatically. Until these problems have been solved, it is more likely that information about the part will be entered into the AMRF through a variety of interactive computer systems designed to handle various phases of process planning. If desired, these systems could operate through a single user interface which organizes the requests of the various systems in a user-friendly manner.

Organizing the CAD system as an interface to handle interactions between the user and several computer systems (rather than as a self-contained system) might be the best approach even if total automation of the process planning becomes feasible. Different types of information may be required of the user during different phases of process planning, and it may be easier to let each system request the information as it is needed rather than design a self-contained CAD system capable of deciding when all necessary information has been entered.

5.1.2. Stock Selection

Selecting a piece of stock appropriate to be machined into a part requires examining the geometry of the part to be created, to decide what kind of stock is suitable, or whether an existing piece of scrap can be used rather than a new piece of stock. The basic problem is to select the smallest piece of stock which can be used to make the part, but there are some additional complications.

For example, a cylindrical piece of stock should be selected for rotational (lathe-turned) parts. Rectangular pieces of stock are harder to mount on a lathe, and they produce a shorter tool life when they are lathe-turned. As another example, it may be more appropriate for some parts to start with a casting rather than raw stock. Determining this requires examining and evaluating the machining processes necessary to produce the part (see Section 5.1.3).

One way that stock selection could be partially automated is as follows. A human sitting at a computer terminal could examine an engineering drawing or a computer graphics representation of a part to be manufactured, and give the computer information on the dimensions and other parameters necessary for a suitable piece of stock. The computer could then examine a database containing information on what pieces of stock are available, and generate a retrieval order for the smallest suitable piece.

5.1.3. Process Selection

Automatically selecting the appropriate processes to manufacture a part requires knowledge about the capabilities of each process. It also requires the ability to--

1. identify the machinable surfaces on the part and on each stage the workpiece passes though;
2. represent the changes made to the workpiece caused by each machining operation;
3. represent and reason about special features of the part, such as chamfers, threads, and tolerances; and

4. identify relevant geometrical relationships between each machinable surface and the rest of the part.

The geometrical relationships mentioned in the fourth item above can affect in what kinds of processes can be used, the order in which the processes must be performed (see Section 5.1.7), or whether the surface can even be machined. For example, it is impossible to machine a hollow cube with no external opening!

As discussed in Section 5.2, sophisticated techniques for representing and reasoning about the workpiece will be required if process selection is to be fully automated. However, it appears feasible to use AI techniques to create an interactive computer system to do process selection for certain kinds of parts. This is discussed further in Section 6.

Let P be a part, and A be a machinable surface of P. Process selection may produce several processes capable of producing A. Some of these processes, however, may not be appropriate for P. The set of eligible processes for A can be cut down by the activities described in Sections 5.1.4 through 5.1.8 below.

5.1.4. Selection of Machine Tool and Cutting Tool Types

Selecting appropriate types of machine tools and cutting tools to perform a given process should be relatively straightforward, given the requirements for that process. For example, one might have a database containing information on the capabilities of each type of machine tool available in the factory. The database would be searched to determine which machine tools, if any, could perform the process if suitable

cutting tools were available. For each eligible machine tool, a second database search could then be done to determine whether a satisfactory cutting tool is available.

It should be noted that two different machine tools performing the same process may not be able to provide the same tolerance. The current practice in industry is to pretend that if two machine tools can perform the same process, they can both do it equally well. For simplicity, it is probably best to do the same thing here, at least initially.

5.1.5. Fixture Selection

Fixture selection is a very complex task. It involves decisions such as where to place the fixtures on the workpiece, what kinds of fixtures are adequate to hold the workpiece firmly without damaging it, how to orient the workpiece, whether a fixture and a workpiece fit each other, and whether and how to design a special fixture to hold the workpiece.

Just how to automate fixture selection is unclear at this point. It is probably best, in the initial stages of the project, to do it by hand. If automation is to be considered later on, one way to simplify the problem would be to restrict the AMRF to use a standard set of fixtures. Fixture selection could be done by hand if a special fixture is needed.

5.1.6. Gripper Selection

Selecting a gripper with which the robot arm can hold the workpiece is a complex task involving problems similar to those of fixture selection. Again, it may be best to do this by hand at first. One way to simplify this problem for future automation

would be to use pallets to hold the workpieces. Then the same grippers could be used to hold all parts.

5.1.7. Process Sequencing

What process orderings are possible in making a part are constrained to some extent by the nature of the processes and the relationships among the machined surfaces. For example, drilling must be done before reaming. Furthermore, certain sequences of processes may produce workpieces which are difficult or impossible to fixture, or which do not have the shape or orientation necessary to be picked up without damaging them. These sequences should be eliminated.

Even after applying the above constraints, there may be more than one suitable sequence of processes. To choose which sequence to use, one might be interested in choosing the one having optimal time, cost, or profit.

One difficulty with approaching process sequencing as an optimization problem is that it may be difficult to obtain all of the necessary information. For example, one might have to figure out how long it might take to mount a workpiece on a complicated fixture.

Whether an optimal process sequence should be determined by the process planning system or left up to the production planning system will depend on how flexible the AMRF is expected to be. Additional flexibility--for example, the ability to choose various sequences dynamically, depending on the work load--will require a more complex system.

To avoid getting bogged down in complex problems at the beginning of the project, it might be best simply to choose a sequence of processes arbitrarily, or perhaps to choose them to minimize the number of time-consuming activities such as fixturing.

5.1.8. Selection of Work Stations.

This section deals with selecting the actual physical work station to be used for a process, rather than the type of machine tool (as discussed in Section 5.1.4). This should probably not be done until completion of the activities described in Sections 5.1.3 through 5.1.6, and some of the activities described in Section 5.1.7.

Activities such as machine tool selection and fixture selection are used to eliminate processes which cannot be performed. For example, let M be a machinable surface on a part, and P, Q, R, and S be processes which can be used to produce M. During machine tool and fixture selection (Sections 5.1.4 and 5.1.5), P may be eliminated because no machine tool in the factory can perform P, and Q may be eliminated because no there is no good way to fixture the workpiece to perform Q.

Work station selection is used to decide which is the most appropriate of several possible sets of processes for creating a part. For example, suppose process R can be done at work station B, and S can be done at work station C. If all other processes for manufacturing the part are to be done at work station B, then it is more efficient to select process R. Trying to make this kind of decision before determining that R and S are

possible to perform would merely complicate the process planning.

Given a set of processes to be performed, it is fairly straightforward conceptually to select work stations to perform them. However, it might require a fair amount of computation.

One possible approach would be to do a table lookup to determine which kinds of work stations are available for each process. Selecting a combination of these work stations for manufacturing the entire part could then be done. One way to do this would be to make selections which minimize the number of transfers of the part among different work stations. Making such a selection would be related to NP-complete problems such as the set covering problem or the traveling salesman problem [Aho et al. 1976].

5.1.9. Machining Parameter Determination

The machining parameters to be determined for each process consist of the feed rate, cutting speed, and sometimes the depth of cut. Determining these parameters is a nonlinear optimization problem involving various equations relating the parameters to factors such as tool life. This can be solved using operations research techniques to optimize for time, cost, or profit, as desired. For simplicity, some existing systems choose the machining parameters by a table lookup rather than trying to find an optimum.

For flexibility, it may be desirable for the machining parameters to be determined by interacting with the process planning system. For example, it may be appropriate to slow the machine tool down if the factory has a low work load. One way to

achieve this flexibility would be to let the process planning system decide what to optimize (cost, time, or profit).

5.1.10. Tool Trajectory Determination

Determining the tool trajectories can be done using mathematical techniques, but it first requires defining the boundaries of the volume of material to be removed. If these boundaries touch any other machined surfaces, this may restrict the possible angles of approach of the cutting tool. Determining such relationships and restrictions is a complex geometrical problem, which should initially be solved by hand.

5.1.11. Creation of Instructions

Creation of instructions to the NC machine tool should be fairly straightforward once the tool trajectories and machine tool parameters are known. Indeed, systems (such as the Autotrol in the NBS machine shop) are already being marketed for this kind of task.

Creation of instructions for the robot arm, robot cart, and to the various levels of the factory (work station, cell, etc.), appears more complicated at this point. Until the technology for interfacing and communicating automatically with these levels has been further developed, it will be hard to make recommendations for these tasks.

5.1.12. Verification of the Process Plan

In an error-free process planning system, verification would be unnecessary. However, errors are bound to occur, and it will be necessary to have ways for a human to examine the process

plan and verify that it is correct. For example, one might have a graphics display terminal on which the processes could be simulated, to see if they result in the desired part.

5.2. Geometric Modelling Requirements

If the process planning is to be done automatically, the part design information will have to be presented or translated into a computer representation suitable for manipulation by the process planning system. In a fully automated system, such a representation would have to be adequate to allow the system to--

1. determine whether a part can be physically produced from a given piece of stock. This requires not only deciding whether the part will physically fit within a region of space having the dimensions of the piece of stock, but also deciding whether it can be oriented correctly (according to some heuristic criterion of correctness) within this region of space. For example, see Figure 5.2.
2. determine how much material must be removed from a piece of stock to produce a part.
3. decide whether rectangular or cylindrical stock is more appropriate, whether a piece of scrap can be used, or whether to use a casting.
4. identify the machinable surfaces of the part.
5. identify intermediate surfaces to be machined on the workpiece (i.e., surfaces which may not appear in the final part).
6. determine reference surfaces (i.e., surfaces to be used for inspecting the part).

7. decide what processes can be used to produce a given machined surface.
8. match machining operations to volumes of material to be removed from the workpiece.
9. check adjacency of geometric surfaces (e.g., to decide whether a process will affect surfaces other than the one for which it was selected).
8. handle special features, such as threads, chamfers, and tolerance information.
9. decide whether two objects fit each other correctly (for example, whether a fixture or a gripper fits a workpiece).
10. check interference of geometric parts (e.g., to decide whether a tool trajectory for a process interferes with a fixture, or with portions of the workpiece that are not supposed to be affected by the process).

Several representations for three dimensional objects are currently the subject of experimentation. Examples of such representations are--

1. wire-frame representation [Nagel et al. 1980] [Requicha 1980], in which an object is represented by the set of edges bounding it;
2. boundary surface representation [Eastman et al. 1779] [Requicha 1980], in which an object is represented by the set of geometric surfaces bounding it;
3. constructive solid geometry (CSG) representation [Requicha 1980] [Voelcker et al. 1978], in which an object is represented as a Boolean combination of simple primitive objects such as blocks or cylinders.

For process planning, it is likely that several such representations will have to be used simultaneously. For example, boundary surface representation might turn out to be be best for identifying machinable surfaces. But for identifying the volume of material to be removed by a process, CSG might be more appropriate. As an added complication, the volume covered by a tool trajectory might be most appropriately represented as a swept volume [Requicha 1980] [Eastman et al. 1977], which is the volume generated by sweeping an object through space.

Several computer aided design systems are under development, using various sorts of computer representations of objects [Requicha 1980]. At present, none of these is adequate for totally automated process selection. For example, none of them allows the requisite handling of tolerances and special features.

The Initial Graphics Exchange Specification (IGES) [Nagel et al. 1980] is a standard for representing descriptions of parts. It was devised for use in transferring such descriptions from one computer system to another. The representation used in IGES is a wire-frame representation. Since this representation is in some cases ambiguous [Requicha et al. 1979], it is not adequate for transmitting information about objects which are represented in, say, CSG or boundary surface representation.

Since a boundary surface representation of an object can be thought of as an extention of a wire-frame representation of the object, it might be fairly straightforward to extend IGES to handle boundary surface representations. However, handling other representations may be more difficult. For example, although

there are procedures for converting a CSG representation to a boundary surface representation [Voelcker et al. 1980], no procedures have been developed for the inverse conversion.

6. RECOMMENDATIONS FOR PROCESS SELECTION

In this section, recommendations are made for the design of an interactive process selection system for the AMRF which uses expert system techniques from artificial intelligence. Section 6.1 is a brief description of the basic features of existing process selection systems. Section 6.2 contains the recommendations for the process selection system. Section 6.3 is a scenario for how such a system might operate.

6.1. Existing Systems

In several existing computer-aided process planning systems, process selection works as follows. A Group Technology (GT) code [Chang 1981] is used to classify a part as being in a family of similar parts. When a process plan for a part is desired, a human user enters the GT code for the part into the system, and the system retrieves a process plan which was previously used for some part in this family. The process plan is then modified by the user to produce a plan for the part. Examples of such systems are CAPP [Holtz 1978] and MIAPP [Hewgley et al. 1979].

Such systems are quite useful in industry, as they allow process plans to be made very quickly. However, if process planning is to be automated more fully, it will be necessary for the computer system to have more complete information about the part than just a GT code. In particular, detailed information must be available about each surface to be machined.

A few process planning systems have been developed experimentally which use as input a represention of each of the surfaces to be machined. Some of them are briefly described below.

CPPP [Mann et al. 1977] [Dunn et al. 1981] is a system which does process selection and sequencing for rotational (lathe-turned) parts. To use CPPP, a user gives CPPP the name of a "part family" containing the desired part, and for each surface to be machined, gives CPPP a code describing the surface.

The part family name is used to retrieve a process model, which is a simple computer program written in a language designed for that purpose. The process model must have been previously written and debugged by a human process planner, and must be general enough to handle every part in the (arbitrarily defined) family. Because of the restrictions of the process planning language, the process model amounts to a decision tree in which information about the surfaces is used to select and sequence processes.

APPAS [Wysk 1977] is a system which does process selection for both rotational and prismoidal parts. To use APPAS, the user enters for each surface a code describing the surface. The system then uses knowledge of the capabilities of various processes to select a process for each surface. Since APPAS is restricted to rotational and prismoidal components, it need not consider the geometrical relationships which may occur among the surfaces. Thus APPAS selects processes for each surface independently.

CADCAM [Chang 1980] [Chang et al. 1981] is an extention of APPAS. The main features of this extention are a graphics interface which allows the user to design a part interactively, and an encoding of the process planning decision logic into decision tables.

6.2. The Recommended System

6.2.1. Interactive Process Selection

It is obvious that the AMRF process selection system must make use of descriptions of the surfaces of a part, as do several of the systems described in Section 6.1. However, the AMRF system must be capable of doing process planning for a wider class of parts than just rotational or prismoidal parts. This will require information about the relationships among the surfaces, as well as descriptions of the surfaces themselves.

Ideally, the system would get the information it needs from examining a detailed representation of the part (as discussed in Sections 5.1 and 5.2). However, this cannot be done at present: it is not known what representations might be used, or how the system could examine these representations to get the necessary information. One way to avoid this problem is to have a human user provide whatever information might be necessary. Such information could include descriptions of the machinable surfaces, their orientation relative to other surfaces, what tolerances are required, which surfaces are adjacent, what special features are present, and so forth.

Not all of the above information will be necessary for every surface of a part, and it may not be clear at the outset

what information will be required for each surface. If the user is required to enter a complete description of the part before the process selection system starts operating, it may turn out that some information is unnecessary and other information is missing. Therefore, it would be better to have an interactive system which would ask the user for information as it is needed.

Another advantage of this approach is that the system could gradually be interfaced with other parts of the process planning system as they are developed. For example, suppose it is decided that AMRF should make use of a GT code describing the parts to be manufactured. Then the process planning system could be instructed that before asking the user each of its questions, it should first check to see if the answer could be deduced from the GT code.

To continue the example, suppose a usable computer representation for machined parts is developed later in the AMRF project. If it becomes obvious how to use this representation to decide what the machinable surfaces are, a module could be written to examine the part representation and give this information to the process selection system. Modules to answer other questions could gradually be added to automate the process selection as much as possible.

6.2.2. AI Techniques for Knowledge Representation

At present, it is not obvious what all of the factors are that might influence the process selection and sequencing for a particular part, or what effects these factors might have. It may be necessary to make extensive modifications and additions to

the decision logic as the system is developed. The system may need decision-making knowledge which can only be gained by talking to a human expert, as was the case with medical diagnosis systems such as MYCIN [Davis et al. 1977], CASNET [Weiss et al. 1978], or MDX [Chandrasekaran et al. 1979]. For these reasons, it appears best to represent the problem-solving knowledge using techniques similar to those used by designers of expert systems in AI. These techniques could be used to represent knowledge about what information to request, the order in which to request it, and how to use it for process selection.

If implemented correctly, such techniques would allow pieces of knowledge easily to be added, changed, or removed from the system without having to modify other pieces of the system to accomodate each change. Such an extensible system would have distinct advantages. Initially, knowledge could be encoded into the system to allow it to do process selection on some simple class of parts (e.g., lathe-turned parts). This would give the workers on the AMRF project some experience with the problems involved with automated process planning. Once working, the system could have its knowledge augmented to allow it to handle other parts.

One way to represent the decision-making knowledge would be to use production rules, as was done in MYCIN [Davis et al. 1977]. However, different types of decision-making knowledge might be better represented in other ways (for example, semantic nets, frames, or actual computer code). Thus it might be worthwhile to allow for interactions between sources of knowledge encoded in arbitrarily different ways, as was done in systems

such as MDX [Chandrasekaran et al. 1978] and HEARSAY-II [Erman et al. 1980].

6.2.3. Summary

In summary, the recommendations for a process selection system for AMRF are as follows.

1. The system should be interactive. It should ask the user questions about the part to be manufactured, and use the answers to these questions to do process selection. This would allow the system to be interfaced to the other parts of the process planning system as they are developed.
2. The knowledge base for the system should be organized using AI techniques. This would allow for the decision-making knowledge to be modified or extended easily.

How these recommendations might be used to design a process selection system is illustrated in Section 6.3.

6.3. Scenario for an Automated Process Selection System

This section is a scenario describing the structure and operation of a hypothetical process selection system. The system is interactive, and uses knowledge representation and problem solving techniques from AI, as suggested in Section 6.2. Section 6.3.1 describes the representation the system uses for machinable surfaces and machine tool processes. Section 6.3.2 contains an annotated dialog between the system and a user.

6.3.1. Representation of Machinable Surfaces

For each type of machinable surface in our hypothetical system (as well as for certain other geometric entities), there is an uninstantiated (i.e., empty) frame representing the possible characteristics of that type of surface. Whenever a user specifies a new surface to be machined, frame corresponding to that type of surface is copied, and the copy is used to represent the surface. During the interaction between the user and the system, the slots of this frame are filled in with values describing the surface.

Below are frames for two types of machinable surfaces: holes and chamfers.

Frame type: hole
name: _____ (a unique identifier for the hole, to be filled in when the frame is copied)

slot name	possible values	comments
bottom	conical or flat	default is conical
length	a real number	length of the hole
pos-length-tol	a real number	pos. tolerance for length
neg-length-tol	a real number	neg. tolerance for length
diameter	a real number	diameter of the hole
pos-diam-tol	a real number	pos. tolerance for diam.
neg-diam-tol	a real number	neg. tolerance for diam.
on-surface	a flat surface name	the surface on which the hole is located
x-loc	a real number	x coordinate of location
y-loc	a real number	y coordinate of location
z-loc	a real number	z coordinate of location
loc-tol	a real number	max acceptable distance from location
x-angle	a real number	angle of tilt to x axis
z-angle	a real number	angle of tilt to z axis
surface-finish	a real number	max acceptable roughness
straightness	a real number	max acceptable deviation
roundness	a real number	max acceptable deviation
parallelism	a real number	max acceptable deviation

frame type: chamfer
name: _____ (a unique identifier for the chamfer, to be filled in when the frame is copied)

slot name	possible values	comments
hole	a name of a hole	the hole on which the chamfer appears
type	simple linear, inner fillet, outer fillet	different types of chamfers
radius	real	

For each type of surface, there is a list of processes capable of creating the surface. For each process in this list, there is a frame specifying the restrictions on the capabilities of the process, and the preconditions that have to be satisfied before the process can be performed.

Below are frames for three processes capable of creating a hole: twist drilling, spade drilling, and rough boring. The system contains similar frames for other hole-creating processes, such as finish boring, rough reaming, and finish reaming.

In the frames below, the hole to be created is referred to by the variable name H. Each of the processes requires the existence of other machined surfaces or geometrical entities. These are referred to by other variable names, such as C, S, and H'. The data used in the process restrictions were taken from Chang [1980], pp. 164-5.

```
frame for a process to create a hole H
process name:  twist-drill
-------------------------------------------------------------------
list of restrictions:
    1.  diameter(H) > .063
      COMMENT:  diameter(H) refers to the value in the diameter
      slot of the frame for the hole H.  The meanings of the
      other functions below are analogous.
    2.  diameter(H) < 2
    3.  length(H) < 12 * diameter(H)
    4.  pos-diam-tol(H) > (.007*diameter(H)) ** 0.5 + .003
    5.  neg-diam-tol(H) > (.007*diameter(H)) ** 0.5
    6.  loc-tol(H) > .008
    7.  straightness(H) >
                (.0005*length(H)/diameter(H)) ** 3 + .002
    8.  parallelism(H) >
                (.001*length(H)/diameter(H)) ** 3 + .003
    9.  roundness(H) > .004
   10.  surface-finish(H) > 100
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
list of preconditions:
    1.  either one of the following:
        1a.     list of preconditions:
                        type(on-surfaceH)) = flat-surface
                        x-angle(on-surfaceH)) = x-angle(H)
                        z-angle(on-surfaceH)) = z-angle(H)
        1b.     spot-face F
                        hole(F) = H
    2.  hole-workspace C
                x-location(C) = x location(H)
                y-location(C) = y location(H)
                z-location(C) = z location(H)
                x-angle(C) = x-angle(H)
                z-angle(C) = z-angle(H)
-------------------------------------------------------------------
```

```
frame for a process to create a hole H
process name:  spade-drill
-------------------------------------------------------------
list of restrictions:
    1.  diameter(H) > 0.75
    2.  diameter(H) < 4
    3.  length(H) < 4 * diameter(H)
    4.  pos-diam-tol(H) > (.005*diameter(H)) ** 0.5 + .003
    5.  neg-diam-tol(H) > (.004*diameter(H)) ** 0.5 + .025
    6.  loc-tol(H) > .008
    7.  straightness(H) >
                (.0004*length(H)/diameter(H)) ** 3 + .002
    8.  parallelism(H) >
                (.0008*length(H)/diameter(H)) ** 3 + .003
    9.  roundness(H) > .004
   10.  surface-finish(H) > 100
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
list of preconditions:
    1.  one of
                flat-surface S
                        x-angle(S) = x-angle(H)
                        z-angle(S) = z-angle(H)
                spot-face F
                        hole(F) = H
    2.  hole-workspace C
                x-location(C) = x-location(H)
                y-location(C) = y-location(H)
                z-location(C) = z-location(H)
                x-angle(C) = x-angle(H)
                z-angle(C) = z-angle(H)
-------------------------------------------------------------
```

```
frame for a process to create a hole H
process name:  rough-bore
-----------------------------------------------------------------
list of restrictions:
    1.  diameter(H) > 0.375
    2.  diameter(H) < 10
    3.  length(H) < 10 * diameter(H)
    4.  pos-diam-tol(H) > .002
    5.  neg-diam-tol(H) > .002
    6.  loc-tol(H) > .0001
    7.  straightness(H) > .0003
    8.  parallelism(H) > .0005
    9.  roundness(H) > .0003
   10.  surface-finish(H) > 8
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
list of restrictions:
    1.  hole H'
                  COMMENT:  H' must be at the same location as H,
                  but the tolerance requirements for H' are
                  different from those for H.
                length(H') = length(H)
                x-loc(H') = x-loc(H)
                y-loc(H') = y-loc(H)
                z-loc(H') = z-loc(H)
                x-angle(H') = x-angle(H)
                z-angle(H') = z-angle(H)
                  (at this point, the frame includes
                  various restrictions on the diameter,
                  diametric tolerances, locational tolerance,
                  surface finish, straightness,
                  roundness, and parallelism of H')
    2.  hole-workspace C
                  (space in which to do boring)
                x-location(C) = x-location(H)
                y-location(C) = y-location(H)
                z-location(C) = z-location(H)
                x-angle(C) = x-angle(H)
                z-angle(C) = z-angle(H)
-----------------------------------------------------------------
```

In the system, the frames for the processes are used in a manner similar to the production rules for STRIPS (Section 2.3.2) or MYCIN (Section 4.1).

6.3.2. A Sample Dialog

The system has a graphics display terminal whose screen is divided into three parts: a display of the workpiece, a display of the surface being currently described by the user, and

a space in which the dialog between the user and the system is typed.

At the top level, the control structure for the system is a simple loop:

```
LOOP
        s := ask-user-for-surface()
        IF s is a command rather than a surface
        THEN execute s
        ELSE find-way-to-produce(s)
REPEAT
```

If we list the process frames for a surface, the process frames relevant to the preconditions for these frames, and so forth, the resultant structure is an AND/OR graph similar to that of MYCIN (see Figure 4.2). The subroutine "find-way-to-produce" is similar in structure to the "find-value-of" subroutine of MYCIN (see Figure 4.1). It starts at the top of this graph and calls itself recursively to do a top-down, depth-first search through the graph. Searching this graph will yield a sequence of processes capable of generating the surface, if such a sequence exists.

Suppose a user is designing a workpiece which has a flat horizontal surface with a hole in it. Suppose the flat surface has already been handled by the system, and the user decides to enter the information about the hole. Then the dialog between the system and the user might proceed as outlined in the following paragraphs.

SYSTEM: "Enter next surface."

USER: "hole"

The system retrieves and copies the frame for holes. The new frame, which represents the hole to be produced, is given the name HOLE-1. Using default values for various slots in the frame, the system draws a picture of HOLE-1 in the "current surface" portion of the screen.

SYSTEM: "The hole is called HOLE-1."

"find-way-to-produce(HOLE-1)" is then called. This causes the system to retrieve the frames for all processes capable of creating a hole, and trying to find one whose restrictions are satisfied by HOLE-1. The first process frame retrieved is the "twist-drill" frame. The system checks this frame's restrictions, one by one, to see if they are satisfied.

The first restriction is

diameter(HOLE-1) > .063.

Whenever the system has no other way of finding out a piece of information, it asks the user.

SYSTEM: "Enter the diameter of HOLE-1."

USER: "1.0"

The value "1" is entered in the "diameter" slot for HOLE-1. Since this changes one of the default values used in the pictorial display of the hole, the picture is redrawn.

Restriction 1 is satisfied, so the system checks restriction 2:

diameter(HOLE-1) < 2.

This restriction is also satisfied, so the system checks restriction 3:

length(HOLE-1) < 12 * diameter(HOLE-1).

For this, the system needs to know the hole's length.

SYSTEM: "Enter the length of HOLE-1."

USER: "1"

Again, the new information is put into the frame for HOLE-1, and the displayed picture of HOLE-1 is redrawn.

Restriction 3 being satisfied, the system checks restriction 4:

pos-diam-tol(HOLE-1) > (.007*diameter(HOLE-1)) ** 0.5 + .003.

SYSTEM: "Enter the positive tolerance of HOLE-1."

USER: "pos-diam-tol = neg-diam-tol = .005" (The user can enter several items at once if he desires, or may even enter arbitrary commands to the system.)

Restriction 4 is not satisfied, so HOLE-1 cannot be created by twist drilling. Thus the system retrieves the frame for another hole-making process: "spade-drill". Using the information already in the frame for HOLE-1, the system ascertains that although the first three restrictions for spade drilling can be satisfied, the fourth one cannot. Thus spade drilling will not work, either.

Continuing its effort to find a usable process, the system retrieves other hole-making process frames, one by one. For each one, it asks whatever questions are necessary to establish whether their restrictions can be met.

The first frame for which all restrictions can be met is the "rough-bore" frame. This means that rough boring can be used to produce HOLE-1, provided that the frame's preconditions are met. As shown in the "rough-bore" frame earlier, there are two preconditions:

1. A hole (H' in the preconditions) must already exist.

However, the tolerances for H' are not as close as those for HOLE-1.

2. There must be sufficient open space (hole-workspace C in the preconditions) around and above the hole for boring to be done.

Each of these preconditions is set up as a subgoal.

The system works on the first precondition first. Since no hole has yet been created in the workpiece, H' does not match any existing hole. Therefore, the system creates a new hole frame, HOLE-2, and assigns H' = HOLE-2.

"find-way-to-produce(HOLE-2)" is now called. The system starts retrieving hole-making process frames and asking questions as before, to determine whether there is a process capable of producing HOLE-2. The slot values for HOLE-2 are not determined by asking questions of the user, but instead are filled in as needed using the information specified in the preconditions for the "rough-bore" process. Since this information refers to slot values in HOLE-1, and since some of these values may not yet be known, the user may be asked questions about HOLE-1.

This time, the system determines that HOLE-2 satisfies the restrictions of the "twist-drill" process, so it starts checking the preconditions for twist drilling.

As shown in the "twist-drill" frame earlier, the first precondition is either (a) that the surface on which HOLE-2 is located be oriented correctly for HOLE-2 to be drilled, or (b) that there be a spot face F for HOLE-2. The surface on which HOLE-2 is located, and its location and orientation relative to this surface are unknown. However, these data can be determined

from the corresponding data for HOLE-1. The system now gets this information from the user. By checking the frame it already has for the surface on which HOLE-2 is located, the system ascertains that the surface is oriented correctly for HOLE-2 to be drilled.

Since the orientation and location of HOLE-1 are now known, the system now moves HOLE-1 from its position in the "current surface" portion of the screen to its correct location on the workpiece.

"twist-drill" also requires sufficient space (hole-workspace C in the preconditions) for drilling to be done. The list of applicable processes for creating a hole-workspace contains a process called "ask-about-hole-workspace". The list of restrictions for this process is empty, so the system checks the list of preconditions. In place of a precondition list, the process contains a call to a subroutine which displays an outline of the required space on the screen, and asks the user if this space intersects the workpiece. The user replies that everything is OK, so this subgoal is satisfied.

The only remaining subgoal is the hole-workspace for the "rough-bore" process. Since this space occupies the same physical volume as the hole-workspace for the "twist-drill" process, the system ascertains that the subgoal is satisfied without having to ask the user any more questions.

At this point, the problem of producing HOLE-1 is solved. The system continues interacting with the user until it has determined how to produce every machined surface on the part to be manufactured, and then it prints out a list of the processes to be used.

ACKNOWLEDGEMENTS

In the preparation of the report, the willingness of the following people to discuss problems, provide source material, and read rough drafts was quite helpful: Howard Bloom, Ted Chang, Bill Gevarter, and Ted Hopp.

Several of the figures in this paper are reprinted or adapted from other sources. Credits are as follows.

1. Figures 2.9 and 2.12: Winston, Artificial Intelligence, c. 1977, Addison-Wesley, Reading, Mass. Figures 3-15 and 3-19 used with permission.
2. Figure 4.1: Artificial Intelligence, Vol. 8, c. 1977, North-Holland Publishing Co. Material used with permission.
3. Figures 2.14, 2.15, 2.16, 4.8, and 4.9, and the quote on pp. 17-18: Nilsson, Principles of Artificial Inelligence c. 1980, Tioga Publishing Co. Material used with permission.
4. Figures 4.3 and 4.4: Artificial Intelligence, Vol. 11, c. 1978, North-Holland Publishing Co. Material used with permission.
5. Figure 4.5, 4.6, and 4.7, and the Table on p. 41: and Table II: ACM Computing Surveys, Vol. 12, c. 1980, Association for Computing Machinery. Material used with permission.
6. Figure 4.10: Proc. Sixth Internat. Joint Conf. Artif. Intelligence, c. 1979, International Joint Conference on Artificial Intelligence. Material used with permission.

REFERENCES

Aho, A. V.; Hopcroft, J. E.; and Ullman, J. D. The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading, MA, 1976.

Bobrow, D. G. and Winograd, T. "An Overview of KRL, a Knowledge Representation Language." Cognitive Science 1:1 (1977), 3-46.

Boothroyd, G. Fundamentals of Metal Machining and Machine Tools. Scripta, Washington, DC, 1975.

Brachman, R. "On the Epistemological Status of Semantic Networks." In Associative Networks: Representation and Use of Knowledge by Computer, N. V. Findler, Ed. Academic Press, 1979, pp. 3-50.

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 (Aug. 1979), 134-142.

Chang, T.-C. Interfacing CAD and CAM - a Study of Hole Design. M.S. Thesis, Virginia Polytechnic Institute, 1980.

Chang, T.-C. and Wysk, R. A. "An Integrated CAD/Automated Process Planning System." AIIE Transactions 13:3 (Sept. 1981)

Chang, T.-C. "Group Technology and its Applications: a Tutorial." Tech. Report., Industrial Systems Division, National Bureau of Standards, in preparation (1981).

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).

Clowes, M. "On Seeing Things." Artificial Intelligence 2:1 (1971), 79-116.

Davis, R.; Buchanan, B.; and Shortliffe, E. "Production Rules as a Representation for a Knowledge-Based Consultation Program." Artificial Intelligence 8:1 (1977), 15-45.

Dunn, M. S. Jr.; Bertelsen, J. D.; Rothauser, C. H.; Strickland, W. S.; and Milsop, A. C. "Implementation of Computerized Production Process Planning." Report R81-945220-14, United Technologies Research Center, East Hartford, CT (June 1981).

Eastman, C. and Henrion, M. "GLIDE: a Language for Design Information Systems." ACM Computer Graphics 11:2 (July 1977), 24-33.

Erman, L. D.; Hayes-Roth, F.; Lesser, V. R.; and Reddy, D. R. "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty." Computing Surveys 12:2 (June 1980), 213-253.

Fahlman, S. E. NETL: A System for Representing and Using Real-World Knowledge. MIT Press, Cambridge, MA, 1979.

Fikes, R. E. and Nilsson, N. J. "STRIPS: a New Approach to the Application of Theorem Proving to Problem Solving." Artificial Intelligence 2:3/4 (1971), 189-208.

Feigenbaum, E.; Buchanan, G.; and Lederberg, J. "Generality and Problem Solving: a Case Study Using the DENDRAL Program." In Machine Intelligence 6, Meltzer, B. and Michie, D., Eds. Edinburgh University Press, Edinburgh, 1971, pp. 165-190.

Feigenbaum, E. A. "Expert Systems in the 1980s." Unpublished. Computer Science Dept., Stanford Univ., Stanford, CA (1980).

Goldstein, I. and Papert, S. "Artificial Intelligence, Language and the Study of Knowledge." Cognitive Science 1:1 (1977), 84-123.

Hart, P. E.; Duda, R. O.; and Einaudi, M. T. "A Computer-Based Consultation System for Mineral Exploration." Unpublished report, SRI International, Menlo Park, CA (1978).

Hewgley, R. E. Jr. and Prewett, H. P. Jr. "Computer Aided Process Planning at the Oak Ridge Y-12 Plant: a Pilot Project." Report Y/DA-8297, Union Carbide, Oak Ridge Y-12 Plant, Oak Ridge, TN (April 1979).

Hewitt, C. "Description and Theoretical Analysis (Using Schemata) of PLANNER: a Language for Proving Theorems and Manipulating Models in a Robot." Tech. Rep. 258, MIT AI Laboratory (1972).

Holtz, R. D. "GT and CAPP Cut Work-in-Progress Time 80%." Assembly Engineering 21:7 (July 1978), 16-19.

Huffman, D. "Impossible Objects as Nonsense Sentences." In Machine Intelligence 6, Meltzer, B. and Michie, D., Eds. Edinburgh University Press, Edinburgh, 1971, pp. 295-323.

Kulikowski, C. Personal communication (1981).

Mann, W. S.; Dunn, M. S.; and Pflederer, S. J. "Computerized Production Process Planning." Report R77-942625-14, United Technologies Research Center (Nov. 1977).

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. (1977), 882-887.

McCracken, D. L. "Representation and Efficiency in a Production System for Speech Understanding." Proc. Sixth Internat. Joint Conf. Artif. Intell. (1979), 556-561.

McDermott, D. "The PROLOG Phenomenon." SIGART Newsletter

Michie, D. "Knowledge-Based Systems." Tech. Rep. UIUCDCS-R-80-1001, Computer Sci. Dept., Univ. of Ill., Urbana, IL 61801 (Jan. 1980). No. 72 (July 1980), 16-20.

Minsky, M. "A Framework for Representing Knowledge." In The Psychology of Computer Vision, P. H. Winston, Ed. McGraw-Hill, New York, 1975, pp. 211-277.

Myolopoulos, J. "An Overview of Knowledge Representation." Proc. Workshop on Data Abstraction, Databases, and Conceptual Modeling (June 1980), 5-12.

Nagel, R. N.; Braithwaite, W. W.; and Kennicot, P. R. "Initial Graphics Exchange Specificiation IGES Version 1.0." Report NBSIR 80-1978 (R), National Bureau of Standards, Gaithersburg, MD (March 1980).

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. (1979), 645-655.

Nilsson, N. J. Problem-Solving Methods in Artificial Intelligence. McGraw-Hill, New York, 1971.

Nilsson, N. J. Principles of Artificial Intelligence. Tioga, Palo Alto, CA, 1980.

Osborn, J.; Fagan, L.; Fallat, R.; McClung, D.; and Mitchell, R. "Managing the Data from Respiratory Measurements." Medical instrumentation 13:6 (Nov. 1979).

Phillips, R. H. A Computerized Process Planning System Based on Component Classification and Coding. Ph.D. Thesis, Purdue University, 1978.

Pople, H. E. "The Formation of Composite Hypotheses in Diagnostic Problem Solving: an Exercise in Synthetic Reasoning" Sect. 1-4 Proc. Fifth Internat. Joint Conf. Artif. Intell. (1977), 1030-1037.

Reggia, J. A. and Perricone, B. T. "Knowledge-Based Decision Support Systems: Development Through High-Level Languages." Twentieth Ann. Tech. Conf., Wash. D.C. Chapter, ACM, College Park, MD (June 1981). (1981), 75-82.

Requicha, A. A. G. and Voelcker, H. B. "Geometric Modelling of Machanical Parts and Machining Processes." Proc. Compcontrol '79, Sopron, Hungary (Nov. 1979).

Requicha, A. A. G. "Representations for Rigid Solids: Theory, Methods, and Systems." Computing Surveys 12:4 (Dec. 1980), 437-464.

Schank, R. C. Conceptual Information Processing. North-Holland, New York, 1975.

Schubert, L. K. "Extending the Expressive Power of Semantic Nets." Artificial Intelligence 7:2 (1976), 163-198.

Sussman, G. J. and McDermott, D. "From PLANNER to Conniver, a Genetic Approach." Proc. Fall Joint Computer Conf. 41 (1972), 1171.

van Emden, M. H. and Kowalski, R. A. "The Semantics of Predicate Logic as a Programming Language." J. ACM 23:4 (1976).

van Melle, W. "A Domain-Independent Production Rule System for Consultation Programs." Proc. Sixth Internat. Joint Conf. Artif. Intell. (1979).

Voelcker, H. B.; Requicha, A. A. G.; Hartquist, E. E.; Fisher, W. B.; Metzger, J.; Tilove, R. B.; Birrell, n. K.; Hunt, W. A.; Armstrong, G. T.; Check, T. F.; Moote, R., and McSweeney, J. "The PADL-1.0/2 System for Defining and Displaying Solid Objects." ACM Computer Graphics 12:3 (Aug. 1978), 257-263.

Voelcker, H. B. and Requicha, A. A. G. "Boundary Evaluation Procedures for Objects Defined via Constructive Solid Geometry." Tech. Memo. No. 26, Production Automation Project, Univ. of Rochester (1980).

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 (1978), 145-172.

Weiss, S. M. and Kulikowski, C. A. "EXPERT: a System for Developing Consultation Models." Proc. Sixth internat. Joint Conf. Artif. Intell. (1979), 942-947.

Winston, P. H. Artificial Intelligence. Addison-Wesley, Reading, MA, 1977.

Wysk, R. A. An Automated Process Planning and Selection Program: APPAS. Ph.D. Thesis, Purdue University, 1977.

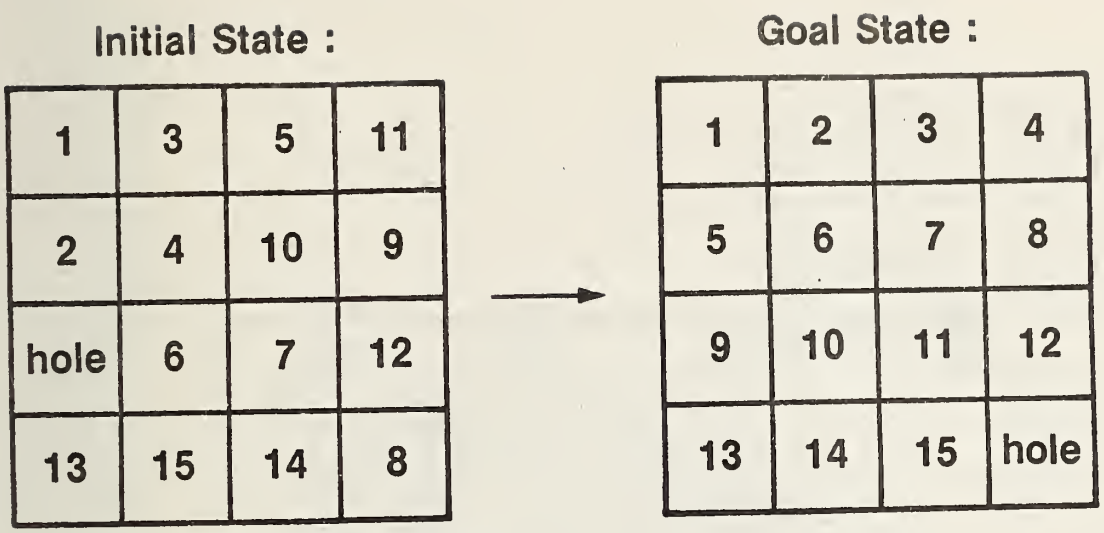


Figure 2.1. The 15-puzzle.

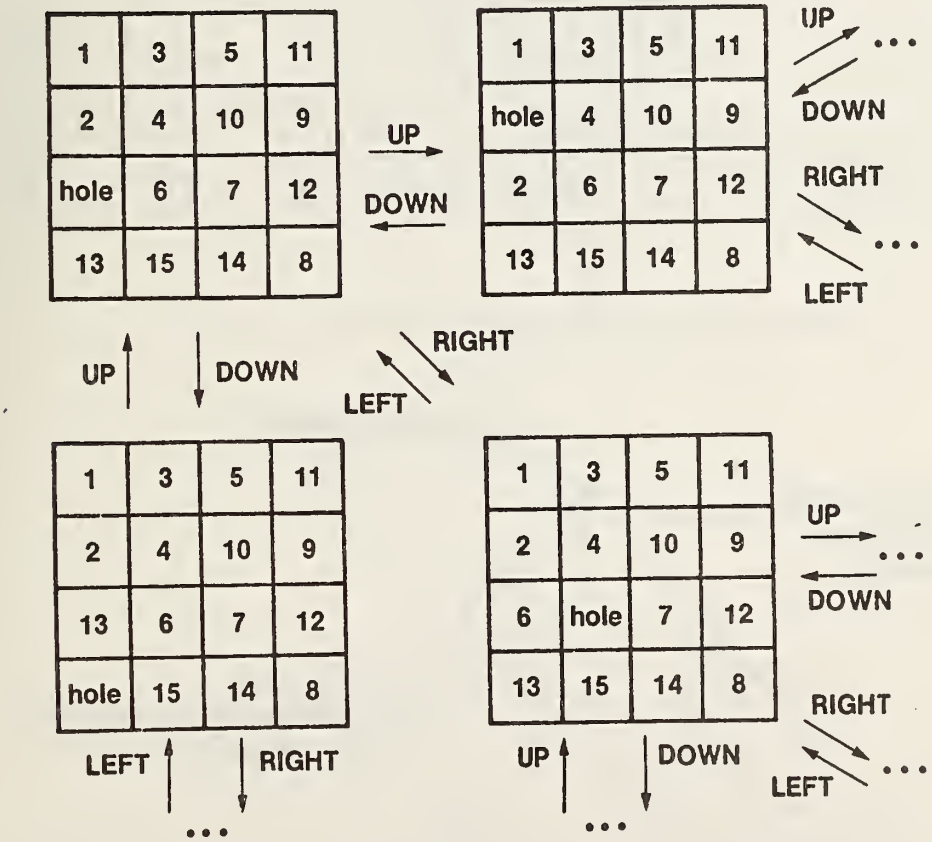


Figure 2.2. A portion of the state space for the 15-puzzle.

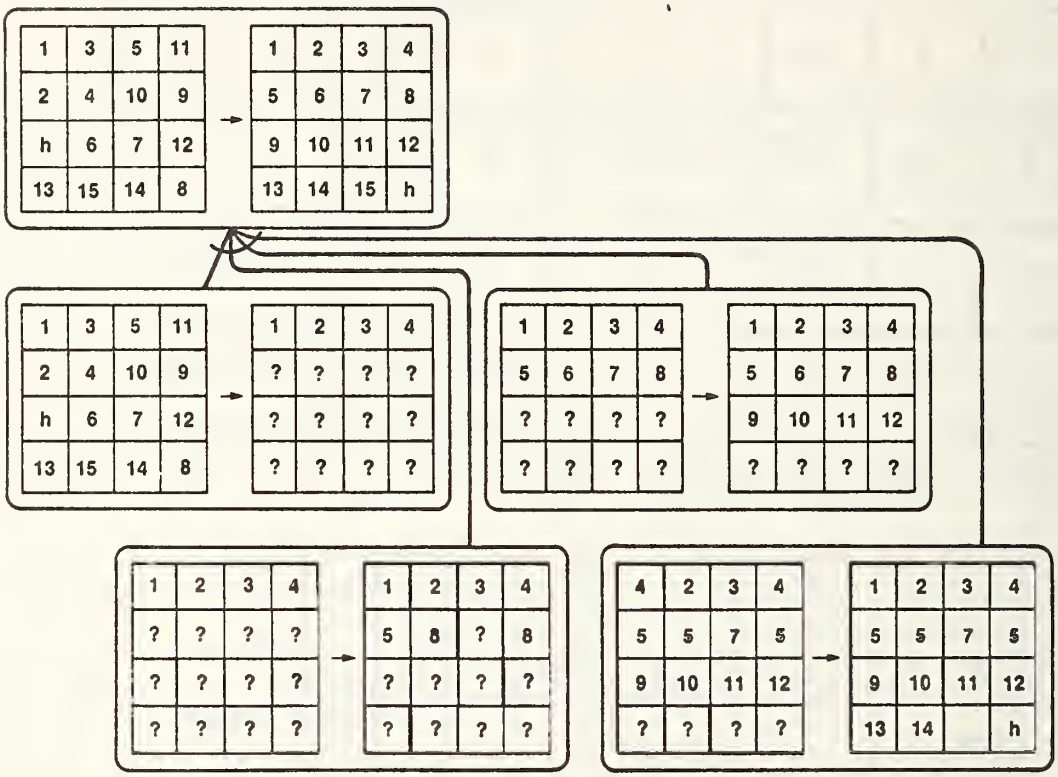


Figure 2.3. Problem reduction on the 15-puzzle.

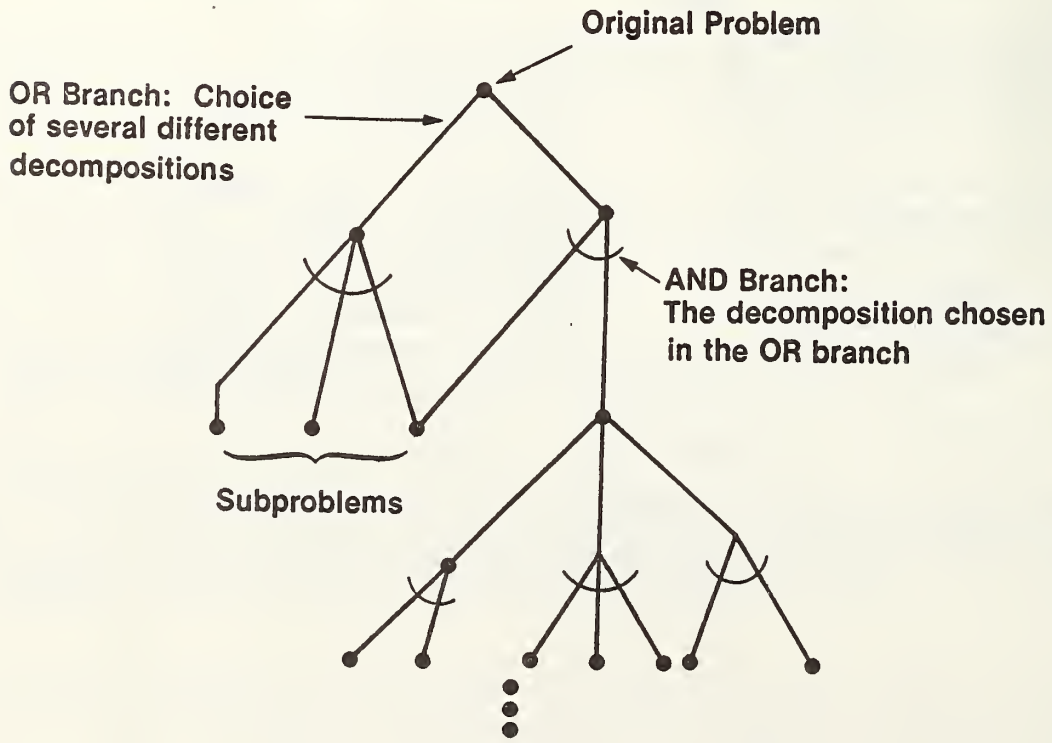
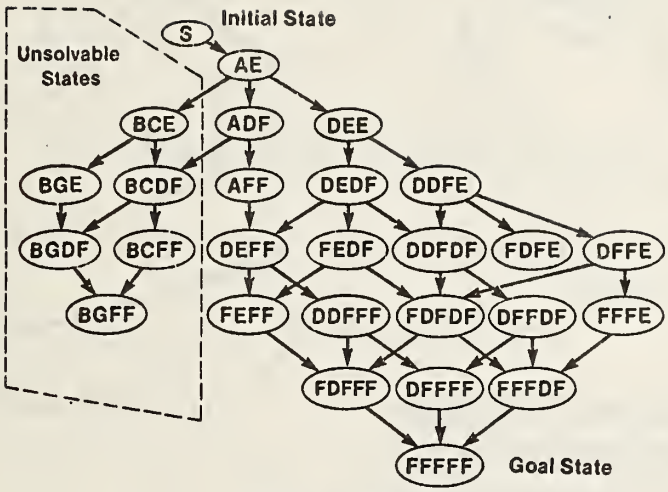
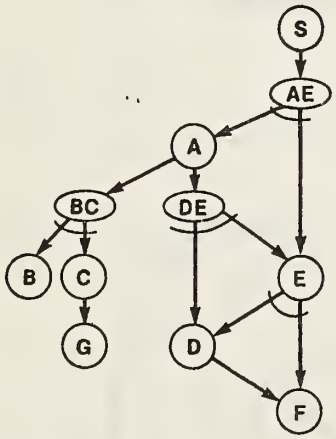


Figure 2.4. An AND/OR graph.



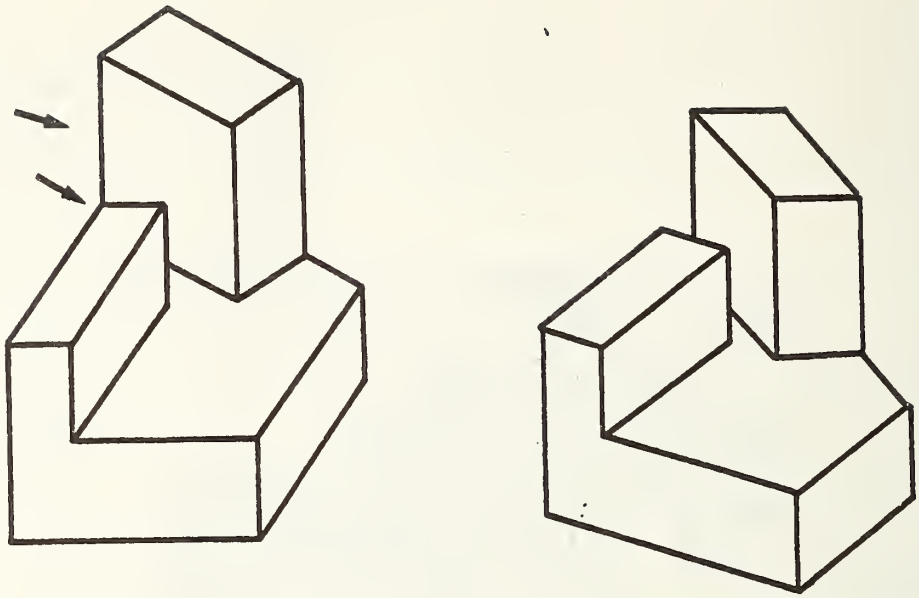
A. State Space Graph



B. Problem Reduction Graph

Figure 2.5. State space and problem reduction 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, and the operators are:

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.



(a) Not allowed, because the indicated line is falsely perceived as going into the indicated vertex

(b) Allowed

Figure 2.6. Not allowable and allowable viewpoints for a line drawing.

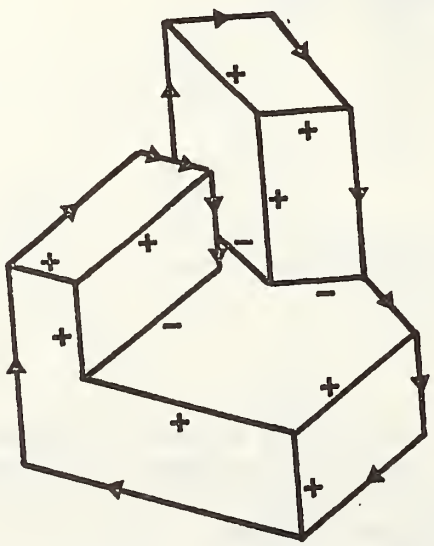


Figure 2.7. Huffman-Clowes labels on the line drawing of Figure 2.6. Convex intersections of surfaces are indicated by "t" labels; concave intersections of surfaces are indicated by "-" labels; and borders of the object are indicated by arrows directed such that the object is to the right of the arrow and the background is to the left.

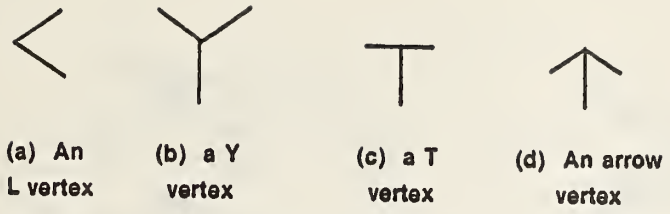


Figure 2.8. Physically possible types of vertices.

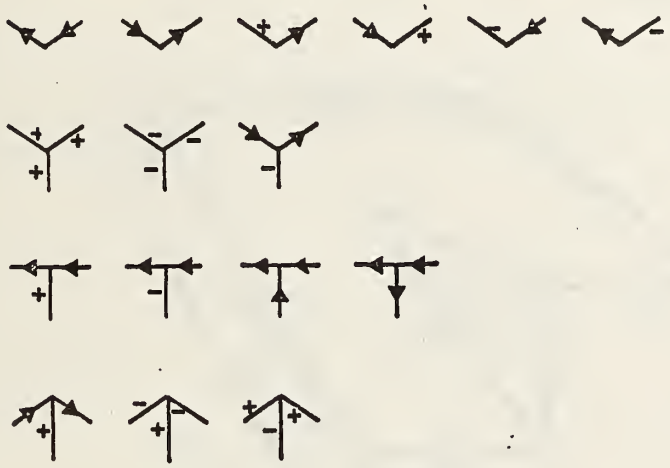


Figure 2.9. Physically possible vertex labelings.

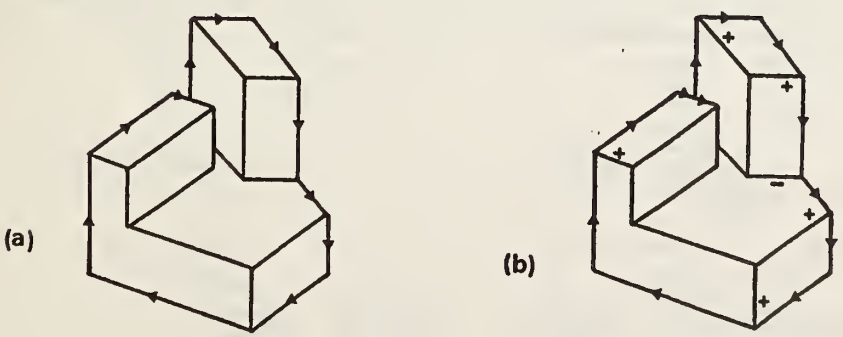


Figure 2.10. Successive steps in labeling the line drawing of Figure 2.6.

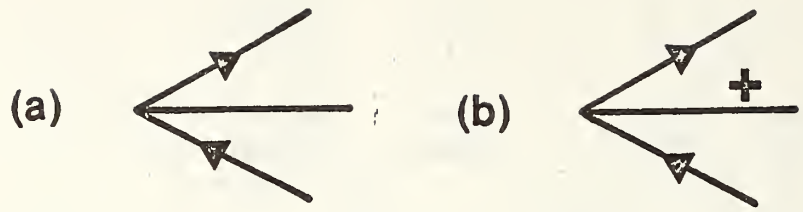


Figure 2.11. Labeling a line based on the labels of other lines going into the same vertex.

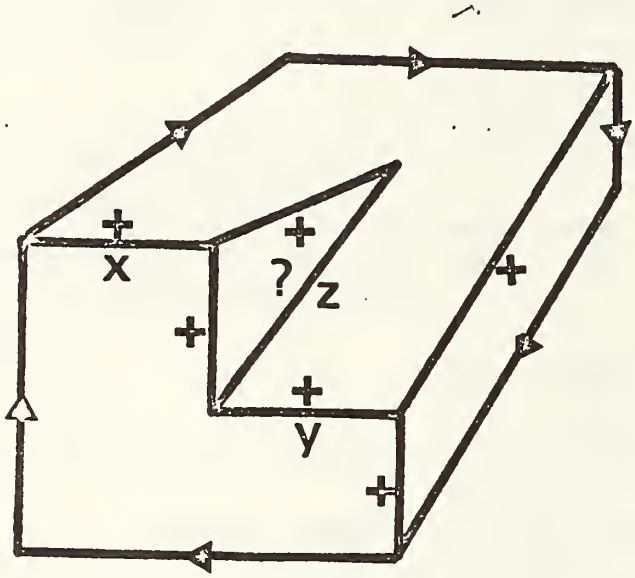


Figure 2.12. An impossible object. There is no way to label line z such that the vertices on both of its ends have legal combinations of labels. This example is taken from Winston [1977], p. 59.

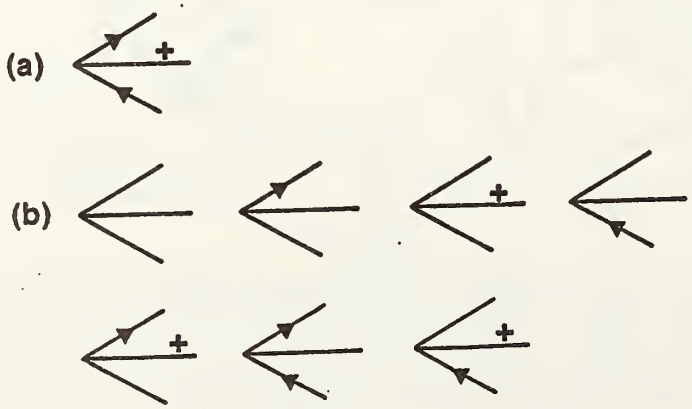


Figure 2.13. A vertex labeling operator, and the vertices to which it is applicable.

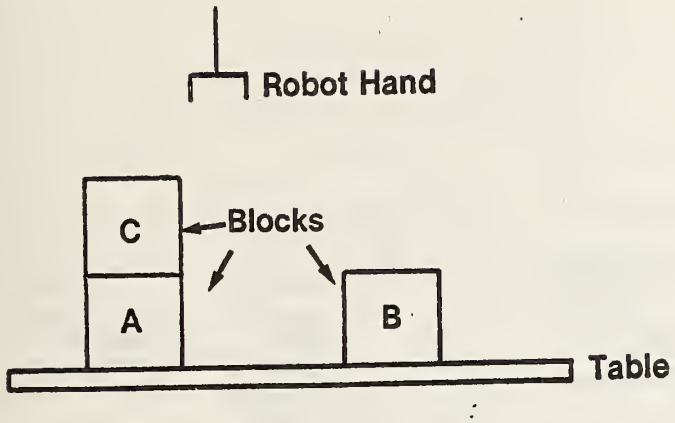


Figure 2.14. An example world state for STRIPS. The example is taken from Nilsson [1980], Chapter 7.

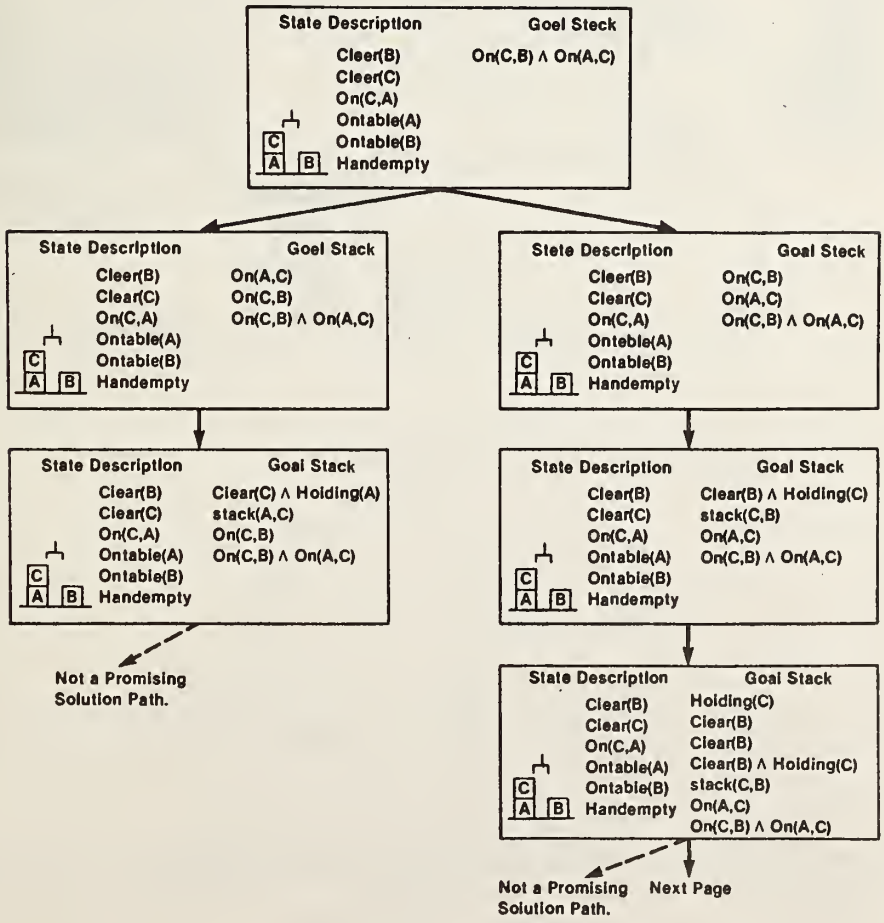


Figure 2.15. A search graph produced by SRTIPS (taken from Nilsson [1980], Chapter 7).

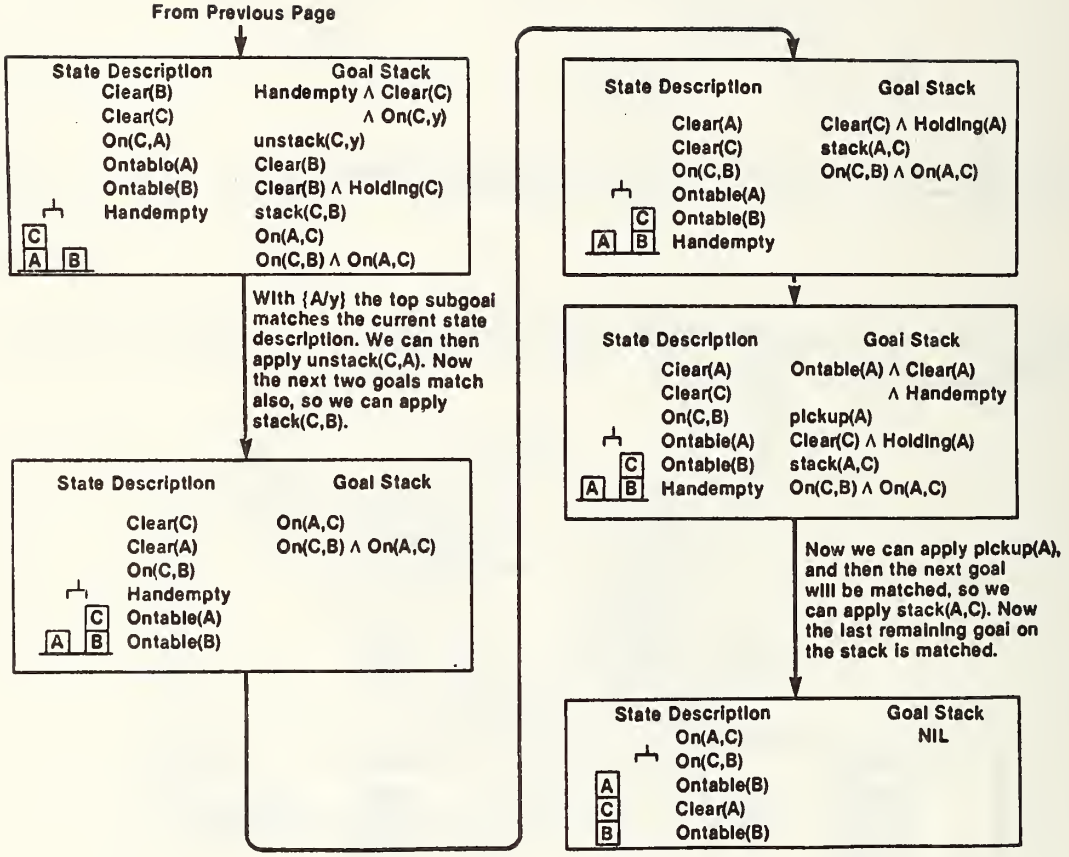


Figure 2.16. A continuation of Figure 2.15.

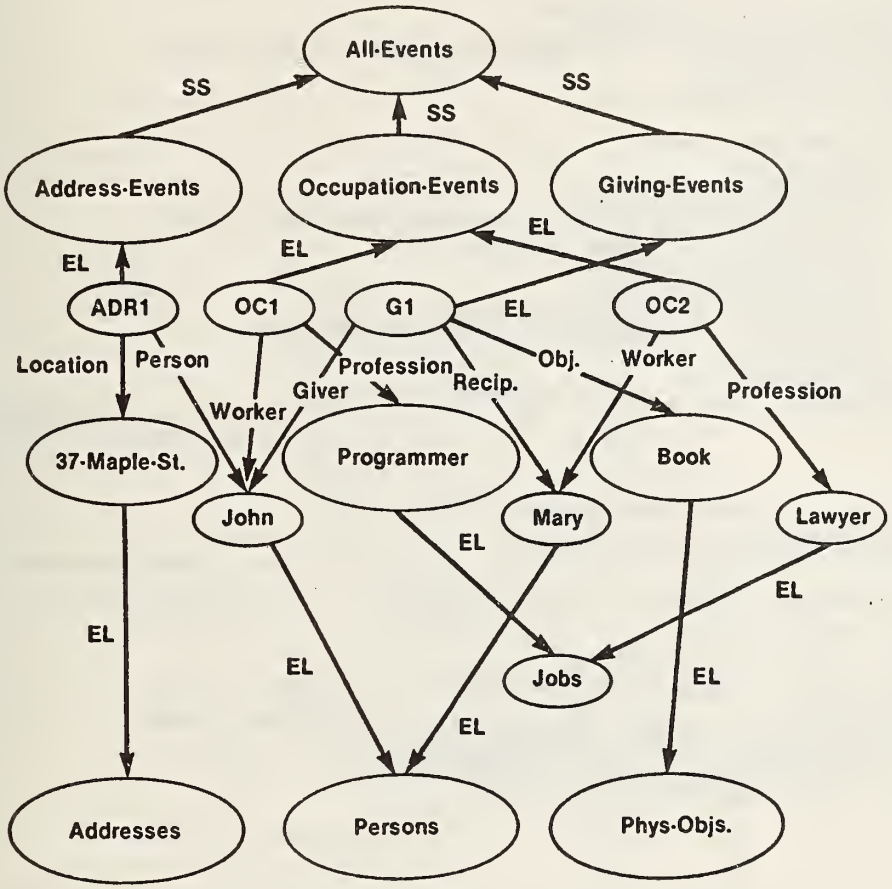


Figure 3.1. A simple sematic net [Nilsson 1980, p. 371].

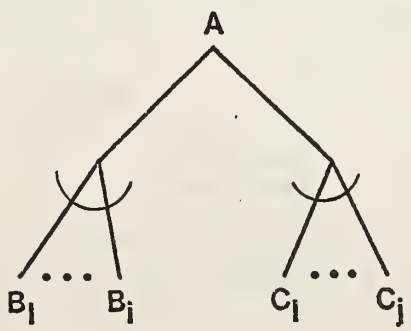


Figure 3.2. An AND/OR graph corresponding to the PROLOG statements

$$A{:-}B_1,\ B_2,\ldots,\ B_i$$

$$A{:-}C_1,\ C_2,\ldots,\ C_j.$$

```
Procedure FINDVALUEOF (item GOAL)
  begin item X; list L; rule R; premise_clause P;
  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 E 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 I;
  UPDATE_VALUE_OF(ATTRIBUTE_IN(CONCLUSION),VALUE_IN(CONCLUSION));
  I←ANTECEDENTRULES_ASSOCIATED_WITH(CONCLUSION);
  I←APPLY_METARULES(ATTRIBUTE_IN(CONCLUSION),I,0);
  for T∈ I do CONCLUDE(CONCLUSION_IN(T));
  end;
```

Figure 4.1. The control structure for MYCIN [Davis et al. 1977, p. 43].

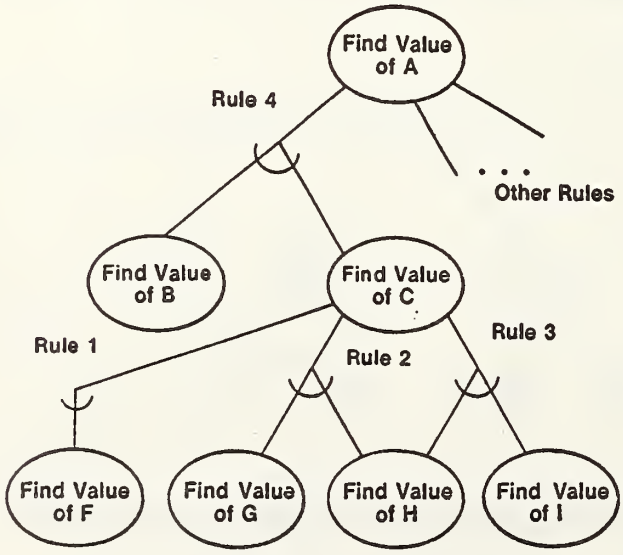


Figure 4.2. An AND/OR graph for MYCIN.

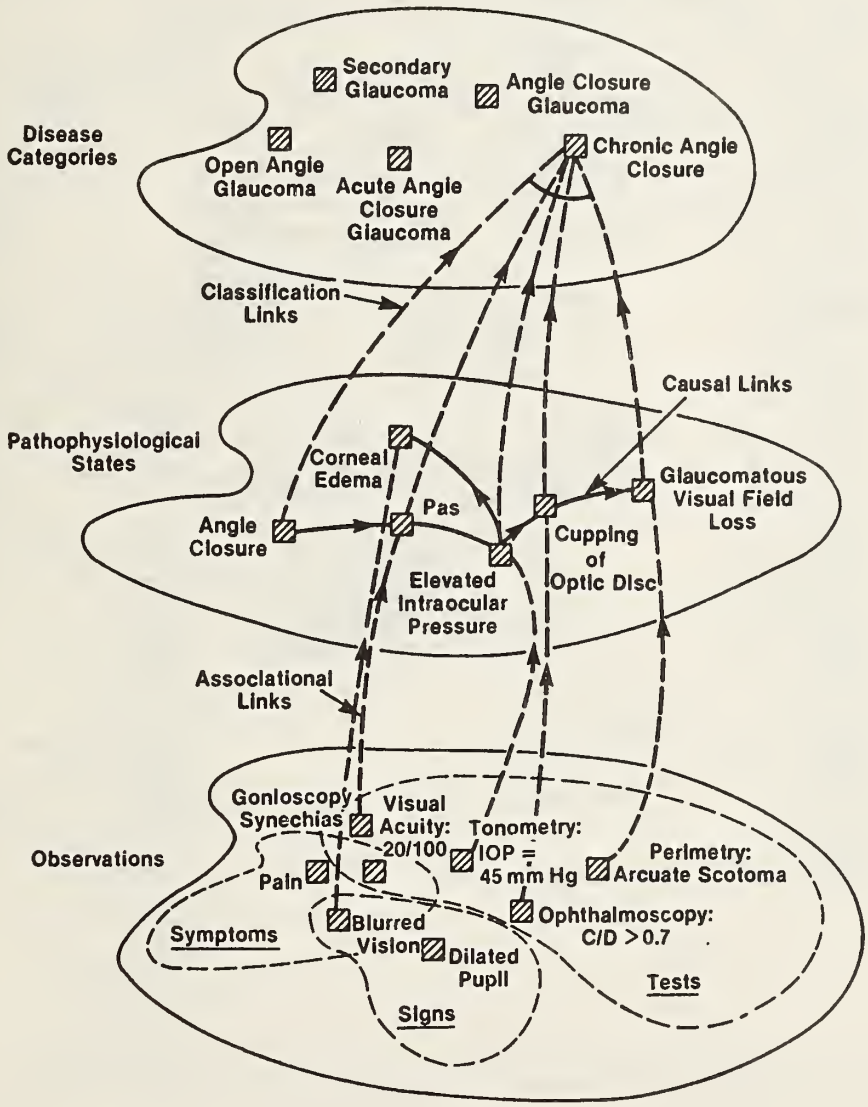


Figure 4.3. Three-level description of a disease process [Weiss et al. 1978, p. 148].

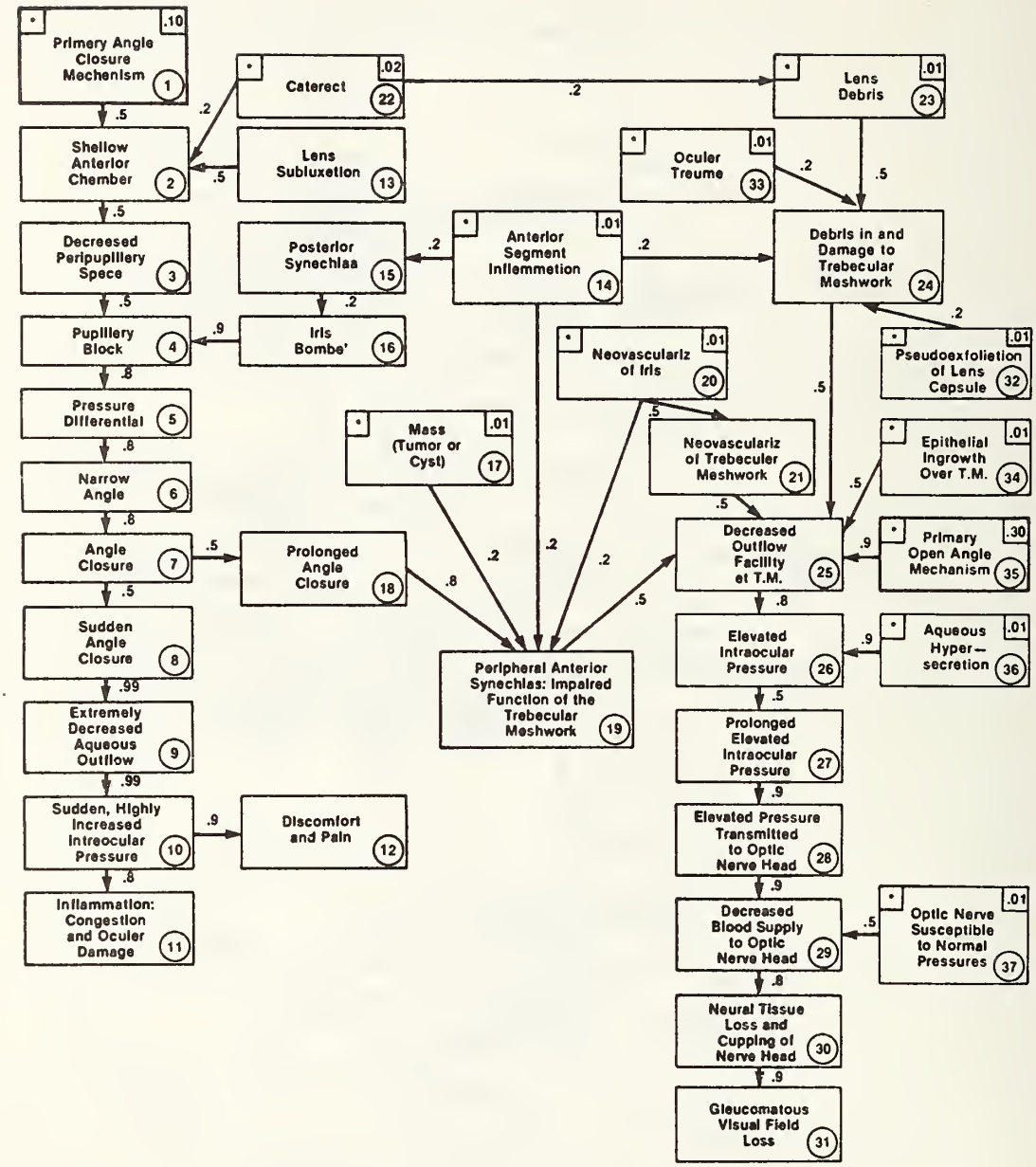


Figure 4.4. Partial causal network for glaucoma. States with no antecedent causes are indicated by asterisks. The circled numbers correspond to the state labels n_i used in the text [Weiss et al. 1978, p. 149].

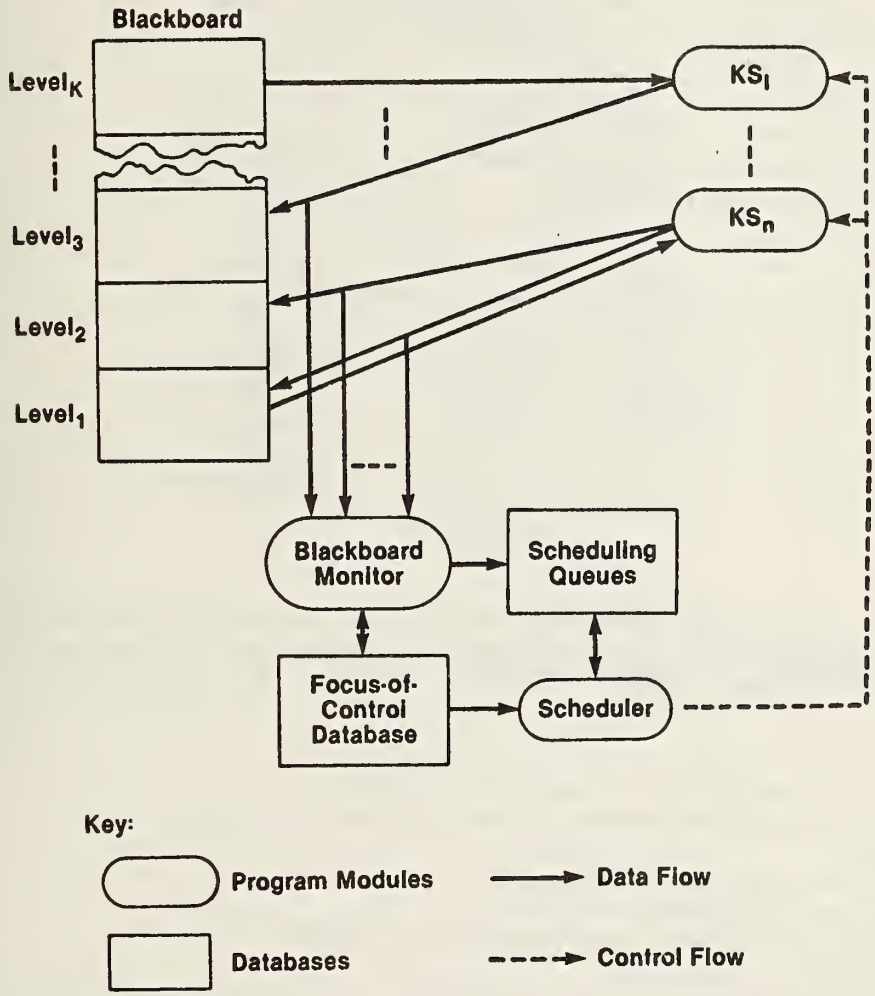


Figure 4.5. Schematic of the Hearsay-II architecture [Erman et al. 1980, p. 222].

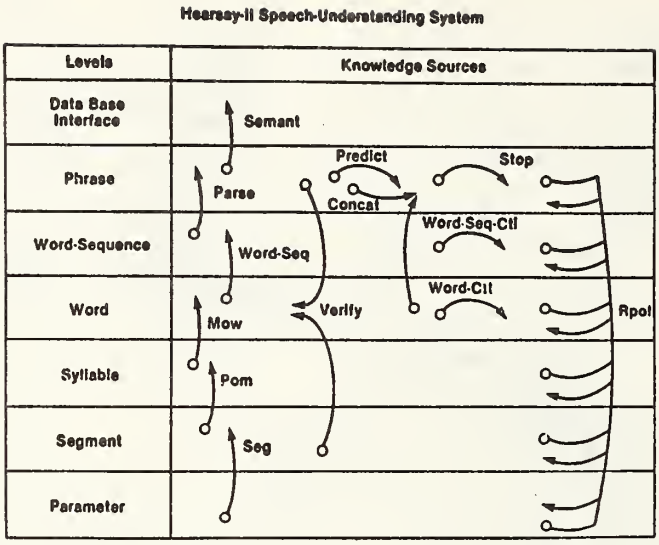


Figure 4.6. Levels and knowledge sources in Hearsay-II, as of September 1976. KS's are indicated by vertical arcs with the circled ends indicating output levels [Erman, 1980, p. 219].

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 which the user has queried.

Figure 4.7. Functional descriptions of a few of the Hearsay-II KS's [Erman 1980, p. 219].

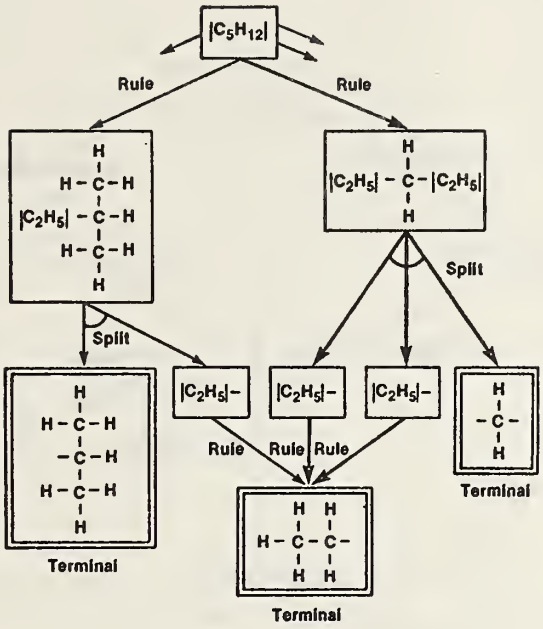


Figure 4.8. An AND/OR graph for an example DENDRAL problem [Nilsson, 1980, p. 44].

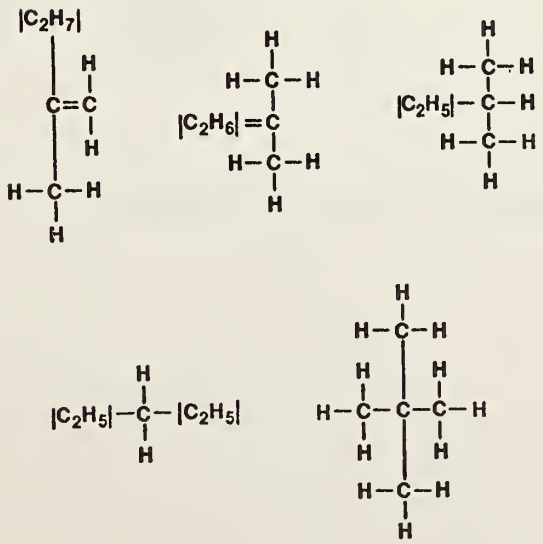


Figure 4.9. Partial structures proposed by the "generate" part of DENDRAL for C_2H_5 [Nilsson, 1980, p. 42].

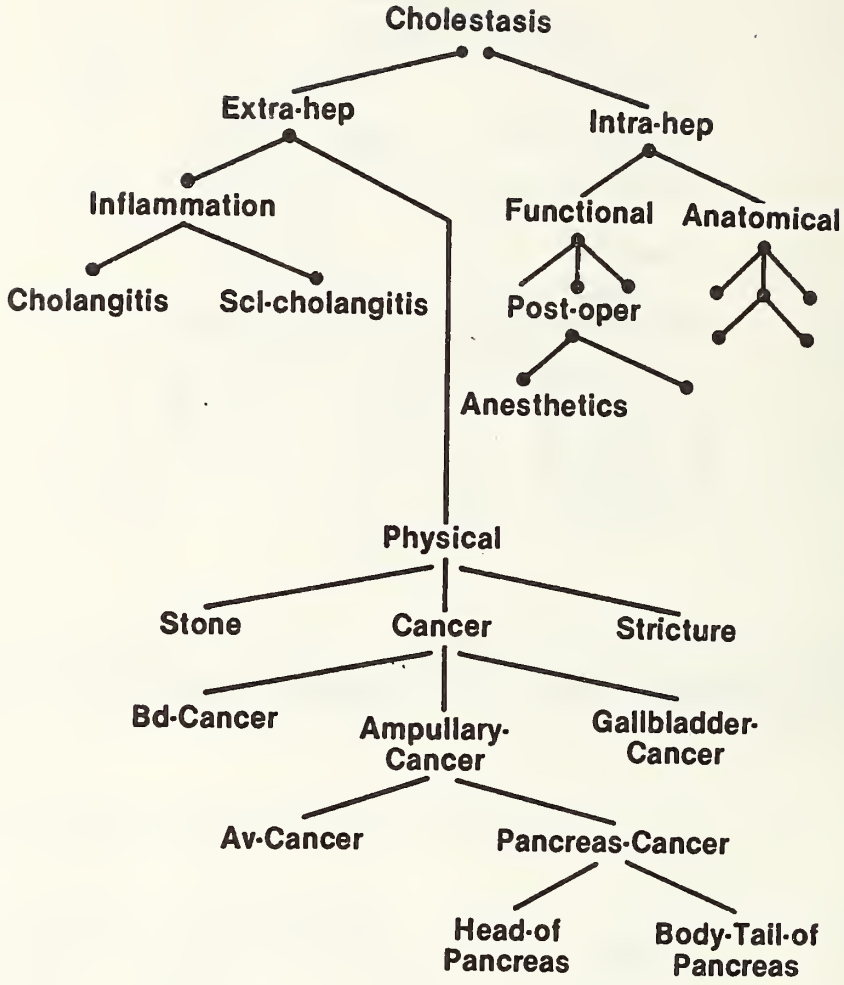


Figure 4.10. Conceptual structure of Cholestasis [Chandrasekaran et al., 1979].

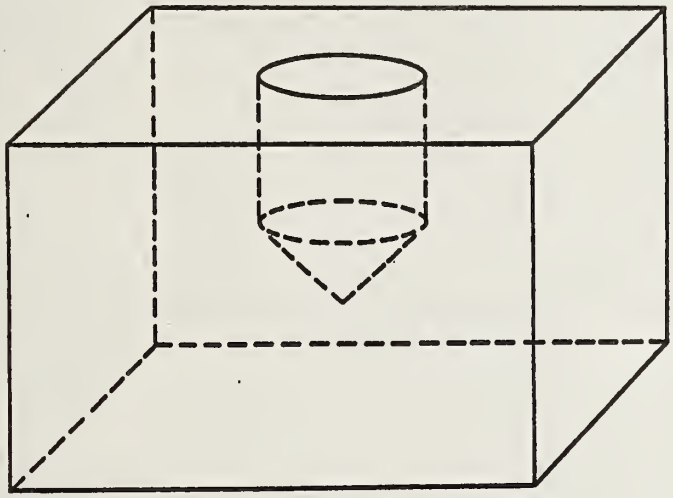


Figure 5.1. A typical drilled hole in an object.

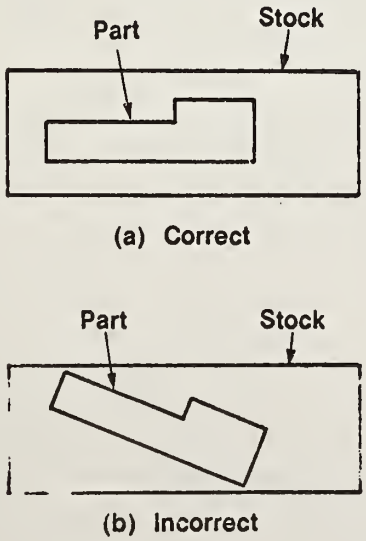


Figure 5.2. Correct and incorrect orientations of a part relative to a piece of stock.

NBS-114A (REV. 2-80)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NBSIR 81-2466	2. Performing Organ. Report No.	3. Publication Date February 1982

4. TITLE AND SUBTITLE

Expert Computer Systems, and Their Applicability to Automated Manufacturing

5. AUTHOR(S)

Dr. Dana S. Nau

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

This paper contains two main parts: a tutorial on techniques used in expert systems, and some recommendations for an automated process planning system for the Automated Manufacturing Research Facility at the National Bureau of Standards (NBS).

The tutorial portion of the paper consists of Sections 2, 3, and 4. Sections 2 and 3 discuss AI problem solving and knowledge representation techniques. Section 4 describes ways in which these techniques have been used to build computer systems which achieve a high level of performance on problems which normally require significant human expertise for their solution.

Section 5 contains a summary of the activities required for process planning in the Automated Manufacturing Research Facility (AMRF) at NBS, and recommendations for how to accomplish these activities. Section 6 contains recommendations for how an expert system could be designed to perform a process planning activity called process selection.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

AMRF; Artificial intelligence; Automated manufacturing; Expert systems; Knowledge-based systems; Knowledge engineering; Knowledge representation; Process planning; Problem solving.

13. AVAILABILITY

☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

☐ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES 110

15. Price $12.00