NASA Technical Memorandum 85839

An Overview of Artificial Intelligence and Robotics

VOLUME I — ARTIFICIAL INTELLIGENCE
Part C — Basic AI Topics

William B. Gevarter
OFFICE OF AERONAUTICS AND SPACE TECHNOLOGY
NASA Headquarters
Washington, D.C. 20546

National Aeronautics
and Space Administration

**Scientific and Technical
Information Branch**

1983

PREFACE

This is Part C of the three parts comprising Volume I on an overview of Artificial Intelligence (AI).

Part A. The Core Ingredients, NASA TM 85836, June 1983

Part B. Applications, NASA TM 85838, September 1983

Part C. Basic AI Topics, NASA TM 85839, October 1983

I. Artificial Intelligence and Automation

II. Search-Oriented Automated Problem Solving and Planning Techniques

III. Knowledge Representation

IV. Computational Logic

Intelligence is involved with knowledge and the access, manipulation, transformation and utilization of that knowledge for the purposes of problem solving and responding appropriately to new situations. Thus, to develop artificial intelligence, one must be concerned with such topics as how to represent knowledge in a computer, how to utilize it, and how to find an answer in a huge search space of possible solution paths. Therefore, this part of Volume I endeavors to provide readily understandable overviews of search-oriented problem solving, knowledge representation, and computational logic. These overviews are elaborations on the basic techniques briefly reviewed in Part A, and provide additional material not covered elsewhere in this series.

To enable the reader to relate artificial intelligence to the broader field of automation, this report opens with a discussion of mechanization, automation and AI, and how they interrelate.

It is anticipated that this report will prove useful to engineering and research managers, potential users and others seeking to obtain a basic understanding of artificial intelligence.

ACKNOWLEDGMENTS

I wish to thank the many people and organizations who have contributed to this report, both in providing information, and in reviewing the report and suggesting corrections, modifications and additions. I particularly would like to thank Terry Cronin of U.S. Army Signal Warfare Lab., Earl Sacerdoti of Machine Intelligence Corporation, Jack Minker of U. of MD., Harry Porta of JPL, Bob Hong and Paul Orlovski at Grumman Aerospace Corp., Jude Franklin and Y. T. Chien of the U.S. Navy Center for Applied Research in AI, Larry Wos of the Argonne Nat. Lab., and David R. Brown of SRI International for their review of portions of this report and their many helpful suggestions. However, the responsibility for any remaining errors or inaccuracies must remain with the author.

It is not the intent of NASA or the National Bureau of Standards to recommend or endorse any of the manufacturers or organizations named in this report, but simply to attempt to provide an overview of the AI field. However, in a diverse and rapidly changing field such as AI, important activities, organizations and products may not have been mentioned. Lack of such mention does not in any way imply that they are not also worthwhile. The author would appreciate having any such omissions or oversights and any other needed modifications called to his attention so that they can be considered for future reports.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont.)

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS (cont.)

I. ARTIFICIAL INTELLIGENCE AND AUTOMATION

A. Mechanization and Automation

To better understand what is meant by Artificial Intelligence (AI) and robotics it is helpful to step back a bit and first look at terms such as mechanization and automation. To do this we will try to synthesize the views of others who have approached this problem.

The original industrial revolution was based on mechanization. Mechanization was the use of machines to take over some of the previous muscle jobs performed by either animals or human beings. Laurie (1979) states:

> When we apply ordinary production techniques—the application of leverage and power—to a process, we are mechanizing it. Automation involves a good deal more...Automated devices are truly automated when feedback information automatically causes the machinery to adjust to reachieve the norm. The internal adjustments of the machine or system are made by servomechanisms (p. 355).
>
> Automation is the achievement of self-directing productive activity as a result of the combination of mechanization and computation...(p. 15).

Peter Marsh (1981, pp. 419-420) elaborates further on mechanized machines, automatic devices and automated devices:

> [The classification of mechanization] depends on whether machines or combinations of animals and people are responsible for the three fundamental elements that occur in every activity (human or otherwise)—power, action and control. [Simple mechanized devices] need a human to control them. If a mechanical device is responsible for control, however, we have a self-acting or automatic device. Automatic devices are not the same as automated ones.... automation equals mechanization plus automatic control plus one (or more) of three extra control features—a "systems" approach, programmability or feedback.
>
> **Extras that make automation**
>
> With a systems approach, factories make parts by passing them through successive stages of a manufacturing process without people intervening. Thus the transfer lines of car factories in the 1930s count as automated systems.
>
> With programmability—the second of the three "extras" that define automation—an automated system can do more than one kind of job. Hence an industrial robot is an automated, not an automatic, device. The computer that controls it can be fed different software to make the machine do different things—for example, spray paint or weld bits of metal together. Finally, [external] feedback makes an automatic machine alter its routine according to changes that take place around it. An automatic lathe with feedback—in which, for instance, a sensor detects that the metal it is cutting is wrongly shaped and so instructs the machine to stop—is thus an automated device. It is clearly more useful than a lathe without this feature.

B. Tools, Machines, Teleoperators, Robots

To extend the concepts of mechanization and automation further, we will consider tools, machines, teleoperators and robots. To do this, we will utilize Marsh's (1981) basic elements—power, action and control.

Tool:	A device used to perform an action. If used by a human, the person provides the power and control.
Machine:	A device that utilizes non-human power to do an action. For a simple machine the human provides the control.
Teleoperator:	A machine capable of action at a distance under the control of a human.

Robot: A flexible machine capable of controlling its own actions for a variety of tasks utilizing stored programs. Basic task flexibility is achieved by its capability of being reprogrammed. More advanced—intelligent—robots would be capable of setting their own goals, planning their own actions and correcting for variations in their environment.

C. Computation and Artificial Intelligence

Laurie (1979, p. 15) defines a computer as "...an electronic device capable of following an intellectual map. We call the map a program." Arden (1980, p. 9) suggests that "...computer science is the study of the design, analysis, and execution of algorithms* in order to better understand and extend the applicability of computer systems."

Though everyone agrees that "Artificial Intelligence" (AI) is difficult to define precisely, the most commonly accepted definition is that "Artificial Intelligence is the branch of computer science devoted to programming computers to carry out tasks that if carried out by human beings would require intelligence."

A slightly different definition is giving by Duda et al. (1979, p. 728):

> Artificial Intelligence (AI) is the subfield of computer science concerned with the use of computers in tasks that are normally considered to require knowledge, perception, reasoning, learning, understanding and similar cognitive abilities. Thus, the goal of AI is a qualitative expansion of computer capabilities.

Nilsson (1980, p. 2) notes that:

> AI has also embraced the larger scientific goal of constructing an information-processing theory of intelligence. If such a science of intelligence could be developed, it would guide the design of intelligent machines as well as explicate intelligent behavior as it occurs in humans and other animals. Since the development of such a general theory is still very much a goal, rather than an accomplishment of AI, we limit our attention here to those principles that are relevant to the engineering goal of building intelligent machines.

More recently, Nilsson (1981/1982) indicated that he would like to narrow the working definition of AI even further to the central processes of intelligence. He thus states:

> With regard to humans, I am inclined to consider as *central* those cognitive processes that are involved in reasoning and planning. Work on automatic methods of deduction, commonsense reasoning, plan synthesis, and natural-language understanding and generation are examples of AI research on central processes.
>
> Perhaps as important as the processes themselves is the "knowledge" they manipulate. In fact, the subject of knowledge representation formalisms is a good starting point for a more detailed explanation of just what I think AI is.

Arden (1980, pp. 22 and 23) states:

> Though "intelligent behavior" is difficult to define, and is currently understood differently by different people, there has been some convergence of views within the AI community as the technical requirements for the computer solution of certain classes of problems becomes better understood. To be sure, the human solution of a complex equation might be classified as intelligent behavior, while the corresponding action by a machine might not be so classified, even though both machine and man had been programmed for (learn) the process. One possible requirement is that there be something unstructured, something nondeterministic, for the solution process to qualify as intelligent. Another is that it depends on the knowledge that must be used in obtaining the solution, or on the methods used...
>
> Another important aspect is the use of heuristic rules** of the kind humans use to solve problems. Although, in general, such rules cannot be proved effective, they often lead to solutions. Some computer scientists argue that heuristic programming better describes the field now called "artificial intelligence."

---

*The Dictionary of Electronics, *(Fort Worth: Radio Shack, 1975) defines algorithm as, "A set of rules or processes for solving a problem in a finite number of steps."*

***Heuristics are "rules-of-thumb" (compiled experience) used to help guide problem solving. They do not guarantee a solution as algorithms do.*

Hayes-Roth (1981, p. 1) notes that:

> AI provides techniques for flexible, non-numerical problem-solving. These techniques include symbolic information processing, heuristic programming, knowledge representation, and automated reasoning. No other fields or alternative technologies exist with comparable capabilities. And nearly all complicated problems require most of these techniques. Many forces combine to identify AI as the central technology for exploitation. Systems that reason and choose appropriate courses of action can be faster, cheaper, and more effective and viable than rigid ones. To make such choices in realistically complex situations, the system needs at least rudimentary understanding of mundane phenomena.

In summary, AI is concerned with intelligent behavior, primarily with non-numeric processes that involve complexity, uncertainty and ambiguity and for which known algorithmic solutions do not usually exist. Unlike conventional computer programming, it is knowledge based, almost invariably involves search, and uses heuristics to guide the solution process.*

Thus AI can be considered to be built upon

1. Knowledge of the domain of interest.
2. Methods for operating on the knowledge.
3. Control structures for choosing the appropriate methods and modifying the data base (system status) as required. This contrasts with conventional computer programs which utilize known algorithms for solution, are primarily numeric (number crunching) in nature rather than symbolic manipulation, and in general do not require knowledge to guide the solution.

D. Relationship of AI to Automation

Artificial Intelligence may be considered to be the top layer of control on the hierarchical road to autonomous machines. This is illustrated in Figure 1-1, derived from Marsh (1981).

However, AI includes a large area of activity which is not normally included in automation, e.g.:

natural language processing
perception and pattern recognition
intelligent information storage and retrieval
game playing
automatic programming
computational logic
problem solving
expert systems

Nevertheless, as Computer Integrated Manufacturing and intelligent robots emerge, AI will have a major role to play. AI contributions to perception and object-oriented programming are reviewed by Brady (1984) for this new breed of robots.

E. AI and Other Fields

Duda, et al. (1978, pp. 729-730) state:

> Historically, AI has both borrowed from and contributed to other closely related disciplines concerned with advanced methods for information processing. Thus, links exist between AI and aspects of such theoretical areas as mathematical logic, operations research, decision theory, information theory, pattern recognition and mathematical linguistics. In addition, research in AI has stimulated important developments in software technology, particularly in the area of advanced programming languages. What distinguishes AI from these related fields, however, is its central concern with all of the mechanisms of intelligence.

**As AI matures, the expectations associated with it are increasing. Schank (1983) states that it is time to demand learning capability from AI programs. He thus suggests a new definition: "AI is the science of endowing programs with the ability to change themselves for the better as a result of their own experiences."*

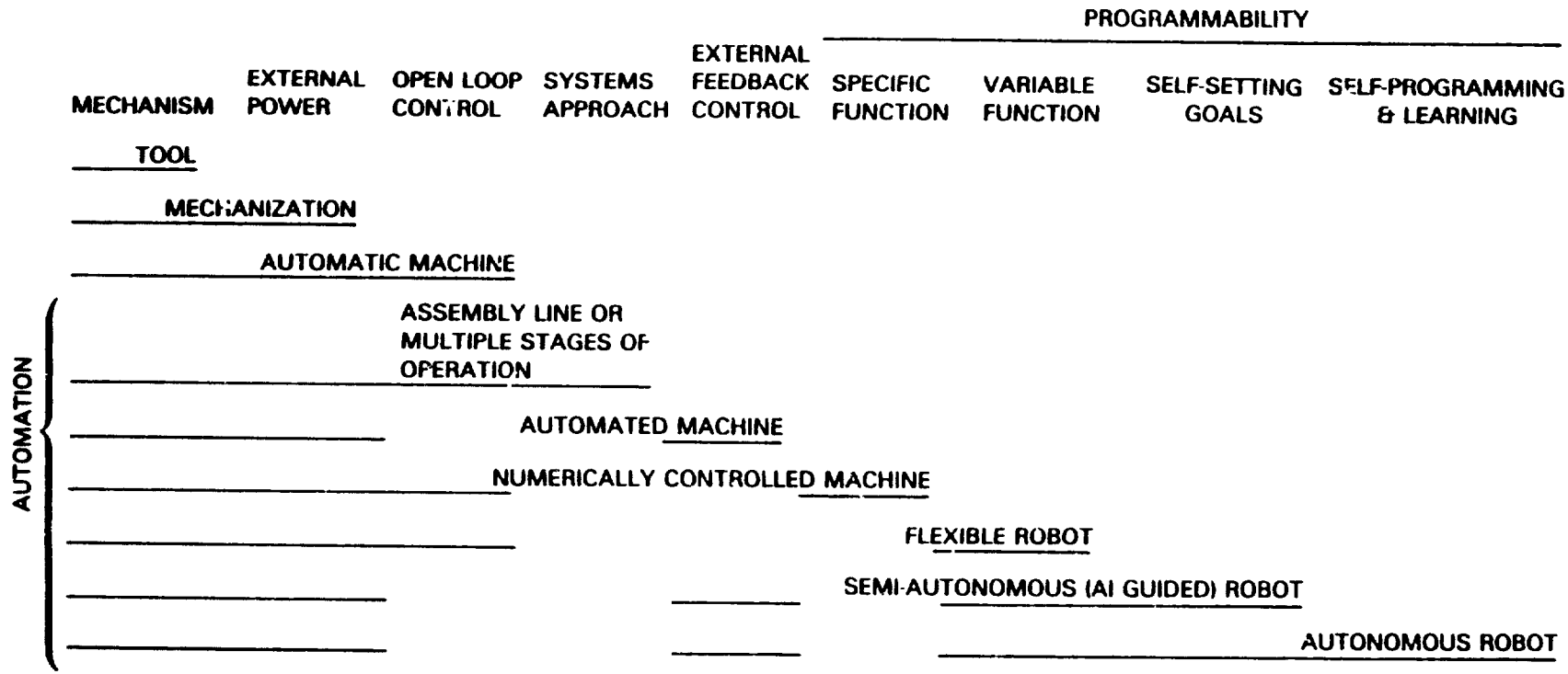


Figure I-1. Mechanization, Automation and Artificial Intelligence

Newell, et al. (1978, pp. 19-20), in referring to AI as Machine Intelligence (MI), note that:

> The aims of research in MI are so broad—nothing less than extending the capabilities of symbol processing machines for intelligent action—that it is not always easy to identify applications as emanating specifically from research in this field.
>
> Consider the important contributions that MI has already made. List processing has become an integral part of programming technology. The abstract theory of programming owes much of its early impetus to LISP. The initiation of verification of programs is likewise due strongly to work in machine intelligence. This is also true of the subfield of symbolic mathematics. Even some of the concepts of structured programming can be traced back to early MI languages—concepts of extensive hierarchization and recursion, for instance. Similarly, fundamental ideas of heuristic search are used widely in operations research programs for domains where powerful optimization techniques are unavailable or inadequate. Heuristic search is especially necessary today for handling large combinatorial problems, such as job-shop scheduling.
>
> As these examples illustrate, MI applications are characterized by helping to initiate fields of application and then becoming freely mixed with independent invention and development from within the field. For example, in operations research, branch-and-bound techniques for limiting search (analogous to alpha-beta procedures in game trees) and optimal scheduling algorithms came not from MI, but from within operations research. Structured programming and symbolic mathematics have run essentially independent courses from work in MI. An extreme example of this "initiation" syndrome in MI applications was the strong effect of MI on the initiation of time sharing, but with little specific technical transfer.
>
> The reported MI research has some ties to specific areas of applications as we note below. However, its ultimate fate is likely to be similar to the examples above, in which most of the applications will not be identified as MI. For instance, work on control structures is of fundamental significance to future applications. Every intelligent system must employ a control structure capable of using partial knowledge, discovering relevant knowledge, coping with pervasive error, etc. But as we discover effective system organizations, they will become assimilated into the application area, their further development being seen as part of the application. Similarly, progress in heuristic search, being of general utility, will diffuse through various applications fields.

Barrow (1979, p. 3) agrees, stating:

> It is probably safe to say that AI seeks to understand and model virtually all human intellectual activity. In doing so, it has drawn from or contributed to many other disciplines, particularly psychology, mathematical linguistics, mathematical logic, operations research, decision theory, pattern recognition, and computer science. It has stimulated important developments in software technology, especially concerning advanced programming languages and systems.

It is interesting to note that no mention is made of an intersection of control theory and AI. As control theory has primarily dealt with analog or numeric computation in relation to servomechanisms, and AI has primarily dealt with symbolic manipulation, this lack of intersection is not too surprising. However, this situation is beginning to change. DeJong (1983) examines the role of AI in control, and Sauers and Walsh (1983) indicate requirements and architectures for future expert systems that can operate in the real-time environment associated with control. A technique for coordinating control and knowledge-based components for an autonomous mobile robot guidance system is proposed by Harmon (1983).

Saridis (1979) lays out a taxonomy of increasingly sophisticated control systems on the path to intelligent controls. The furthest point on the path he identified as AI. Based on Saridis, Figure I-2 indicates the increasing level of sophistication in control.

Saridis (1979, p. 1129) states that, "Intelligent controls should represent the perfect interface between control hardware and a digital computer for higher level decision making according to the principle of increasing intelligence with decreasing precision in a hierarchical control structure." To state it another way, AI can provide the topmost level in a control structure. "Such systems, implemented by the fast large modern digital computer, can solve problems, identify objects, or plan a strategy for a complicated function of a system" (Saridis, 1979, p. 1129).

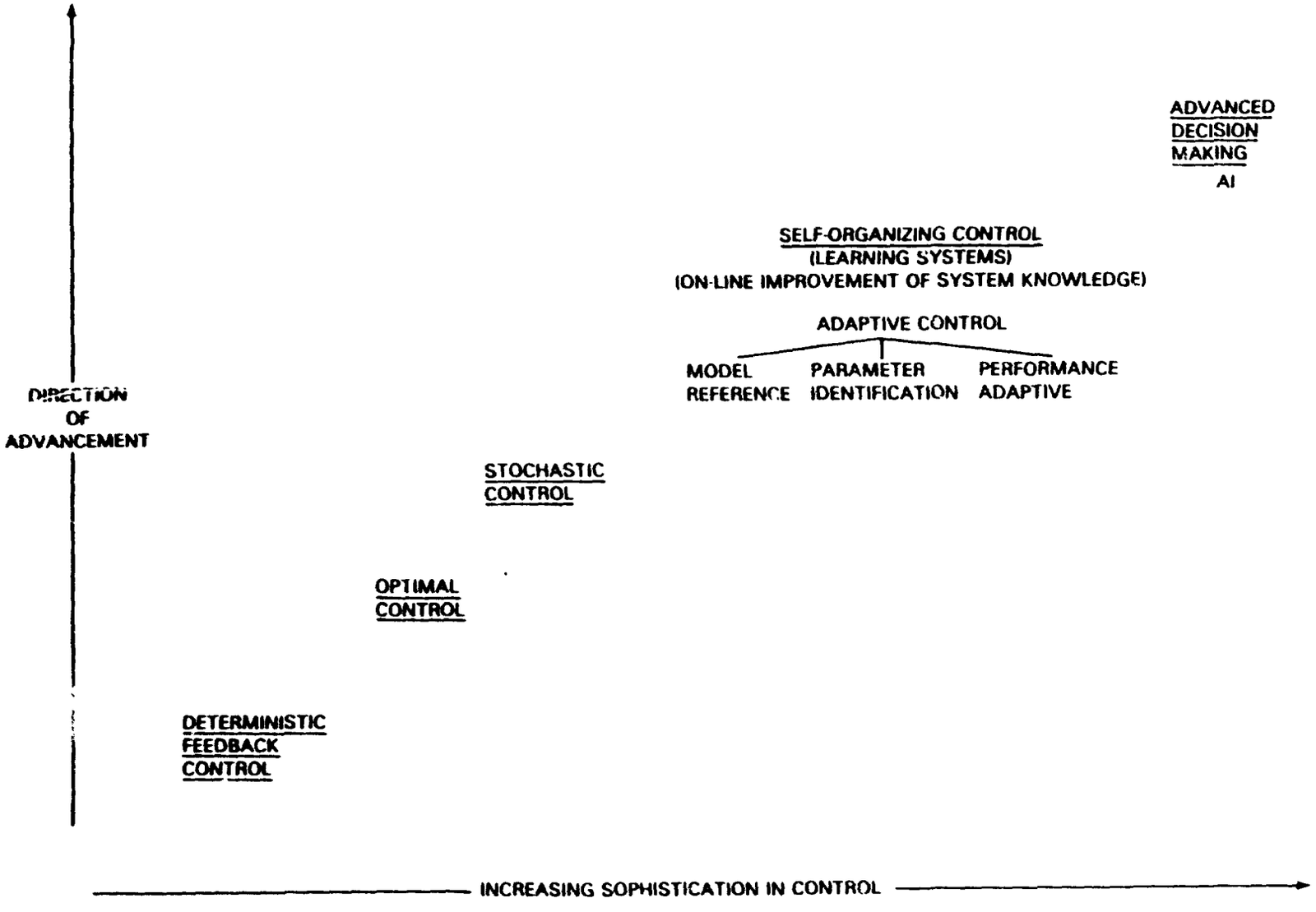


Figure I-2. Increasing Sophistication on the Path to Intelligent Controls

Albus, et al. (1981, 1983) have developed a theory of hierarchical control that can exhibit learning and thereby provide learned reflex responses to complex situations. It also provides for the problem-solving and planning functions that are normally associated with the highest level of intelligent action that is considered to be the domain of AI, and incorporates expert system rules for error correction at the intermediate levels of the control hierarchy. It therefore appears that AI will not only be a key to the development of intelligent robots and factories of the future, but is destined to become a central ingredient in advanced control systems as well.

REFERENCES

- Albus, J.S., et al., "Theory and Practice of Hierarchical Control," *Proceedings of the Twenty-third IEEE Computer Society International Conference,* Sept. 1981, pp. 18-39.
- Albus, J.S., McLean, C.R., Barbera, A.J. and Fitzgerald, M.L., "Hierarchical Control for Robots in an Automated Factory," *Proc. of 13th Inter. Symp. on Industrial Robots and Robot 7,* Chicago, IL, April 17-21, 1983, pp. 13-29 to 13-43.
- Arden, B.W. (Ed)., *What Can Be Automated*, Cambridge: MIT Press, 1980.
- Barrow, H., "Artificial Intelligence: State-of-the Art," Menlo Park: SRI Intl. Technical Note 198, October 1979.
- Brady, M., "Artificial Intelligence and Robotics," To be published in *Artificial Intelligence,* M. Brady and L. Gerhardt (Eds.), New York: Springer Verlag, 1984.
- DeJong, K., "Intelligent Control: Integrating AI and Control Theory," *Proc. IEEE Trends and Applications 1983*, Gaithersburg, MD, May 25-26, 1983.
- Duda, R.O., et al., "State of Technology in Artificial Intelligence," in *Research Directions in Software Technology*, Wenger, P. (Ed.), Cambridge: MIT Press, 1979, pp. 729-749.
- Graham, N., *Artificial Intelligence*, Blue Ridge Summit, PA: Tab Books, 1979.
- Harmon, S.Y., "Coordination Between Control and Knowledge Based Systems for Autonomous Vehicle Guidance," *Proc of IEEE Trends and Applications 1983*, Gaithersburg, MD, May 25-26, 1983, pp. 8-11.
- Hayes-Roth, F., "AI: The New Wave—A Technical Tutorial for R&D Management," Santa Monica, Rand Corp., 1981 (AIAA-81-0827).
- Laurie, E.J., *Computers, Automation and Society*, Homewood, IL: R.D. Irwin, Inc., 1979.
- Marsh, P., "The Mechanization of Mankind," *New Scientist*, 12 February 1981, pp. 418-421.
- Newell, A., et al., "Research in Information Processing," Pittsburgh: Carnegie-Mellon Univ., AD/A-064 845, Final Report, December 1978.
- Nilsson, N.J., *Principles of Artificial Intelligence,* Palo Alto: Tioga Publishing Co., 1980.
- Nilsson, N.J., "Artificial Intelligence: Engineering Science, or Slogan," *AI Magazine*, Vol. 3, No. 1, Winter 1981/1982 pp. 2-9.
- Sardis, G.N., "Toward the Realization of Intelligent Controls," *Proc. of the IEEE,* Vol. 67, No. 8, Aug. 1979, pp. 1115-1132.
- Sauers, R. and Walsh, R., "On Requirements of Future Expert Systems," *Proc. of the 8th Inter. Joint Conf. on AI: IJCAI-83*, 8-12 Aug 1983, Karlsruhe, W. Germany. Los Altos, CA: W. Kaufmann, 1983.
- Schank, R.C., "The Current State of AI: One Man's Opinion," *AI Magazine,* Vol. 4, No. 1, Winter-Spring 1983, pp. 3-8.

II. SEARCH-ORIENTED AUTOMATED PROBLEM SOLVING AND PLANNING TECHNIQUES

This chapter provides an overview of search-oriented automated problem solving and planning techniques. It endeavors to present the basic approaches to automated problem-solving at a level where the concepts involved can be readily understood. It also provides an indication of the state of the art and current and future research.

A. AI as Problem Solving

One way of viewing intelligent behavior is as problem-solving. Many AI tasks can naturally be viewed this way, and most AI programs draw much of their strength from their problem-solving components. AI applications that have strong problem-solving components include scene analysis, natural language understanding, theorem proving, task planning, expert systems, game playing, and information retrieval and extraction.

Two important types of problem solving tasks are 1) synthesizing a set of actions (a plan) to achieve a goal and 2) deduction. The latter involves deducing (or inferring) conclusions from data or a given set of propositions (applications include theorem proving and information retrieval). In this chapter we will restrict ourselves to action synthesis, leaving a review of deduction for Chapter IV.

Many tasks can be formulated in terms of: given a goal, how do we achieve it? If direct methods are not available for solution, as is the usual case in AI problems, then a search procedure to select from the various possible alternatives is required. Thus, finding efficient search methods is one of the central issues in automated problem solving.

B. Elements of a Problem Solver

All problems have certain common aspects: an initial situation, a goal (desired situation) and certain operators (procedures or generalized actions) that can be used for changing situations. In solving the problem, a control strategy is used to apply the operators to the situations to try to achieve the goal. This is illustrated in Figure II-1, where we observe a control strategy operating on the procedures to generate a sequence of actions (called a plan) to transform the initial conditions in the situation into the goal conditions. Normally, there are also constraints and preconditions (conditions necessary for a specific procedure to be applied) which must be satisfied in generating a solution. In the process of trying to generate a plan, it is necessary for the problem solver to keep track of the actions tried and the effects of these actions on the system state. Figure II-2 is a restatement of Figure II-1 in which we can view the operators as manipulating the data base (representing the problem status) to change the current situation (system state).

C. State Graphs as an Aid to Problem Representation

One easy way to focus on the relationships between the operators and the states is through the use of state graphs. State graphs are networks made up of points (called nodes) connected by lines (called arcs). For our purposes, we let nodes correspond to system states and arcs correspond to operators. Figure II-3 illustrates a state graph for a simple problem (such as finding the simplest route from city A to city D). Note that there are several sequences (of different lengths) that will achieve goal state D, as well as a dead-end F.

The various paths through a state graph can be represented, as shown in Figure II-4, as a hierarchical structure called a tree. The solution paths run from the initial state along the branches and

terminate on the leaves (terminal nodes) labelled "goal state." We could have also generated a tree of paths starting from the goal state, as shown in Figure II-5. From Figures II-4 or II-5, it is apparent that the plan with the smallest sequence is to first use operator R, then operator S.

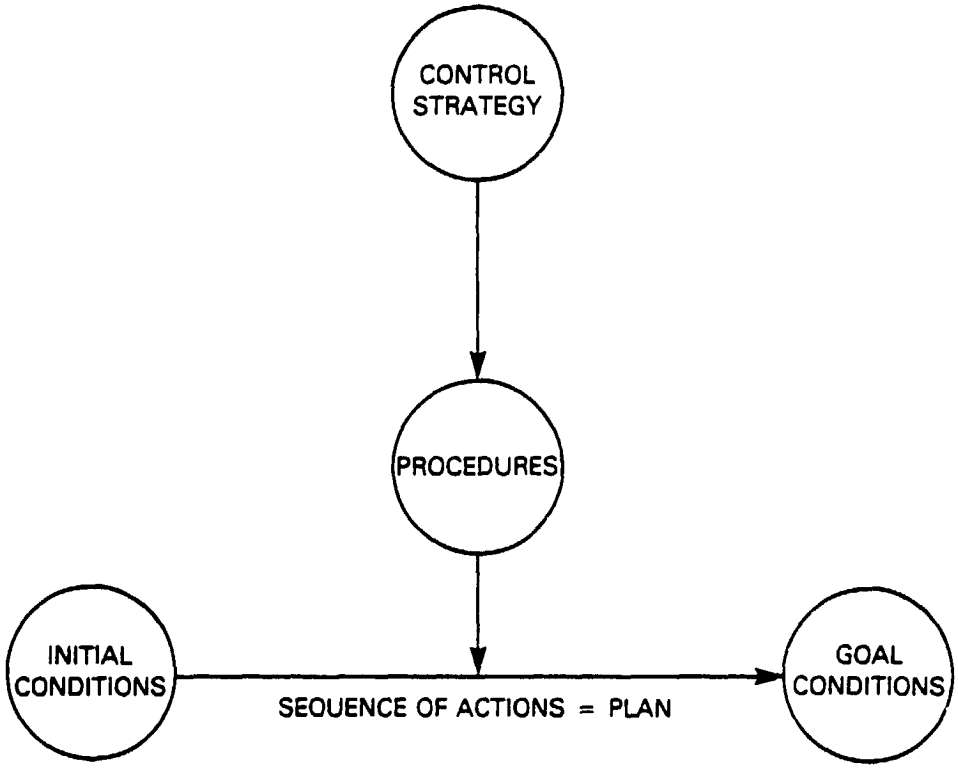


Figure II-1. Problem Solving

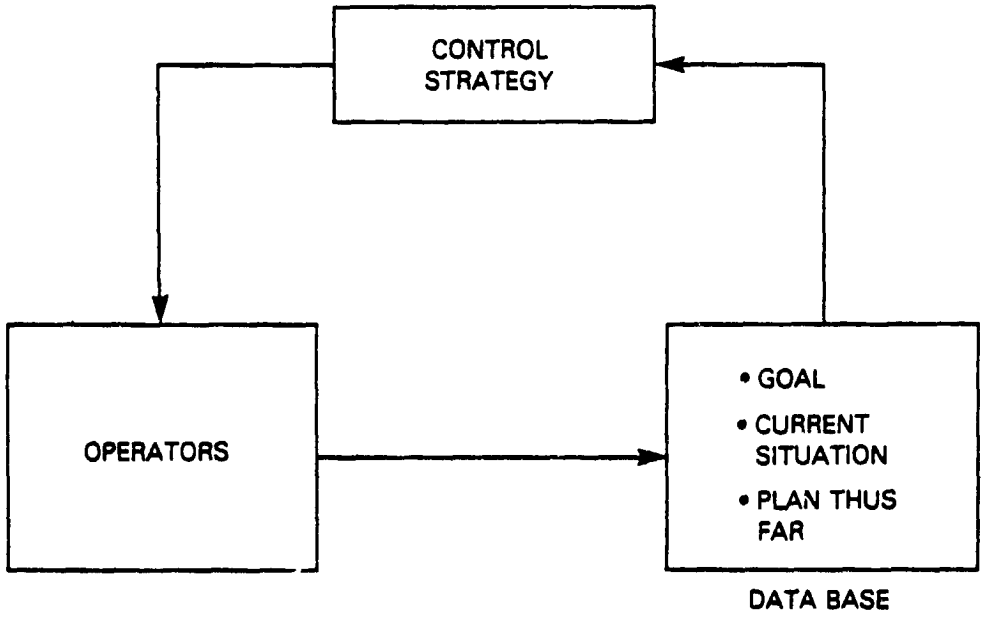


Figure II-2. Automated Problem Solving Relationships

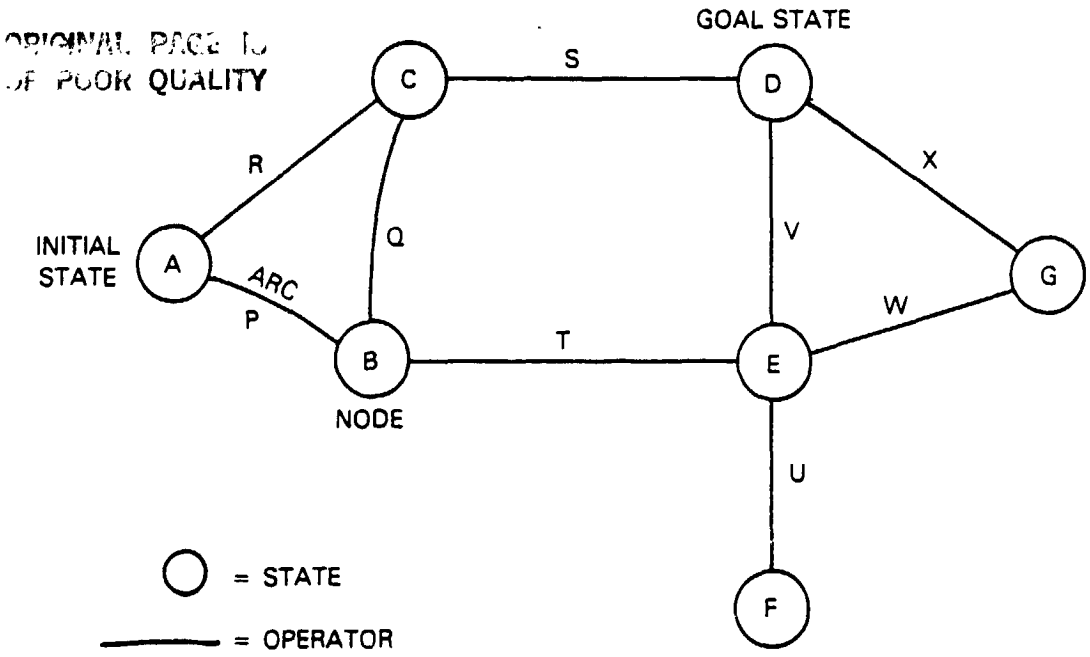


Figure II-3. State Graph for a Simple Problem

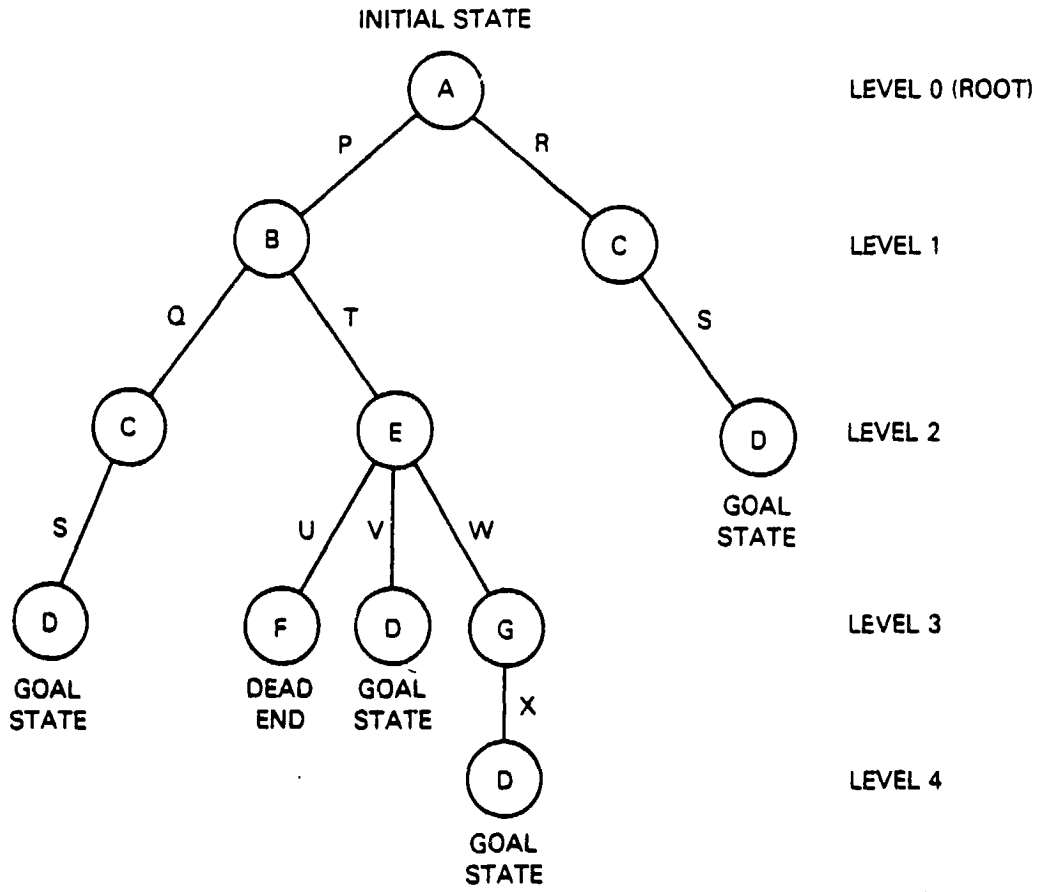


Figure II-4. Tree Representation of Paths Thru the State Graph of Figure II-3

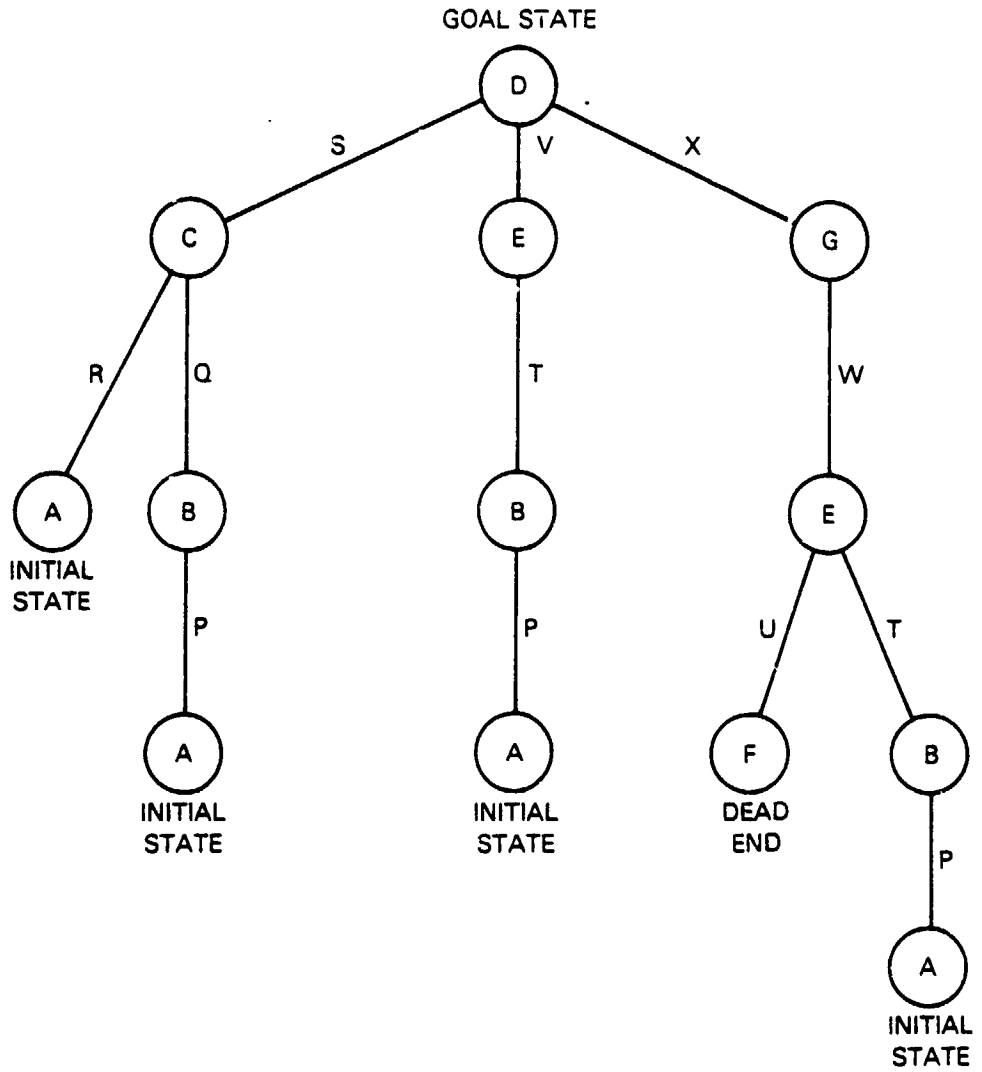


Figure II-5. Working Backwards from the Goal State

For a large complex problem, it is obviously too cumbersome to explicitly draw such trees of all the possibilities and directly examine them for the best solution. Thus, the tree is usually implicit; the computer generating branches and nodes as it searches for a solution.

D. Reasoning Forward and Backward

In searching for a solution we may reason forward from the initial state, as in Figure II-4, or we may reason backward from the goal, as in Figure II-5 (or both). Forward reasoning is said to be data driven or bottom-up. Backward reasoning is said to be goal-directed or top-down.

E. Problem Solving Using Blind Search

If a procedure for successfully generating a solution in a reasonable time (an algorithm) is known, it is applied and the problem is solved. Unfortunately, for many important problems no such algorithm is known and a search procedure is required.

For fairly simple problems, a straightforward, but time-consuming, approach is blind search, where we select some ordering scheme for the search and apply it until the answer is found.

The search proceeds by successively generating and examining the branches emanating from the nodes, starting with the root node and proceeding along generated branches to new nodes. The search tree grows as operators are applied to the nodes and the various paths explored. A node is referred to as "open" if branches have not yet been generated from it.

As indicated in Figure II-4, each node can be assigned a level. The root node is at level 0, its immediate successors at level 1, and so on, with the level number being referred to as the depth.

1. Breadth-First Search

In this approach, the nodes of the search tree are generated and examined level by level starting from the root node. No nodes at a deeper level are examined until all nodes at the previous level have been explored. Breadth-first search always finds the shortest number of steps to the goal. (However, this may not be the most desirable or cheapest solution, because of the different costs associated with applying the various operators. See following discussion on heuristic search.)

2. Depth-First Search

As a search proceeds, new nodes are generated from the node currently being examined. These successor nodes are called children and the generating node is called the parent. A depth-first search is one which always continues in the parent-to-child direction until forced to backtrack. To prevent consideration of paths that are too long, a depth bound is often specified.

A depth-first search does not necessarily find the shortest solution, but often can be programmed to minimize memory requirements by only saving in memory the path currently being explored.

3. Backward Chaining

Backward chaining is a name given to depth-first, backward reasoning—an important search strategy. An operator is chosen that would achieve the goal if selected. If it is applicable in the initial state, it is applied and a solution has been found. If not, operators that would achieve the preconditions required for its applicability are sought and the search continues recursively until a sequence of operators are found that transform the initial state into the goal state. If the search fails, the program backtracks and a new candidate operator is selected that would achieve the goal if applied, and the process is repeated.

For problems requiring only a small amount of search, backward chaining strategies are often perfectly adequate and efficient. For larger problems, it is critical that the correct operator be chosen first almost always, because this strategy follows out a line of action fully before rejecting it, which can result in very lengthy searches.

4. Problem Reduction

A generalization of backward chaining is problem reduction. Very often to satisfy a goal, several subproblems (conjuncts) must be satisfied simultaneously. For this case of backward reasoning, applying an operator may divide the problem into a set of subproblems, each of which may be significantly simpler to solve than the original problem.

A good example of problem reduction is readying a space vehicle for launch, as indicated in Figure II-6. Note that we can represent the goal—spacecraft ready to launch—as a conjunction of subgoals, e.g., spacecraft fueled, all systems checked, power on. These in turn can consist either of a set of simultaneous ("AND") subgoals, or of one of several acceptable alternatives ("OR" subgoals). The AND subgoals are denoted on the graph by horizontal arcs connecting the lines leading to them.

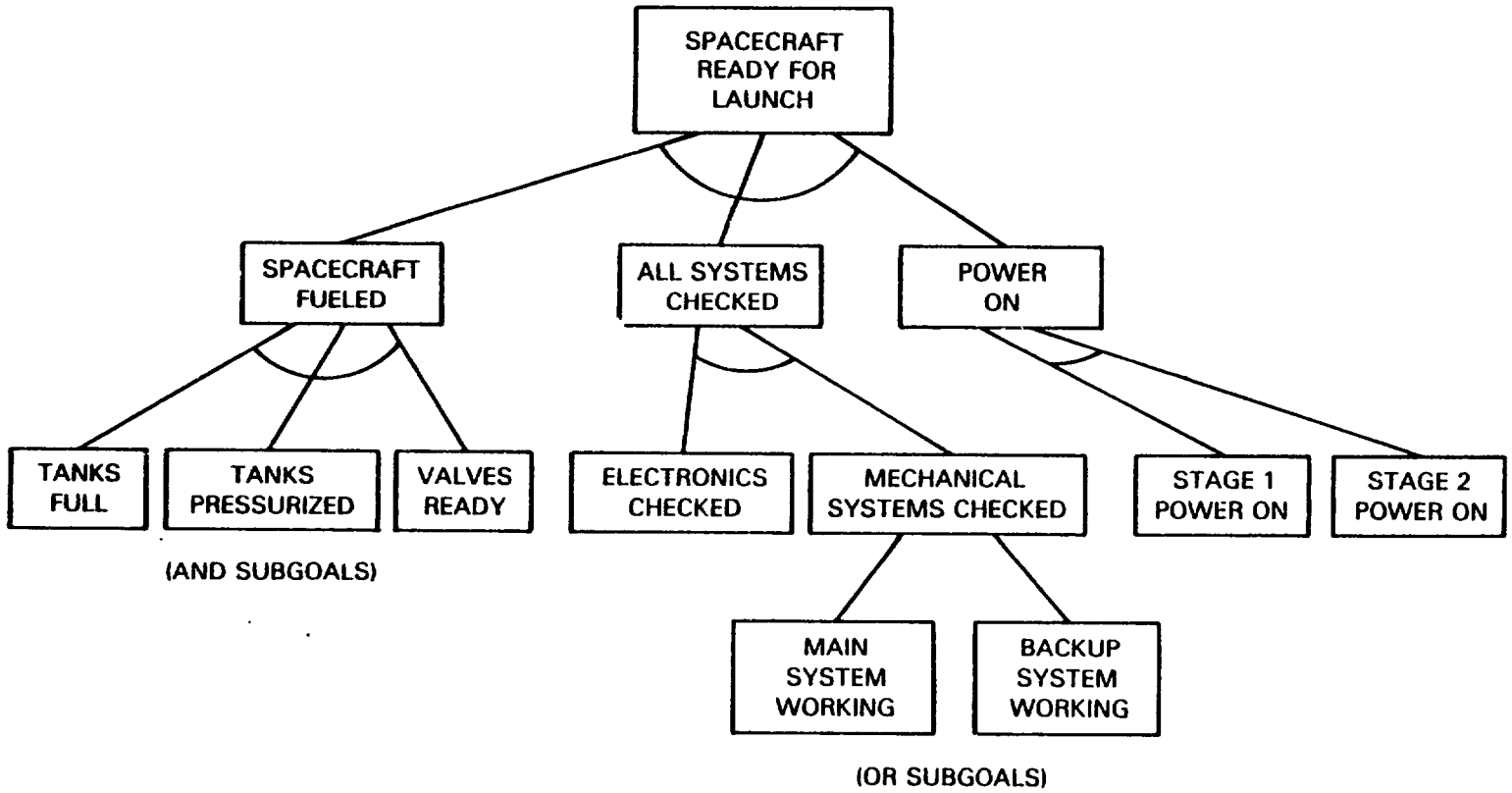


Figure II-6 Simplified AND/OR Graph for Readying a Spacecraft for Launch

Problem reduction often runs into difficulties without specific problem knowledge, as there is otherwise no good reason to attack one interacting conjunct before another. Lack of such knowledge may lead to an extensive search for a sequence of actions that tries to achieve subgoals in an unachievable order.

F. Heuristic State-Space Search

Blind search does not make any use of knowledge about the problem to guide the search. In complex problems, such searches often fail, being overwhelmed by the combinatorial explosion of possible paths. If on the average there are n possible operators that can be applied to a node, then the size of the search space tends to grow as n^d, where d is the depth to be searched. Heuristic methods have been designed to limit the search space by using information about the nature and structure of the problem domain. Heuristics are rules of thumb, techniques or knowledge that can be used to help guide search. Heuristic search is one of the key contributions of AI to efficient problem solving. It operates by generating and testing intermediate states along a potential solution path.

One straightforward method for choosing paths by this approach is to apply an evaluation function to each node generated, and then pursue those paths that have the least total expected cost. Thus, we can calculate the cost from the root to the particular node that we are examining and, using heuristics, estimate the cost from that node to the goal. Adding the two, produces the total estimated cost along the path, and therefore serves as a guide as to whether to proceed from that node or to continue from another, more promising, node among those thus far examined.

Nilsson (1980) and Barr and Feigenbaum (1981) describe the "A* algorithm," which is guaranteed (under appropriate conditions) to find a solution path of minimal cost if any solution path exists. The A* algorithm uses an evaluation function for the n-th node of:

$$f^*(n) = g^*(n) + h^*(n)$$

where $g^*(n)$ estimates the minimum path cost from the start node of the tree to node n and $h^*(n)$ estimates the minimum cost from node n to the goal. For the A* algorithm to find the minimum cost path, the heuristic estimate of $h^*(n)$ of the cost from node n to the goal, must be non-negative and less than the actual cost h(n). An example $h^*(n)$ for our problem of Figure II-3 (whose paths are represented by the tree in Figure II-4) would be the straight line (airline) distance from node n to the goal.

Though the A* algorithm produces a minimum cost path, it does not usually minimize the search effort, in fact usually producing an exponential running time for the search. If one leans more to minimizing the search effort rather than the solution cost, one would put more emphasis on $h^*(n)$, the estimate of the remaining cost to the goal, rather than on $g^*(n)$ the cost from the start node. Barr and Feigenbaum (1981) describe various approaches to this tradeoff of speed versus solution quality, and indicate that a considerable reduction in running time is possible if the optimal solution requirement is relaxed.

G. Game Tree Search

1. Representation

Most games played by AI computer programs involve two players making alternate moves. A game representation must thus take into account the opponent's possible moves as well as the player's own moves. The usual representation is a game tree, which shares many features with a problem reduction representation. A complete game tree is a representation of all possible plays of such a game.

The root node is the initial state, in which it is the first player's (A's) turn to move. The successors of the root node are the states that A can reach in one move. The successors of these nodes are the states resulting from the other player's (B's) possible replies; etc. At each play, the players must take into account all the opponent's possible responsive moves. This can be represented by an AND/OR tree. Figure II-7 is an example of such a tree from the standpoint of player A, who is to move next. Drawn

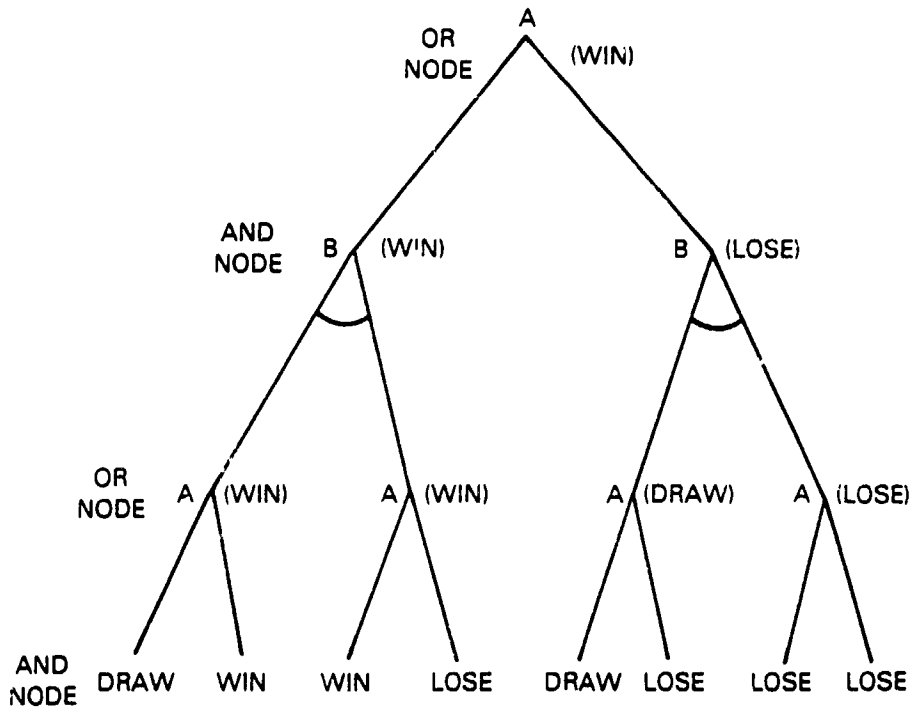


Figure II-7. A Game Tree Drawn from A's Point of View, A's Move

from point A's point of view, A's possible moves under his control are represented by lines leading to AND nodes. These successor nodes are called AND nodes since they control sets of moves all of which A must be able to respond to. A's nodes are called OR nodes, because A can choose any of the moves emanating from these nodes.

2. The Minimax Search Procedure

The minimax procedure is a strategy for playing a two-person game. According to the minimax technique, player A should move to the position of maximum value to him, B responding by choosing a move of minimum value to player A. Given the values of the terminal positions (see Figure II-7), the value (shown in parentheses) of a nonterminal position to player A is computed by backing up from the terminals as follows:

- The value of an OR node is the maximum value of any of its successors.
- The value of an AND node is the minimum value of any of its successors.

3. Searching a Partial Game Tree

For most games, the tree of possibilities is much too large to be generated fully or searched backward for an optimal move. Thus a reasonable portion of the tree is generated starting from the current position, a move is made on the basis of partial knowledge, the opponent reply found, and the procedure recursively repeated from the new position. The minimax procedure thus starts with an estimate of the tip nodes thus far generated, and assigns backed-up values to the ancestors (e.g., values in parentheses in Figure II-7). The value estimates for the tip nodes are generated using a "static evaluation function" based on heuristics.

To reduce the number of nodes that need to be examined, various pruning techniques have been devised, "Alpha-Beta" (see, e.g., Marsland, 1983) being the best known. All these techniques are based on keeping track of backed-up values so that branches that cannot lead to better solutions need not be further explored.

4. Heuristics in Game Tree Search

A "static evaluation function" is one that estimates a board position without looking at any of the positions' successors. The function is usually a linear polynomial whose variables represent various features of the position. For chess, the features of importance include remaining pieces, king safety, center control and pawn structure.

5. Other Considerations

Alternatives to search in choosing moves include opening or end game "book" moves, and recognizing patterns on the board and associating appropriate playing methods with each pattern. The most successful game-playing programs thus far, have made search, rather than knowledge, their main ingredient. Various combinations of more extensive use of specific game knowledge to prune less desirable paths, and increased look-ahead have been utilized in chess in efforts to improve program success.

H. Difference Reduction ("Means-Ends" Analysis)—Another Basic Approach

The difference reduction approach differs from pure search (which usually starts with either the goals or the initial conditions) by instead progressively nibbling away at the problem to reduce the differences between the initial and goal status. Difference reduction was introduced by the General Problem Solver (GPS) Program developed by Newell, Shaw and Simon beginning in 1957 (Ernst and Newell, 1969). This was the first program to separate its general problem-solving method from knowledge specific to the current problem.

Figure II-8 is a simplified flow diagram of the difference reduction approach. The analysis first determines the difference between the initial and goal states and selects the particular operator that would most reduce the difference. If this operator is applicable in the initial state, it is applied and a new current state is created. The difference between this new current state and the goal state is then calculated and the best operator to reduce this difference is selected. The process proceeds until a sequence of operators is determined that transforms the initial state into the goal state.

If at any point, the operator chosen cannot be applied in the current state, a new intermediate goal state is established that is the precondition for the chosen operator to be applied. The difference between the current state and this new intermediate goal state is then used as before. If the new intermediate goal cannot be achieved, a new operator is chosen to reduce the initial difference and the problem proceeds recursively until a solution is achieved.

The difference reduction approach assumes that the differences between a current state and a desired state can be defined and the operators can be classified according to the kinds of differences they can reduce. If the initial and goal states differ by a small number of features and operators are available for individually manipulating each feature, then difference reduction works. However, there is no inherent way in this approach to generate the ideas necessary to plan complex solutions to difficult problems.

I. More Efficient Tactics for Problem Solving

For more efficient problem solving than the methods described above, it is necessary to devise techniques to guide the search by making better use of initial knowledge about the problem or of the information that can be discovered or learned about the problem as the problem solver proceeds through the search. These techniques are reviewed in the Non-Deductive Problem Solving Approaches Section of Chapter III of Part A of this volume.

J. Future Directions for Research

Sacerdoti (1979) suggests the following lines of research as being especially important for the future.

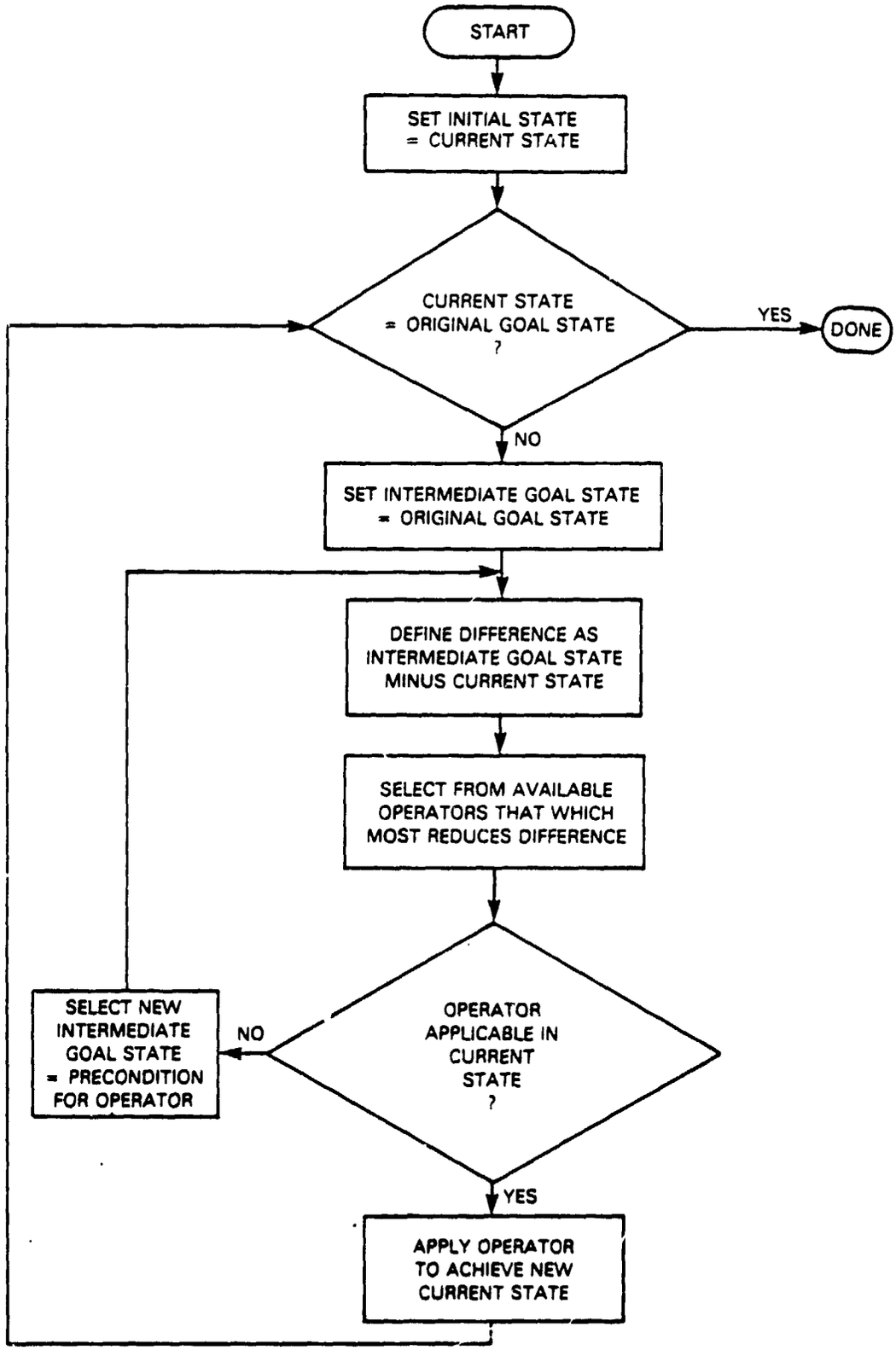


Figure II-8. Simplified Flow Diagram of the Difference Reduction Approach

1. Integrating a Significant Number of Tactics

This approach, if successful, could result in a very powerful problem solver, particularly where hierarchical planning provides the framework and all the other techniques can be applied at each level of planning in the hierarchy. (Some of the more complex expert systems have taken such an approach.)

2. Flexible Control Structure

In the real world, unexpected events occur frequently, so it is often more appropriate to only rough out a plan, creating only its critical components in detail. Then, when the plan is executed, detailed plans can be developed using real world feedback.

3. Planning for Parallel Execution

It appears that problem solvers that distribute plan generation and execution tasks will be one of the major waves of the future. Pseudo-reduction tactics create plans that are partially ordered with respect to time. Therefore they share with hierarchical plan structures the virtue of being particularly amenable to planning in parallel by multiple problem solvers and to execution in parallel by multiple effectors.

4. Partial Goal Fulfillment

Thus far, problem solvers have been designed to fully satisfy their goals. However, in the real world, full goal satisfaction during execution is often impossible. Thus it becomes important to be able to prioritize goals and plan for their partial satisfaction. (This is further explored in Part B, Chapter VI on Problem Solving and Planning.)

5. Feedback of Lessons Learned from Plan Execution to Plan Generation

Lessons learned from plan execution can be extremely valuable for future plan generation. Therefore focussing on integrated systems for plan generation, execution and repair may be one of the best approaches to advancing the state of the art. Particularly, developing catalogs of successful plan generation tactics can be valuable in dealing with complex, interactive environments which have been beyond our capability thus far.

K. Current Research

Table II-1 presents an indication of current research activities in search-oriented automatic problem solving and planning techniques. A more detailed view of current research in this area is provided by the Special Issue on Search and Heuristics of the *Artificial Intelligence Journal* (Pearl, 1983).

L. Current State of the Art

Real, complex problems tend to have the characteristic that their search space tends to expand exponentially with the number of parameters involved. This "NP Complete" type of problem still is out of bounds for searches that do not have powerful heuristics to guide them. Chess has been one indicator of the state of the art in problem solving emphasizing search (though computer capability has been an equally important factor). Berliner (1981) reports that 1981 chess programs (emphasizing look-ahead) had reached an expertise of 2300 points compared with roughly 2500 points for the best human experts.

Current problem solvers emphasizing search have thus far succeeded only in solving elementary or toy problems, or very well structured problems such as games. Thus, the AI community's emphasis has shifted toward expert systems (Duda, 1981) as problem solvers, where the emphasis is on knowledge rather than search.* In addition, there are trends toward distributed problem solving systems and toward interactive problem solving systems where humans make the major decisions and the computer program offers choices and works out the details.

** Even in chess, there is beginning to be an emphasis on knowledge as evidenced by the CHUNKER Program (Campbell and Berliner, 1983) where the incorporation of knowledge about patterns of chess positions drastically reduces search requirements in appropriate situations.*

TABLE II-1. Examples of Current and Recent Research in Search-Oriented Problem Solving Techniques

Technique	Program	Institute	Researcher	Funding
Heuristic Search	Heuristic Search Theory	UCLA	J. Pearl	NSF
	Efficiency in Backtracking	Indiana Univ.	C.A. Brown P.W. Purdom	NSF
	Blockhead	U of CA Irvine	D. Kibler P. Morris	NOSC
	B Algorithm for Heuristic Search	Hungarian Academy of Science	L. Mero	
	DELTA MIN for Backtracking	CMU	J. Carbonell	ONR
Constraint Satisfaction and Relaxation Algorithms	Invasion Procedure	UBC	R. Seidel	
Multiple Agent Planning Systems	Distributed AI	SRI	K. Konolige N.J. Nilsson	ONR
Multi-Agent Planning	Synchronization of Multi-Agent Plans	Stanford U.	J.S. Rosenchein	ONR
Parallel Search	Algorithm for Parallel Processing in Heuristic Search	MIT	W.A. Kornfeld	
	Parallel Search Efficiency	Several Canadian Researchers		
Minimax Game Tree Search	Algorithm for Games with Chance Events	Duke	B.W. Ballard	AFOSR
Game Playing Algorithms	SSS* Minimax Algorithm	U. of MD	L. Kanal V. Kumar	NSF
	Brute Force Intelligence SNAC Optimum Search	CMU	H.J. Berliner	ARPA
Branch and Bound	General Formulation	U. of MD	D. Nau et al.	NSF
Analytical Evaluation of Search Methods	Unified Approach	Indiana U.	P.W. Purdom & C.A. Brown	NSF
Generalization of Bayes Rule	A Distributed Hierarchical Approach	UCLA	J. Pearl	NSF
Coordinated Multiple Blackboard Global Data Bases	Unifying Data-Directed and Goal-Directed Control in a Multi-Level Cooperating Knowledge Source Problem Solver	U. of MA	D.D. Corkill V. Lesser	NSF
Distributed Problem Solving	Meta-Level Control	U. of MA	V. Lesser et al.	NSF ARPA
Other	KAMP Planning System Using Procedural Network	SRI	D.E. Appelt	ARPA

M. Forecast

It is expected that within the next five years, the increased speed and capability of computers and the ability to do parallel searches could have as much effect on search performance as new search methods. However, as search usually grows exponentially with depth, heuristics to restrict the paths to be searched will also be of continuing importance. It is also expected that techniques to combine shallow and deep reasoning (e.g., non-monitonic reasoning, causality, first principles, theorem proving) will be major contributors to limiting and guiding search.

Schank (1983) states that ". . .search is one of the key AI problems. However,. . . the approaches to search have been inadequate. Searching massive amounts of information requires not efficient algorithms but representations that obviate the need for these algorithms." (Knowledge representation is the subject of the next chapter.)

REFERENCES

- Arden, B.W. (Ed.), *What Can Be Automated*, Cambridge: MIT Press, 1980.
- Barr, A. and Feigenbaum, E.A., *The Handbook of Artificial Intelligence*, Vol. 1, Los Altos, CA: William Kaufman, Inc., 1981.
- Berliner, H.J., "An Examination of Brute Force Intelligence," *Proceedings of the Seventh International Conference on Artificial Intelligence*, August 1981, pp. 581-587.
- Campbell, M. and Berliner, H., "A Chess Program That Chunks," *Proc. of the Nat. Conf. on AI: AAAI-83*, Wash, D.C., August 22-26, 1983, pp. 49-53.
- Duda, R.O., "Knowledge-Based Expert Systems Come of Age," *Byte*, Vol. 6, No. 9, Sept. 1981, pp. 238-281.
- Duda, R.O., "State of Technology in Artificial Intelligence," *Research Directions in Software Technology*, Wegner, P. (Ed.), Cambridge: MIT Press, 1979, pp. 729-749.
- Ernst, G. and Newell, A., *GPS: A Case Study in Generality and Problem Solving*, New York: Academic Press, 1969.
- Graham, N., *Artificial Intelligence*, Blue Ridge Summit, PA: Tab Books, 1979.
- Hayes-Roth, F., "AI: The New Wave—A Technical Tutorial for R&D Management," AIAA-81-0827, Santa Monica, CA: Rand Corp., 1981.
- Marsland, T. A., "Relative Efficiency of Alpha-Beta Implementations," *Proc. of the Eighth Inter. Joint Conf. on AI: IJCAI-83*, Karlsruhe, W. Germany, 8-12 Aug 1983, Los Altos, CA: W. Kaufmann, 1983, pp. 763-766.
- Nilsson, N.J., *Principles of Artificial Intelligence*, Palo Alto: Tioga Publishing Co., 1980.
- Pearl, J. (Ed.), Special Issue on Search and Heuristics, *Artificial Intelligence*, Vol. 21, Nos. 1, 2, March 1983.
- Sacerdoti, E.D., "Problem Solving Tactics," *AI Magazine*, Vol. 2, No. 1.
- Schank, R.C., "The Current State of AI: One Man's Opinion," *The AI Magazine*, Vol. IV, No. 1, Winter/Spring 1983.

III. KNOWLEDGE REPRESENTATION

A. Introduction

Artificial Intelligence views knowledge as the key to high-performance intelligent systems. Thus the representation and management of knowledge is a central topic in AI today.

Newell (1981) defines knowledge as the information used by intelligent agents (human or machine) to make rational decisions.* Further, Newell states that "knowledge is not just a collection of symbolic expressions plus some static organization; it requires both processes and data structures." Thus knowledge representation consists of a system for providing access to a body of knowledge—a data structure for representation in memory and a means (the computational process) for accessing that knowledge.

Structure and access are thus intertwined, with ideally a representation being chosen that simplifies access to the knowledge for the particular task at hand. Thus, a variety of knowledge representations exist, arising from the search for the most useful representation for the class of problems for which they have been devised.

Myopolis (1981, p. 5) states that, "the basic problem of knowledge representation is the development of a sufficiently precise notation for representing knowledge." To this must be added the requirement for efficiency and rapid access.

For the purpose of knowledge representation (KR), Myopolis treats a knowledge base as a model of a world/enterprise/slice of reality. The Heuristic Programming Project (1980, pp. 5-6) indicates that the knowledge base (KB) of AI programs contains both factual knowledge of the task at hand and heuristic knowledge representing the tacit judgmental knowledge comprising domain expertise, and often meta-knowledge of how to solve problems efficiently and effectively.

B. Purpose

The purpose of knowledge representation is to organize the information required into a form such that the AI program can readily access it for making decisions, planning, recognizing objects and situations, analyzing scenes, drawing conclusions, and other cognitive functions. Thus knowledge representation is especially central to "expert systems," "computational vision," and "natural language understanding."

C. Techniques

Representation schemes** are classically classified into declarative and procedural ones. Declarative refers to representation of facts and assertions, while procedural refers to actions, or what to do. It is virtually impossible to come up with a pure system of either type as ultimately both assertions and what to do with or about them are involved in the data structures and the access mechanism in any knowledge representation.

A further subdivision for declarative (object oriented) schemes includes relational (semantic network) schemes and logical schemes.

The principal KR schemes are briefly discussed in the following paragraphs and summarized in Tables III-1.

---

*More precisely, Newell (1981, p. 20) defines knowledge as "Whatever can be ascribed to an agent, such that its behavior can be computed according to the principle of rationality."

**The discussion of KR techniques given in this section is based primarily on Myopolis (1981), Barr and Feigenbaum (1981, pp. 141-222) and Graham (1979, pp. 188-208).

1. Logical Representation Schemes

The principal method for representing a knowledge base logically is to employ first order predicate logic. In this approach, a knowledge base (KB) can be viewed as a collection of logical formulas which provides a partial description of the world. Modifications to the KB results from additions or deletions of logical formulas.

Examples of logical representations are:

IN(SHUTTLE, ORBIT) = The shuttle is in orbit.

$\forall$(x). EXTRA-TERRESTRIAL BODY(x)$\rightarrow$POSSESSES (x, NO KNOWN LIFE)

= For all x, where x is an extra-terrestrial body, x possesses no known life. Or more simply, all extra-terrestrial bodies have no known life.

Logical representations are easy to understand and have available sets of inference rules needed to operate upon them. Table III-1a summarizes the various aspects of logical KR's.

2. Semantic Networks

A semantic network is an approach to describing the properties and relations of objects, events, concepts, situations or actions by a directed graph consisting of nodes and labelled edges (arcs connecting nodes). Because of their naturalness, semantic networks are very popular in AI.

In a semantic net, the program can start at a node of interest and follow arcs to related nodes, and in turn follow arcs to still more distant nodes. This approach is very natural—being reminiscent of human thinking. However, the multiplicity of pathways, as we go further from the starting node, makes it easy to get lost in the maze, unless a strong organizing or guiding principle is used (such as the "beam-search" approach employed by the HARPY speech-understanding system).

The various aspects of semantic networks are summarized in Table III-1b.

3. Procedural Representations and Production Systems

In procedural representations, knowledge about the world is contained in procedures—small programs that know how to do specific things (how to proceed in well specified situations). Classification of procedural representation approaches are based on the choice of activation mechanisms for the procedures, and the forms used for the control structures.

The two common approaches consist of procedures representing major chunks of knowledge—subroutines (see Table III-1c)—and more modular procedures, such as used in PLANNER (Hewlitt, 1972) and the currently popular production rules. The common activation mechanism for procedures is matching of the preconditions needed for the procedure to be invoked. In PLANNER, this is referred to as "pattern directed procedure invocation." The main difference between PLANNER and the more recent "production rules" is that PLANNER's elemental procedures (called theorems) can communicate directly with each other, while the communication between production rules is only by modification of the pattern in the Global Data Base (GDB) for the individual production rules to observe.

Production rules (PR) are characterized by a format of the type:

Pattern, Action
If, Then
Antecedent, Consequent
Situation, Procedure

A PR system consists of a knowledge base (KB) of rules, a global data base (GDB) which represents the system status, and a rule interpreter (control structure) for choosing the rules to execute. In a simple production rule system, the rules are tried in order and executed if they match the pattern in the GDB.

TABLE III-1a. Knowledge Representation Schemes
First Order Predicate Logic

Nature of KB	Representation & Applications	Advantages	Disadvantages
A collection of logical formulas which provides a partial description of the world. Modifications of KB occur by the introduction/deletion of logical formulas.	Using quantifiers and logical connectives, one can make a statement about objects, properties, situations and relationships. *Example:* All people have heads = V(x). PERSON (x)→HAS (x, HEAD)	Simplicity of notation — descriptions are readily understandable. Natural, precise, flexible, modular. Availability of well-understood formal semantics. Each fact need be represented only once. Availability of inference rules for proof procedures. Derivation of new facts from old can be mechanized using automated versions of theorem-proving techniques.	Difficulty in representing procedural and heuristic knowledge. Lack of organizational principles makes a large KB difficult to manage. Weak manipulation (proof) procedures. When the number of facts becomes large, there is a combinatorial explosion in the possibilities of which rules to apply at each step of the proof.

TABLE III-1b. Knowledge Representation Schemes
Semantic Nets

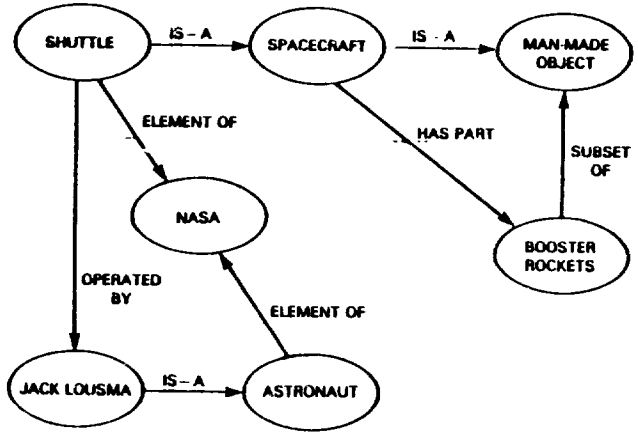
Nature of KB	Representation	Advantages	Disadvantages
A world described by a collection of nodes representing objects, object properties, concepts, events, situations or actions, and arcs or links (binary associations or labeled edges) in a directed labeled graph. Modifications to this network KB occur through the insertion/deletion of objects and the manipulation of associations.	A directed labelled graph that naturally links objects in a relational way: *Example:*  *Uses:* • Domains where much of the reasoning is based on a very complicated taxonomy. • Domains where it is necessary to represent properties-of or relations-between objects, events, situations, or actions. (Very popular in AI)	Important associations can be made explicitly and succinctly. Relevant facts can be inferred from the nodes to which they are directly linked without a search thru a large data base. Can use ISA and SUBSET links to establish a property inheritance hierarchy in the net. Easy to make deductions about inheritance hierarchies. Can establish states and actions in terms of a small no. of primitive concepts. (Trend towards network schemes with a fixed no. of primitive association types, which have well-defined semantics, and are descriptively adequate)	Inferences drawn by manipulation of the net are not assuredly valid. No standard terminology or conventions about meaning. The interpretation (semantics) depends solely upon the program that manipulates the network. A strong organizing principle is needed to guide search thru the maze. Difficult to represent Boolean relationships other than disjunction.

TABLE III-1c. Knowledge Representation Schemes
Procedural — Subroutines

Nature of KB	Representation	Advantages	Disadvantages
The KB is a collection of procedures. Modification of KB by addition/subtraction, or modification of subroutines or their access conditions.	Knowledge about the world is contained in procedures — small programs that know how to do specific things, how to proceed in well-defined situations. *Control information* is inherent in the way one states the facts. *Example:* A subroutine is called when a certain situation occurs.	• Facility for representing heuristic knowledge. • Ability to perform extended-logical inferences like default reasoning. • May have advantages for modeling. • Sometimes much easier to keep track of changes and side effects as the procedure that performs the action can update the data base immediately. • Can do reasoning thru simulation.	• Difficult to verify or change — as knowledge is implicit in procedures. • Control information gets in the way, limiting alternative approaches.

However, in more complex systems, such as used in expert systems, a very complex control structure (see, e.g., Gevarter, 1982) may be used to decide which group of PR's to examine, and which to execute from the PR's (in the group) that match patterns in the GDB. In general, these control structures work in a repetitive cycle of the form:

1. Find the conflict set (the set of rules which match some data in the GDB).
2. Choose a rule from among the conflict set.
3. Execute the rule, modifying the GDB.

Because of their modular representation of knowledge and their easy expansion and modifiability, PR's are now probably the most popular AI knowledge representation, being chosen for most expert systems.

Table III-1d summarizes the central aspects associated with production rule systems.

4. Analogical or Direct Representations

In many instances it is appropriate to use natural representations such as an array of brightness values for an image, or a further reduced sketch map of the scene delineations in a computer vision system. This "homomorphism" (structural similarity) is evident in the use of maps, geometric models, etc. These direct representations are analogous to some properties of the situation being represented.

These natural representations are useful in computational vision, spatial planning, geometric reasoning and navigation. One even notices analogical aspects in musical notation where the rise and fall of the musical frequency is apparent in the representation of the notes in the score.

This form of representation has the advantages of being easy to understand, simple to update, and often allows important properties to be directly observed, so that they don't have to be inferred. A direct or analogous representation can usually be more exhaustive and specific, making for more efficient problem solving. It also can facilitate search and working with constraints. However, this form of representation is clumsy for some tasks, particularly when generalization is needed.

Table III-1e summarizes the attributes of direct representation.

5. Property Lists

One approach to describing the state of the world is to associate with each object a property list; that is a list of all those properties of the object pertinent to the state description. The state and therefore the object properties can be updated when a situation is changed.

Table III-1f briefly indicates the attributes of such a representation.

6. Frames and Scripts

Humans are able to handle with relative ease a large variety of circumstances in everyday life because to a great extent our days are filled with a series of stereotyped situations such as going to work, eating, shopping, etc.

Minsky (1975) conceived of "frames," which are complex data structures for representing stereotyped situations. A frame has slots for objects and relations that would be appropriate to the situation. Attached to each frame is information such as:

— how to use the frame
— what to do if something unexpected happens
— default values for slots.

Frames can also include procedural as well as declarative information. Frames facilitate expectation-driven processing—reasoning based on seeking confirmation of expectations by filling in the slots. Frames organize knowledge in a way that directs attention and facilitates recall and inference.

TABLE III-1d. Knowledge Representation Schemes
Procedural — Production Systems (PS)

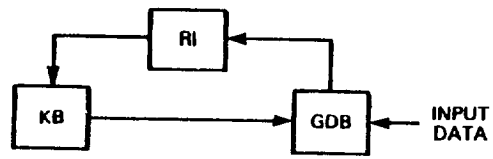
Nature of KB	Representation	Advantages	Disadvantages
The KB is a collection of loosely coupled production rules (PRs) representing knowledge about the world. These PR's may be organized into sets, called knowledge sources (KS), for particular uses. In adddition, a global data base (GDB) is used in PS's to represent the system status, and a control structure (rule interpreter, RI) is used to select the rules to activate and execute.	A PS can be represented as:  The PR's take the form: Situation → Action Antecedent → Consequent If → Then *PR Example:* If the Shuttle doors fail to automatically close when actuated, and the fault cannot be discovered. Then disengage motors and close doors manually. *Uses:* • Knowledge about what to do in a specific situation (Expert Systems) • Domains where: —the current task is a sequence of transitions from one state to another —knowledge is diffuse (e.g., medicine) —knowledge can be easily separated from the manner in which it is used. —processing can be represented by a set of independent actions	Modular: Provides a high granularity of information (facts and rules). Information can be easily added, removed or updated. Naturalness. Facility for representing heuristic knowledge, particularly domain-specific information that might permit more directed deduction processes. Easy to keep track of changes due to actions. Useful as a mechanism for controlling the interaction between statements of declarative and procedural knowledge.	Hard to maintain modularity (non-overlap) between rules in large systems. Constraining interactions between rules can lead to inefficiencies. Inefficiency of program execution. Hard to follow the flow of control in problem-solving. Poor separation of knowledge and control when: (1) chunks of knowledge are large (2) dealing with basically sequential information. PR's are pure associations, and therefore don't provide substantive explanations. Knowledge about structure is represented in PRs only implicitly. PR's not too well suited to often-needed structural and causal models. Problems with consistency and completeness.

TABLE III-1e. Knowledge Representation Schemes
Direct or Analogical Representations

Nature of KB	Representation	Advantages	Disadvantages
The knowledge base is a collection of natural representations such as maps, images and geometric models. Modifications to KB occur by updating, adding to or removing representations.	The approach is to use a representation that is analogous with respect to some properties of the situation being represented. The representation is used by direct observation of the information desired such as adjacency, linkages, size, shape, constraints, etc. *Examples* Maps Line Drawings Geometric models.	• Natural representation — readily understandable. • Easy to update. • Important properties often directly observable, so don't have to be inferred. • Usually more exhaustive and specific — making for more efficient problem solving. • Facilitates search and working with constraints.	• Clumsy for some tasks. • May be inappropriate when generalization is needed.

TABLE III-1f. Knowledge Representation Schemes
Property Lists

Nature of KB	Representation	Advantages	Disadvantages
• Lists of properties of objects • KB modified by addition and/or subtraction of properties as appropriate to the state of the system.	State of world is described by obects in the world and lists of their pertinent properties. A property is an attribute-value pair. *Example:* List for a Robot: Attribute — Value in State 1. — Value in State 2. Location — Room A — Room B Holds — Box — Ball On — Floor — Chair Erect — True — False	• All appropriate properties for object grouped into a list. Program need not search a huge DB to find properties of object. • Lists are a natural structure in the LISP programming language.	• Difficult to carry out inferential operations.

An example of a frame is:

Airplane Frame:

Specialization of: Aerospace vehicle

Types:
  range: (fighter, transport, trainer, bomber, light plane, observation)

Manufacturer:
  range: (McDonnell-Douglas, Boeing....)

Empty Weight:
  range: (500 lbs to 250,000 lbs)

Gross Weight:
  range: (500 lbs to 500,000 lbs)
  if needed: (1.6 × empty weight)

Name:
  if needed: (Choose name satisfying type and manufacturer)

Max Cruising Range:
  if needed: (Look up in table cruising range appropriate to type and gross weight)

Number of Cockpit Crew:
  range: (1 to 3)
  default: 2

Scripts are frame-like structures designed for representing stereotyped sequences of events such as eating at a restaurant or a newspaper report of an apartment fire.

Table III-1g summarizes the central aspects of frame representations.

7. Semantic Primitives:

For any knowledge representation scheme it is necessary to define an associated vocabulary. For semantic nets, there has been a real attempt to reduce the relations to a minimum number of terms (semantic primitives) that are non-overlapping. A similar effort has emerged for natural language understanding.

A natural language is an attempt to describe all of the world's aspects important to humans. Unfortunately, as these languages evolved naturally, rather than being scientifically ordained, a great deal of ambiguity has entered the languages, such that the meaning is often dependent on context and background knowledge. Several attempts have been made to describe all of the world's aspects in terms of primitives that are unique, unambiguous representations into which natural language statements can be converted for later translation into another language or for other cognitive actions.

Wilks (1977) has proposed a system designed to be used for language translation. His system is centered around a dictionary for distinguishing among the various senses of the words that can appear in the input text. Definitions in the dictionary are defined in terms of some 80 semantic primitives grouped into the following five classes:

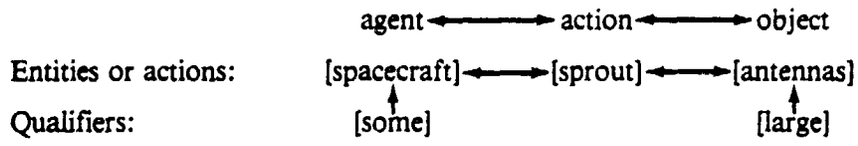
Class	**Example Primitives**
Entities	man, stuff, part
Actions	cause, be, flow
Cases	to, in
Qualifiers	good, much
Type Indicator	now, kind

TABLE III-1g. Knowledge Representation Schemes
Frames and Scripts

Nature of KB	Representation	Advantages	Disadvantages
An organized collection of frames.	• A frame is a complex data structure representing a stereotyped situation. • A frame has slots for: — objects — relations • Attached to each frame is information such as: — how to use frame — what to do if something unexpected happens — default values of slots — procedures if needed • Scripts are frame-like structures for representing stereotyped sequences of events *Uses* Stereotyped events and situations such as eating out, identifying standardized manufactured devices, scene analysis, writing stereotyped newspaper reports, analyzing stories, natural language understanding, standardized plans, etc.	• Facilitates expectation-driven processing. • Can be designed to determine its own applicability in a given situation, or to suggest other frames if appropriate. • Organizes knowledge in a way that directs attention and facilitates recall and inference.	• Many real situations depart considerably from prototypes. • New situations not easily accommodated.

The completed representation of a text is in terms of semantic formulas constructed from the primitives. For example:

"Some spacecraft sprout large antennas".



where the terms in square brackets would be replaced by semantic formulas representing their appropriate sense.

Wilks has also incorporated inference rules and other structures to assist in clarification and organization of the resulting text representation.

Schank (see, e.g., Schank and Riesbeck, 1981) has developed a "conceptual dependency" theory as an attempt to provide a representation of all actions in terms of a small number of primitives. Schank's goal is broader than language translation, the representation being task-independent so as to be applicable to inferring, paraphrasing and answering questions as well.

Schank's primitives are intended to be unambiguous and unique. The system relies on 11 primitive physical, instrumental and mental ACTs (propel, grasp, P trans, A trans, speak, attend, etc.), plus several other categories, or concept types.

Detailed rules are provided in conceptual dependency for combining the elements into representations or meaning. There are two basic kinds of combinations or conceptualizations. One involves an actor doing a primitive ACT; the other involves an object and a description of its state. Attached to each primitive act is a set of inferences that could be associated with it.

More recently, Schank has added clarifying elements in terms of goals, scripts, plans, themes and social acts, designed to provide additional meaning, purpose and context to the representations.

An example of a representation in conceptual dependency is:

Armstrong flew to the moon.

Actor:	Armstrong
Action:	flew
Direction to:	the moon
From:	Unknown

The use of semantic primitives allows propositions to be stored in canonical (standardized) form, with resultant computational advantages for many uses.

D. Representation Languages

A number of programming languages have been designed to facilitate knowledge representation. Table III-2 lists some of the more popular ones.

It will be observed that usually one form of knowledge representation (such as production rules or frames) is chosen as central to the language, though some (such as UNITS) provide for multiple representations.

TABLE III-2. Programming Tools Facilitating Knowledge Representation

Tools	Organization	Nature
OPS 5	CMU	A programming language, built on top of LISP, designed to facilitate the use of production rules.
ROSIE	Rand	A general rule-based programming language that can be used to develop large knowledge bases. Translates near-English into INTERLISP.
UNITS	Stanford U.	A knowledge representation language and interactive knowledge acquisition system. The language provides both for "frame" structures and production rules.
KRL	Xerox PARC	Knowledge representation language developed to explore frame-based processing
SAM	Yale	A system of computer programs to analyze scripts.
FRL	MIT	A frame representation language that provides a hierarchical knowledge base format consisting of frames whose slots carry comments, default values, constraints, and procedures that are activated when the value of the slot is needed.
KL-ONE	BBN	A uniform language for representation of natural language conceptual information, based on the idea of structured inheritance networks. Networks use epistemological primitives as links.
NETL	CMU	A comprehensive, domain independent, knowledge-base system. It uses a parallel intersection technique for searching rapidly through large bodies of knowledge.
DAWN	DEC	A general programming and system description language with automated help procedures.
OWL	MIT	A semantic network knowledge representation language for use in natural language question answering and for building expert systems.
FRAIL	Brown U.	A KR language that combines predicate calculus with frame representation for use in natural language understanding.

E. State of the Art

Though production rules have emerged as the dominant KR for expert systems, and semantic networks for image understanding, KR is still in a state of flux with many researchers, various representations, and no clear general understanding of which representations are most appropriate for which problems. As a result, KR research is one of the most active areas in AI today.

F. Issues

SIGART's (1980) "Special Issue on Knowledge Representation," indicates that there are many areas of concern (pp. 114-115). Virtually every aspect of KR still is an issue. A few of these issues are:

1. First Order Predicate Logic (FOPL) as a Standard of Representation.

   Many researchers such as Kowalski (p. 44) feel that FOPL is the only language suitable for KR—whether declarative or procedural. Correspondingly Kowalski maintains there is only one intelligent way to process information—and that is by applying deductive inference methods.

   Others such as Sloman (p. 48) declare that there is "No such thing as an ideal representational formalism...No one formula is equally adequate for all things for all purposes...No doubt all

knowledge representation can be embedded in predicate calculus, but this may be of little practical importance."

Zadeh (p. 48) observes that most human knowledge is imprecise in nature. Therefore two valued logic and associated representation techniques are not appropriate—fuzzy logic being necessary.

2. How to provide everyday context and common-sense know-how in representations? Drefus (p. 42) observes that the background context continually varies, while rule behavior tends to assume "everything else being equal."
3. Need to consider representations in a broader sense, such as holograms which can be used to process information, but is not a data structure.
4. Hobbs (pp. 43-44) declares that, "Standard practice in the representation of knowledge is the scandal of AI. . . Ninety percent of what is done in the representation of knowledge is reinvention, most frequently in predicate calculus." There is a multitude of items for similar aspects. "The consequence is a jungle of incomparable results."

   In this regard Newell (1981) observes in regard to the SIGART KR survey that, "The main result was overwhelming diversity—a veritable jungle of opinions. There is no consensus on any question of substance."
5. Doyle (p. 41) declares that there is a need to consider intention, action, purposive communication and the processes of problem solving in KR. Also needed are systems which are self-referent (both to descriptions and parts and to belief systems). Better KR's for learning processes and belief revision also need to be developed.
6. Need to clarify which KR's are best for which purposes.
7. How do we find the most appropriate representation for given problems?
8. Problem of selecting the appropriate level of abstraction for a problem—scope and grain size (Davis, 1982).
9. KR's that facilitate knowledge acquisition.
10. Designing KB's to facilitate updating—modularity.
11. Need for multiple representations for different aspects (or at different stages of problem solving) of the same problem.
12. Problem of incompleteness inherent in all KR's.
13. Understandability—transparency.
14. Lack of a theory of KR.
15. How to represent knowledge so as to enable AI programs to behave as if they knew something about the problems they solve.
16. How best to choose a representation to provide the greatest efficiency in deductive reasoning (Moore, 1982).

G. Some Research Needs:

1. Standardization of nomenclature and techniques
2. Methods of matching representations to problems
3. Methods to handle imprecise knowledge

4. Methods to evaluate efficiency of representation
5. Need to be able to conveniently represent intentions, beliefs, etc. in representations.
6. Methods to provide self-knowledge in representations.
7. Methods for quantification—the ability to specify properties of arbitrarily defined sets.
8. Representation methods for people's beliefs.
9. Representations of processes that consist of sequenced actions over time.
10. Representations for complex and amorphous shapes.
11. Techniques for indexing into a large data base of models.

H. Who Is Doing It

Review of SIGART's (1980) "Special Issue on Knowledge Representation" indicates that the following are the principal organizations involved in KR research.

1. Universities

Stanford University
University of Hamburg (West Germany)
CMU
Simon Fraser University (Canada)
University of Paris
University of Pittsburgh
MIT
Yale
University of Toronto (Canada)
University of Maryland
SUNY, Buffalo
University of Ottawa (Canada)
Rutgers University
University of Amsterdam (Netherlands)
University of Essex (England)
University of California (Berkeley)
N. Dakota State University

2. Other

IBM
DEC
SRI
BBN

I. Future Directions

The knowledge representation field has begun to exhibit some structure—rule-based systems predominating in Expert Systems, but network representations also being important. For image understanding systems, direct representations (such as line sketches) are common, with network representations being widely employed.

In the future, we will probably see increased standardization of terminology, standardized primitives, and the use of multiple types of representations in a single problem. We can also expect increased emergence of self-reflective systems that can reason about their own structure and knowledge.

Also emerging will be knowledge representation systems that are appropriate for learning, generalization and abstraction—currently difficult subjects.

KR languages are on the increase, which should help in constructing knowledge-based systems and encourage standardization of representations.

Within the next five years, we can expect a clearer understanding of which representations are appropriate for which problems.

We can also expect KB's to vastly increase in size with KR techniques being developed to ease the addition of knowledge to them and the retrieval of knowledge from them.

REFERENCES

• Barr, A. and Feigenbaum, E. A., *The Handbook of Artificial Intelligence*, Vol. I, Los Altos, CA: W. Kaufman, 1981.

• Brachman, R. J. and Smith, B. C. (Eds.), "Special issue on Knowledge Representation," *SIGART Newsletter*, No. 70, Feb. 1980.

• Davis, R., "Expert Systems: Where are We? And Where Do We Go From Here?" *AI Magazine*, Vol. 3, No. 2, Spring 1982, pp. 3-22.

• Gevarter, W., *An Overview of Expert Systems*, National Bureau of Standards, NBSIR 82-2505, May 1982 (Revised October 1982).

• Hewitt, C., "Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theories and Manipulating Models in a Robot," Ph.D. thesis, Dept. of Mathematics, MIT, 1972.

• McCalla, G. and Cercone, N., (Eds.), Special Issue on Knowledge Representation, *Computer*, Vol. 16, No. 10, Oct. 1983.

• Minsky, M., "A Framework for Representing Knowledge," in P. Winston (ed.), *The Psychology of Computer Vision*, New York: McGraw-Hill, 1975, pp. 211-277.

• Moore, R. C., "The Role of Logic in Knowledge Representation and Common Sense Reasoning," *Proc. of the Nat. Conf. on A.I.*, CMU/U. of Pittsburgh, Aug. 1982, pp. 428-433.

• Myopoulis, J., "An Overview of Knowledge Representation," *SIGART Newsletter*, No. 74, Jan. 1981, pp. 5-12.

• Newell, A., "The Knowledge Level," Dept. of Computer Science, CMU, Rept. No. CMU-CS-81-131, July 1981.

• Schank, R. C. and Riesbeck, C. K., *Inside Computer Understanding*, Hillsdale, N.J.: Lawrence Erlbaum, 1981.

• Wilks, Y., "Knowledge Structures and Language Boundaries, *IJCAI 5*, 1977, pp. 151-157.

IV. COMPUTATIONAL LOGIC

A. Introduction

It is frequently necessary to develop computer programs to deduce facts that are not explicitly represented but that are implied by other represented facts. An intelligent robot may have to use logical facts about its environment, e.g., to deduce when a goal state has been reached or how to reach the goal state in the first place. A data base query system may have to deduce desired information from other information in the data base.

Computational logic has been developed to address such problems. In addition, the associated predicate calculus expressions have proven to be a powerful means for knowledge representation for AI programs. Computational logic is thus an important AI area and is briefly reviewed in this chapter.

Raphael (1976, pp. 110-111) states:

> A typical task posed for a logical system is the following. Given some logical sentences representing premises, and a sentence called a theorem, which represents some assertion whose truth we wish to determine, demonstrate whether the theorem is guaranteed to be true provided only that the premises are true. If such a demonstration can be obtained, it is called a proof of the theorem from the given premises, and we say that the premises imply the theorem.
>
> There are two approaches to attempting to construct proofs. One, called the semantic approach, depends heavily upon the meanings of the symbols in the [logical statements]. In a sense, when we use a semantic proof, we reason primarily by considering all the possible interpretations of the logical statement to be proved. In the other approach, called syntactic, we totally ignore the meanings of the symbols; instead, we use formal symbol-manipulation rules of the logical system to construct new [logical statements] out of old ones. The syntactic approach is frequently easier to use, especially for a computer, because one can apply rules in a mechanical way without having to think about what they mean.
>
> A logical system consists of both a specification for the structure of the [logical statements] of the system, and a set of rules, called the rules of inference of the system, for constructing proofs. Many different logical systems have been invented; in fact, each mathematician is free to invent his own as he sees fit.

Traditional computational logic—a computational approach to logical reasoning—is divided into two principal parts, the simpler *"propositional logic"* and the more complex *"predicate logic."*

B. Propositional Logic

In logic a "proposition" is simply a statement that can be true or false. Rules used to deduce the truth (T) or falsehood (F) of new propositions from known propositions are referred to as "argument forms." The interesting and useful things we can do with propositions result from joining propositions together with connectives such as OR, AND, NOT, and IMPLIES to make new propositions. The symbols for these connectives are given in Figure IV-1.

The simplest argument form is the "conjunction," which utilizes the connective AND. It states that if proposition p is true and proposition q is true, then the conjunction "p AND q" is true. In symbolic form we have

p	(premise)
q	(premise)
$p \wedge q$	(conclusion).

which simply states that for a conjunction, the conclusion is true if the premises are true.

Connective	Symbol	Meaning
And	$\wedge$ or $\cap$	both
Or	$\vee$ or $\cup$	either or both
Not	$\neg$ or $\sim$	the opposite
Implies	$\supset$ or $\rightarrow$	If the term on the left is true, then the term on the right will also be true.
Equivalent	$\equiv$	has the same truth value

Figure IV-1. Typical Mathematical Logic Symbols.

Deduction means obtaining solutions to problems using some systematic reasoning procedures to reach conclusions from stated premises. (In mathematical logic, deductive procedures are sometimes referred to as "formal inference.")

One simple form of deduction can be represented as a mathematical form of argument called "Modus Ponens" (MP):

p	(premise)
p IMPLIES q	(premise)
q	(conclusion)

An example of MP is:

I'm feeling very sick	(premise)
When I'm feeling very sick, I must call the doctor	(premise)
I must call the doctor	(conclusion)

The conclusion is usually stated as a theorem to be proved.

The method of truth tables is the best-known method for proving theorems in propositional calculus. This is a semantic method, in which all the possible combinations of interpretations for the propositional variables are examined.

Graham (1979, pp. 165-168) enlarges on this:

> Suppose we are given some expression involving propositions and logical connectives. Suppose further that we know whether each individual proposition in the expression is true or false. We would like to be able to calculate whether or not the proposition represented by the entire expression is true or false.

We can do this in two steps. First we assign each proposition in the expression a truth value of either T or F. True propositions get the value T and false ones get the value F.

Second, we treat the connectives AND, OR, NOT and IMPLIES as operators operating on T and F, just like the +, −, ×, and / in an algebraic expression operate on numbers. In other words, we do "logical arithmetic" to calculate the truth value of the entire expression.

The proposition

p OR q

is true if p is true, if q is true, or if both are true. This gives us the following truth table for OR:

p	q	p OR q
T	T	T
T	F	T
F	T	T
F	F	F

Another argument form, the "implication relation" is defined such that if p IMPLIES q then when p is true, q will be true, and nothing more. The implication relation does not say that p and q have any cause-and-effect relationship to one another. When p is false, nothing whatever is asserted about q. Therefore, the only way in which p implies q can be false is if p is true and q is false. The resultant truth table is:

p	q	p IMPLIES q
T	T	T
T	F	F
F	T	T
F	F	T

A large number of argument forms are available in traditional logic. All these forms can be easily verified using simple truth tables.

Raphael (1976, pp. 113-114) observes:

> The task of constructing a truth table can certainly be programmed for a computer, and the truth-table method will work to prove or disprove any theorem of propositional calculus. However, this method is not entirely satisfactory, because it can be extremely inefficient. If n different propositional variables occur in the premises and the theorem, then a table with 2^n rows must be filled out; a problem with ten variables requires more than a thousand lines.

Wang (1960) at Harvard University developed a syntactic method that is about as efficient as any general method for propositional calculus can be. It produces exactly the same results as truth tables, usually requires much less computational effort, and is easy to program.

C. Predicate Logic

Propositional logic is limited in that it deals only with the T or F of complete statements. Predicate logic remedies this situation by allowing one to deal with assertions about items in statements, and allows the use of variables and functions of variables.

Propositions make assertions about items (individuals). The "predicate" is the part of the proposition that makes an assertion about the individuals. A proposition is conveniently written as:

$$\text{Predicate}\ \overbrace{(\text{Individual, Individual,}\ldots)}^{\text{arguments of the predicate}}$$

(e.g.)

"The box is on the table." (proposition) is denoted as:

ON(BOX, TABLE)

The predicate, together with its arguments, is a proposition. Its value is T or F, and any of the operations of propositional logic may be applied to it.

A variable stands for any individual. Variables allow us to make statements that would not be possible in propositional logic. For a proposition containing variables to be true, it must be true for any individual names that are substituted for the variables.

Using variables, if we want to write the English sentence:

"If x is by y and z is on x, then z is also by y."

as a predicate logic expression, it would take the form:

BY(x,y) AND ON(z,x) IMPLIES BY(z,y).

Substituting the name of a particular individual for a variable is known as "instantiation." It is called instantiation because the individual is a particular "instance" of the variable. We can assert that something exists by making up a name for it (e.g., a, b — individual constants) and use that name in our expression. For example, to state that "a" is a box, we write

BOX(a).

We can instantiate our previous expression (for the case of a window, table and ball) as:

BY(TABLE, WINDOW) AND ON(BALL, TABLE)
IMPLIES BY(BALL, WINDOW)

which translates as: if the table is by the window and the ball is on the table, then the ball is also by the window.

Sometimes the individual whose existence we wish to assert will depend on some other individual; then we can use functions (f, g, h) to do this. For example:

The mother of a = f(a).

More generally, if we want to assert that every person has a mother, we can write:

PERSON(x) IMPLIES MOTHER(f(x),x)

which can be read as:

If x is a person, then there exists an f(x) that is x's mother.

D. Resolution

Resolution has been the primary technique used in modern computational logic programs. Resolution is a syntactic method of deduction which replaces all the many argument forms of traditional logic. Resolution is a simple concept but to discuss it, a few additional definitions will be helpful.

Atom: a proposition that cannot be broken down into other propositions (i.e., a proposition that is not formed from other propositions by using connectives).

Literal: an atom (e.g., q) or an atom preceded by NOT (e.g., NOT q)

Clause: a series of literals joined by OR, e.g.: (NOT p) OR q OR (NOT r)
[Duplicate literals in clauses can be eliminated. This process is called factoring.]

Resolution Principle (R): an argument form that applies to clauses:

p OR l OR m OR...	(premise)
(NOT p) OR n OR o OR...	(premise)
l OR m OR n OR o OR...	(conclusion)

(If the premises are T, then by resolution (the cancellation of contradicting literals between clauses) the conclusion is T.)

Empty Clause: (□) indicates a contradiction:

p
NOT p
□ (by R)

Equivalence: Two propositions are equivalent if they have the same truth value.

Identity: States that two propositions are equivalent (proved by using a truth table):

e.g., NOT (NOT p) = p (identity)

After first putting the original premises and the conclusion into clause form using standard identities, we are ready to prove the truth of a conclusion from a set of premises using resolution. Start by negating the desired conclusion (the theorem to be proved). Then derive new clauses using unification* followed by factoring and resolution, working toward deriving the empty clause. If the empty clause is derived, then (as a result of this proof by contradiction) the desired conclusion follows from the original premises. If we stop before the empty clause is derived, then either the conclusion does not follow from the premises or we gave up too soon.

Graham (1979, pp. 186-187) observed:

> Resolution is complete in the sense that if the conclusion does follow from the premises, then repeated unification, resolution, and factoring will eventually derive the empty clause.
>
> Resolution can be more easily programmed on a computer, and the resulting program is more efficient than was the case with any previous computational-logic programs.
>
> [At present, resolution programs] cannot handle such complex tasks as proving deep mathematical theorems, verifying computer programs, or aiding a robot to cope with the complexities of the real world (as opposed to a limited laboratory world). For these tasks the resolution program uses up the available time or memory before deriving the empty clause.
>
> The trouble, as is usual in AI, is a combinatorial explosion. Unification, resolution, and factoring derive many clauses that are not relevant to deriving the empty clause. The program wastes its time following lines of reasoning that come to a dead end.
>
> Because of these difficulties some people have given up the possibility that computational logic can handle complex theorem-proving tasks. Others seek restrictions on the way resolution and factoring are done that will reduce the number of clauses generated without destroying completeness. Still others (including the author) feel that the answers lie in using powerful heuristic and planning techniques to guide the resolution program to its goal of deriving the empty clause.

*Unification is the name for the procedure for carrying out instantiations. In unification we attempt to find substitutions for variables that will make two atoms identical.

E. Computational Logic Today

Computational logic has evolved into several distinguishable subareas: theorem proving, logic programming, non-monotonic logics, and multi-valued and fuzzy logics.

1. Theorem Proving

This branch of computational logic is an outgrowth of resolution theorem proving with additional techniques and modifications added to attempt to restrain combinatorial explosions. With restrictions on resolution clause generation, theorem proving approaches can be made sufficiently efficient to be used in practical problems. An outstanding example of this is the AURA (AUtomated Reasoning Assistant) theorem proving system (Wos, 1983) that has successfully been applied to real applications in mathematics, formal logic, program verification, logic circuit design, chemical synthesis, database inquiry and robotics.

Three techniques (used in the AURA system) that have had a major impact on making theorem provers practical are:

(1) Demodulation: Employing rewrite rules to simplify or canonicalize the expressions to achieve a normalized form.

(2) Subsumption: A technique that recognizes and discards many equivalent or weaker rules or facts than those that have already been generated.

(3) Strategy Rules: Ordering strategies that direct the system as to what to do next.

These three powerful techniques in AURA are domain independent (though the strategy rules have provision for weighting so that the user can assign priorities to concepts).

Other strategies have been important for further reducing the expressions that are generated or retained during the proof process. One class is restriction strategies which provide guidance as to which operations can be skipped. For example, there is the "set of support" strategy that discourages looking at facts that don't have support (e.g., general information used alone, unsupported by other facts).

There are indications that there remain many important domain-independent inference rules yet to be discovered. Examples of research in theorem proving are given by Table IV-1.

2. Logic Programming

It was realized in the early 1970s that logic representations could also function in a procedural mode by using the technique of unification to search for instantiations that would satisfy stated goals. This has led to the PROLOG programming language (see, e.g., Chapter III, Vol IA).

As the manner in which the representations are written and the order (e.g., top to bottom, and left to right) chosen for the execution of the logic statements can have an important influence on the efficiency and effectiveness of executing the program, such representational and ordering choices can be thought of as a form of programming, hence the name logic programming. PROLOG, and logic programming in general, has become very popular in the last few years.

An indication of current research in Logic Programming is given by Table IV-2.

3. Non-Monotonic Logic

One of the popular issues in AI problem solving has been concerned with how to handle lines of reasoning and conclusions that may have to be retracted when new information is received. For example, it is usually reasonable to conclude that if a creature is a bird, then it can fly. However, if it is later learned that the bird is a penguin or is dead, the conclusion must be reconsidered. Recent research efforts on how to handle such situations are indicated in Table IV-3. Etherington and Reiter's (1983)

TABLE IV-1. Examples of Research in Theorem Proving

Technique	Program	Institute	Researcher
Resolution-based automated reasoning program	AURA (AUtomated Reasoning Assistant)	Argonne Nat. Laboratory Argonne, IL	L. Wos R. Overbeek
Set of procedures for tailoring an automated reasoning machine to given specifications	LMA (Logic Machine Architecture, from which a portable reasoning program, ITP, was built)	NW Univ. Evanston, IL	E. Lusk W. McCune R. Overbeek
A strategy for semantic paramodulation of Horn Sets	NUTS (NW U. Theorem-proving System)	NW Univ. Evanston, IL	W. McCune L. Henschen
Special purpose program for program verification		Univ. of TX Austin, TX	R. Boyer J. Moore
Use of examples in automated theorem proving to help guide proof discovery and to determine instantiation of set variables		Univ. of TX Austin, TX	W. Bledsoe
Many-Sorted Calculus based on resolution and paramodulation		Univ. of Karlsruhe W. Germany	C. Walther
A very fast algorithm for unit refutation for the MKR-Procedure	TERMINATOR	Univ. of Karlsruhe W. Germany	G. Antoniou H. Ohlbach
Superposition-oriented theorem prover		L.I.T.P. Paris, France	L. Fribourg
Associative-commutative operators for a refutationally-complete theorem prover		U. of IL Urbana, IL	N. Dershowitz N. Josephson D. Plaisted
		SUNY Stonybrook, N. Y.	J. Hsiang
Procedures for building non-equational theories into a resolution theorem-proving program		SRI Inter.	M. Stickel
Improving the expressiveness of Many Sorted Logic		U. of Warwick Coventry, England	A. Cohen

Sources: IJCAI-83, AAAI-3

work on providing a formal semantics for networks of inheritance hierarchies with exceptions appears particularly promising.

4. Multi-Valued and Fuzzy Logics

Conventional logics deal with the truth or falsity of statements. However, this binary approach is often inadequate for situations in which degrees of certainty are involved as, for example, in medical diagnosis. Thus, work in multi-valued and fuzzy logics has been undertaken to attempt to address this problem. Table IV-4 provides an indication of research in these areas. Several approaches for handling degrees of certainty have already been successfully incorporated into expert systems such as MYCIN and PROSPECTOR.

F. Future Directions

Moore (1982) argues that a number of important features of commonsense reasoning involving incomplete knowledge of a problem situation can be implemented only within a logical framework. Thus logic-based systems will continue to be an important element of AI.

TABLE IV-2. Examples of Research in Logic Programming

Technique	Program	Institute	Researcher
PROLOG Development	PROLOG	Faculte des Sciences de Luminy Marseille, France	A. Colmerauer
Development of a PROLOG/LISP type of programming language	QUTE	U. of Tokyo Japan	M. Sato T. Sakurai
Inclusion of assertions about equality in PROLOG		M.I.T. Cambridge, MA	W. Kornfeld
Use of PROLOG for semantic code analysis of assembler listings		IBM Poughkeepsie, N.Y.	W. Wilson C. John
Extension of PROLOG to increase range of explanation capability	PROLOG/EX1	IBM San Jose, CA	A. Walker
Augmentation of PROLOG to include uncertainties		Weizmann Inst. Rehovot, Israel	E. Shapiro
Addition of algorythmic control structures to PROLOG	LOGAL	U. of Nottingham Med. School U.K.	D. Dodson A. Rector
A method for building libraries of routines and data in PROLOG		Bell Labs. Murray Hill, N.J.	A. Feuer
Integrating PROLOG into the POPLOG environment	POPLOG	Univ. of Sussex Brighton, U.K.	C. Mellish S. Hardy
An interpreter for logic programs which executes goals in parallel		U. of CA Irvine, CA	J. Conery D. Kibler
An experimental tool for parallel execution of distributed AI problem solvers based on logic programming	PRISM	U. of MD College Park, MD	S. Kasif M. Kohli J. Minker
A simple unification algorithm for infinite trees for structure sharing implementarions of logic programming languages		Inst. for New Generation Computer Technology (ICOT) Tokyo, Japan	K. Mukai
A program to debug logic programs		Uppsala Univ. Sweden	A. Edman S. Tarnlund

Source: IJCAI-83

The advent of powerful resolution-based theorem proving systems (such as AURA) — utilizing both domain-independent and domain-dependent inference rules to constrain combinatorial explosions — has resulted in opening up practical applications for such systems. However, much research remains to be done to discover more effective strategies, to devise methods for linking rules together to take larger reasoning steps, to explore parallel processing approaches, to build user-friendly interfaces, and to develop more rapid and improved knowledge representation techniques.

TABLE IV-3. Examples of Research in Non-Monotonic Reasoning

Activity	Institute	Researcher
Data Dependencies on Equalities	Yale U.	D. McDermott
Inheritance Hierarchies with Exceptions using Default Logic	Univ. of BC Canada	D. Etherington R. Reiter
Default Reasoning as Likelihood Reasoning	Univ. of TX Austin	E. Rich
Default Reasoning using Monotonic Logic	Tulane U.	J. Nutter
Reason Maintenance	Carnegie Mellon U. (CMU) Pittsburgh, PA	J. Doyle
Semantic Considerations in non-monotonic Logic	SRI International	R. Moore

Source: AAAI-83, IJCAI-83

TABLE IV-4. Examples of Research in Multi-Valued and Fuzzy Logics, and Plausible Reasoning Techniques

Activity	Institute	Researcher
Approximate Reasoning Techniques	Universite Paul Sabatier Toulouse, France	H. Prade
Consistency and Plausible Reasoning	Rand Santa Monica, CA	J. Quinlan
Propagation of Uncertainty	Advanced Information & Decision Systems Mt. View, CA	R. Tong D. Shapiro J. Dean B. McCune
Heuristic Reasoning about Uncertainty	Stanford U. Stanford, CA	P. Cohen M. Grinberg
"Evidence Space" for Dealing with Uncertain Reasoning	Tech. Univ. of Berlin Fed. Rep. of Germany	C. Rollinger
Use of Baysian Statistics in Common Sense Reasoning	Brown U. Providence, R.I.	E. Charniak
Fuzzy Logic	U. of CA Berkeley, CA	L. Zadeh
A Method for Computing Generalized Bayesian Probability Values for Expert Systems	SRI International	P. Cheeseman

Source: IJCAI-83

The advent of portable theorem proving systems opens up the opportunity for much increased experimentation, which should be instrumental in rapidly advancing the field. Wos (1983) predicts that as a result, automated reasoning systems with the capability for being used in a wide variety of real applications will be commonplace within five years.

As expert systems technology pushes forward toward employing causality and structure, in addition to empirical association rules, deeper levels of reasoning will be required. It is anticipated that advanced theorem proving systems will play an important role in this arena.

PROLOG, the rapidly proliferating language for logic programming, has been earmarked for the Japanese Fifth Generation Computer project. The powerful inference rules (such as the set of support strategy) used in advanced theorem provers are now being considered for use with PROLOG. These, coupled with domain-specific control strategies and making provisions for taking advantage of many of the features of LISP (as in LOGLISP) may well make a hybridized PROLOG the dominant AI language within the next decade.

Examination of Tables IV, and the associated textual comments, indicate that the basic reasoning problems of non-monotonic reasoning and reasoning in the presence of uncertainty, are beginning to succumb to some of the recent research. We can thus conclude that computational logic, which earlier appeared doomed by the combinatorics generated by the pure resolution approach, has become revitalized with new representational approaches, inference rules, domain heuristics, and advanced computers and will play an increasingly important role in future AI applications.

Additional material on computational logic from an AI point of view can be found in Boyer and Moore (1979), Kowalski (1979), Nilsson (1980), Clocksin and Mellish (1981), Robinson and Sibert (1981), Cohen and Feigenbaum (1982), Clark and Tarnlund (1982), Rich (1983), and Wos et al. (1984).

REFERENCES

• Blasius, K., et al., "The Markgraf Refutation Procedure," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence,* Vancouver, Aug. 1981, pp. 511-518.

• Boyer, R. and Moore, J., *Computational Logic,* New York: Academic Press, 1979.

• Clark, K. and Tarnlund, S. (Eds.), *Logic Programming,* New York: Academic Press, 1982.

• Clocksin, W. F. and Mellish, C. S., *Programming in PROLOG,* Berlin, Springer-Verlag, 1981.

• Cohen, D. N., *Knowledge Based Theorem Proving and Learning,* Ann Arbor: UMI Press, 1981.

• Cohen, P. R. and Feigenbaum, E. A., *The Handbook of Artificial Intelligence,* Vol. 3, Los Altos, CA: W. Kaufmann, 1982.

• Graham, N., *Artificial Intelligence,* Blue Ridge Summit, PA: Tab Books, 1979.

• Hayes-Roth, R., "AI: The New Wave — A Technical Tutorial for R&D Management," Santa Monica: Rand Corp., 1981 (AIAA-81-0827).

• Kowalski, R., *Logic for Problem Solving,* New York: North Holland, 1979.

• Moore, R. C., "The Role of Logic in Knowledge Representation and Common Sense Reasoning," *Proc. of the Nat. Conf. on A.I.,* Pittsburgh, Aug. 1982, Los Altos, CA: W. Kaufmann, 1982, pp. 428-433.

• Nilsson, N. J., *Principles of Artificial Intelligence,* Palo Alto: Tioga, 1980.

• Raphael, B., *The Thinking Computer,* San Francisco: Freeman and Co., 1976.

• Rich, E., *Artificial Intelligence,* New York: McGraw-Hill, 1983.

• Robinson, J. A. and Sibert, E. E., Logic Programming in Lisp, RADC-TR-80-379, Vol. 1, Syracuse U., School of Computer and Information Science, Syracuse, NY, Jan. 1981.

• Wang, H., "Toward Mechanical Mathematics," *IBM J. Research and Development,* Vol. 4, 1960, pp. 2-22.

• Wos, L., "Automated Reasoning: Real Uses and Potential Uses," *IJCAI-83,* pp. 867-876.

• Wos, L., Overbeek, R., Lusk, E., and Boyle, J., *Automated Reasoning: Introduction and Applications,* Englewood Cliffs, N.J.: Prentice Hall, 1984.

• *IJCAI-83: Proc. of the Eighth Inter. Joint Conf. on AI,* 8-12 Aug. 1983, Karlsruhe, W. Germany, Los Altos, CA: W. Kaufmann, 1983.

• *AAAI-83: Proc. of the Nat. Conf. on AI,* Aug. 22-26, 1983, Wash., D.C., Los Altos, CA: W. Kaufmann, 1983.