

Stanford Heuristic Programming Project
Memo HPP-80-3

11
March 1980

Computer Science Department
Report No. STAN-CS-80-793

12
LEVEL #

12
84

AD A089073

6
Representation of Knowledge.

by

10
Avron Barr and James Davidson

9
Technical rept.

a section of the

Handbook of Artificial Intelligence

edited by

Avron Barr and Edward A. Feigenbaum

15
MDA 903-77-C-0322,
✓ ARPA Order-3423

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

DTIC
SELECTE
SEP 10 1980

A



DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

BDC FILE COPY

80 9 10 033
094 L20

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM	
1. REPORT NUMBER STAN-CS-80-793 (HPP-80-3)	2. GOVT ACCESSION NO. AD-A089 073	3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle) Representation of Knowledge		5. TYPE OF REPORT & PERIOD COVERED technical, March 1980	
7. AUTHOR(s) Avron Barr and James Davidson		6. PERFORMING ORG. REPORT NUMBER STAN-CS-80-793 (HPP-80-3) ✓	
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Computer Science Stanford University Stanford, California 94305 USA		8. CONTRACT OR GRANT NUMBER(s) ARPA MDA 903-77-C-0322 ✓ NIH RR-00785-06	
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency Information Processing Techniques Office 1400 Wilson Avenue, Arlington, Virginia 22209		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS	
14. MONITORING AGENCY NAME & ADDRESS (if diff. from Controlling Office) Mr. Philip Surra, Resident Representative Office of Naval Research, Durand 165 Stanford University		12. REPORT DATE March 1980	13. NO. OF PAGES 76
16. DISTRIBUTION STATEMENT (of this report) Approved for public release; distribution unlimited.		15. SECURITY CLASS. (of this report) Unclassified	
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from report)		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE	
18. SUPPLEMENTARY NOTES			
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)			
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report is the section of the Handbook of Artificial Intelligence about knowledge representation research. The Handbook is a compendium of articles about AI ideas, techniques, and systems intended for non-AI scientists, engineers, and students. The material in this report discusses the problems addressed in knowledge representation research in AI and suggests some ways of comparing the various representation schemes. Additional articles describe the AI representation techniques: logic, procedural representations, semantic nets, production systems, direct (analogical) representations, semantic primitives, and frames.			

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Accession For	
NTIS Avail.	<input checked="" type="checkbox"/>
DOC Tab	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or
A	-es.etal

Representation of Knowledge

Avron Barr and James Davidson

a section of the

Handbook of Artificial Intelligence

edited by

Avron Barr and Edward A. Feigenbaum

This research was supported by both the Defense Advanced Research Projects Agency (ARPA Order No. 3423, Contract No. MDA 903-77-C-0322) and the National Institutes of Health (Contract No. NIH RR-00785-06). The views and conclusions of this document should not be interpreted as necessarily representing the official policies, either express or implied, of the Defense Advanced Research Projects Agency, the National Institutes of Health, or the United States Government.

Copyright Notice: The material herein is copyright protected. Permission to quote or reproduce in any form must be obtained from the Editors. Such permission is hereby granted to agencies of the United States Government.

Foreword

Those of us involved in the creation of the Handbook of Artificial Intelligence have attempted to make the concepts, methods, tools, and main results of artificial intelligence research accessible to a broad scientific and engineering audience. Currently, AI work is familiar mainly to its practicing specialists and other interested computer scientists. Yet the field is of growing interdisciplinary interest and practical importance. With this book we are trying to build bridges that are easily crossed by engineers and scientists who want to understand and use AI techniques.

In the Handbook we intend to cover the breadth and depth of AI, presenting general overviews of the scientific issues, as well as detailed discussions of particular techniques and important AI systems. Throughout we have tried to keep in mind the reader who is not a specialist in AI.

As the cost of computation continues to fall, new areas of computer applications become potentially viable. For many of these areas, there do not exist mathematical "cores" to structure calculational use of the computer. Such areas will inevitably be served by symbolic models and symbolic inference techniques. Yet those who understand symbolic computation have been speaking largely to themselves for twenty years. We feel that it is urgent for AI to "go public" in the manner intended by the Handbook.

Several other writers have recognized a need for more widespread knowledge of AI and have attempted to help fill the vacuum. Lay reviews, like Margaret Boden's *Artificial Intelligence and Natural Man* and Pamela McCorduck's *Machines Who Think*, have tried to explain what is important and interesting about AI, and how research in AI progresses through our programs. In addition, there are a few textbooks that attempt to present a more detailed view of selected areas of AI, for the serious student of computer science. But no textbook can hope to describe all of the sub-areas, to present brief explanations of the important ideas and techniques, and to review the forty or fifty most important AI systems.

The Handbook contains several different types of articles. Key AI ideas and techniques are described in core articles (e.g., basic concepts in heuristic search, semantic nets). Important individual AI programs (e.g., SHRDLU, MYCIN) are described in separate articles that indicate, among other things, the designer's goal, the techniques employed, and the reasons why the program is important. Overview articles discuss the problems and approaches in each major area. The overview articles should be particularly useful to those who seek a summary of the underlying issues that motivate AI research.

Eventually the Handbook will contain approximately two hundred articles. We hope that the appearance of this material will stimulate interaction and cooperation with other AI research sites. We look forward to being advised of errors of omission and commission. For a field as fast moving as AI, it is important that its practitioners alert us to important developments, so that future editions will reflect this new material. We intend that the Handbook of Artificial Intelligence be a living and changing reference work.

The articles in this edition of the Handbook were written primarily by graduate students in AI at Stanford University, with assistance from graduate students and AI professionals at other institutions. We wish particularly to acknowledge the help from those at Rutgers University, SRI International, Xerox Palo Alto Research Center, MIT, and the RAND Corporation.

This report, which contains the chapter of the Handbook on knowledge representation research, has been edited by Avron Barr and James Davidson. Douglas Appelt, James Bennett, Robert Filman, and Anne Gardner also contributed original material. Special thanks to Mark Stefik for his careful review of an earlier draft of this chapter.

Avron Barr
Edward Feigenbaum

Stanford University
March, 1980

Handbook of Artificial Intelligence

Topic Outline

Volumes I and II

Introduction

The Handbook of Artificial Intelligence
Overview of AI Research
History of AI
An Introduction to the AI Literature

Search

Overview
Problem Representation
Search Methods for State Spaces, AND/OR Graphs, and Game Trees
Six Important Search Programs

Representation of Knowledge

Issues and Problems in Representation Theory
Survey of Representation Techniques
Seven Important Representation Schemes

AI Programming Languages

Historical Overview of AI Programming Languages
Comparison of Data Structures and Control Mechanisms in AI Languages
LISP

Natural Language Understanding

Overview - History and Issues
Machine Translation
Grammars
Parsing Techniques
Text Generation Systems
The Early NL Systems
Six Important Natural Language Processing Systems

Speech Understanding Systems

Overview - History and Design Issues
Seven Major Speech Understanding Projects

Applications-oriented AI Research -- Part 1

Overview

TEIRESIAS - Issues in Expert Systems Design
Research on AI Applications in Mathematics (MACSYMA and AM)
Miscellaneous Applications Research

Applications-oriented AI Research -- Part 2: Medicine

Overview of Medical Applications Research
Six Important Medical Systems

Applications-oriented AI Research -- Part 3: Chemistry

Overview of Applications in Chemistry
Applications in Chemical Analysis
The DENDRAL Programs
CRYSLIS
Applications in Organic Synthesis

Applications-oriented AI Research -- Part 4: Education

Historical Overview of AI Research in Educational Applications
Issues in ICAI Systems Design
Seven Important ICAI Systems

Automatic Programming

Overview
Techniques for Program Specification
Approaches to AP
Eight Important AP Systems

The following sections of the Handbook are still in preparation and will appear in the third volume:

Theorem Proving
Vision
Robotics
Information Processing Psychology
Learning and Inductive Inference
Planning and Related Problem-solving Techniques

Representation of Knowledge

Table of Contents

A. Introduction to Knowledge Representation	1
B. Survey of Representation Techniques	8
C. Representation Schemes	15
1. Logic	15
2. Procedural Representations of Knowledge	25
3. Semantic Networks	31
4. Production Systems	39
5. Direct (Analogical) Representations	47
6. Semantic Primitives	53
7. Frames and Scripts	68
References	65
Index	73

A. Introduction to Knowledge Representation

Artificial intelligence research involves building computer systems capable of performing tasks like talking, planning, playing chess, and analyzing molecular structure. When we talk about people who do these things, we always talk about what they have to "know" in order to do them. In other words, we describe someone's ability to behave with intelligence in terms of his or her *knowledge*. Similarly, we say that a computer program *knows* how to play cards, or understand spoken English, or manipulate a robot. We *ascribe* knowledge to programs in the same manner that we ascribe it to each other--based on observing certain behavior; we say that a program knows about objects in its domain, about events that have taken place, or about how to perform specific tasks.

The nature of knowledge and intelligence has been pondered by psychologists, philosophers, linguists, educators, and sociologists for hundreds of years. Since our research methodology involves the design of programs that exhibit intelligent behavior, AI researchers have often taken a rather pragmatic approach to the subject of knowledge, focusing on improving the behavior of their programs. In AI, a *representation of knowledge* is a combination of data structures and interpretive procedures that, if used in the right way in a program, will lead to "knowledgeable" behavior. Work on knowledge representation in AI has involved the design of several classes of data structures for storing information in computer programs, and the development of procedures that allow "intelligent" manipulation of these data structures to make inferences.

Keep in mind that a data structure is not "knowledge," any more than an encyclopedia is knowledge. We can say, metaphorically, that a book is a source of knowledge; but without a reader, the book is just ink on paper. Similarly we often talk of the "list" and "pointer" data structures in an AI database as knowledge per se, when we really mean that they represent facts or rules when used by a certain program to behave in a knowledgeable way. (This point is expanded in article C5).

Techniques and theories about knowledge representation have undergone rapid change and development in the last five years. The articles in this chapter try to give a general review of the different representation schemes that researchers have thought up, what they can do well and what they cannot do. Our understanding of these matters is still incomplete; knowledge representation is the most active area of AI research at the present time.

This introductory article should help guide the reader's understanding of the various formalisms described in the articles in this chapter. After a brief discussion of the kinds of "knowledge" that needs to be represented in AI systems, we introduce some issues that will serve as a vocabulary for talking about and comparing different representation methods--terms like scope, understandability, and modularity. The second article in the chapter is a brief survey of the most important representation formalisms, intended to give an overview of the kinds of systems we are talking about. The remaining articles describe, in more detail, the mechanics of the various representation schemes, their development, and some of the current research problems.

Knowledge

What kinds of knowledge are needed to behave knowledgeably? What things do we know "about"? To approach these questions, consider the following list of types of knowledge that might need to be represented in AI systems:

Objects: Typically, we think of knowledge in terms of "facts about" objects in the world around us. *Birds have wings. Robins are birds. Snow is white.* So, of course, there should be some way to represent objects, classes or categories of objects, and descriptions of objects.

Events: We also know about actions and events in the world. *Bob kissed Mary behind the barn. The sky will fall tomorrow.* In addition to a representation for the events themselves, a representation formalism may need to indicate time course of a sequence of events, and their cause-and-effect relationships.

Performance: A behavior like riding a bicycle involves knowledge beyond that of objects and events, knowledge about *how* to do things, the performance of skills. Like bike-riding, most cognitive behaviors, e.g., composing sentences and proving theorems, involve performance knowledge, and it is often hard to draw the line between performance- and object-knowledge. (Beware: pushing too hard on this point leads right back to the fundamental philosophical issue of what knowledge is!)

Meta-knowledge: We also use knowledge about what we know, called *meta-knowledge*. For example, we often know about the extent of our knowledge about a particular subject, about reliability of certain information, or about the importance or acquisition history of specific facts about the world. Meta-knowledge also includes what we know about our own performance as cognitive processors: our strengths, weaknesses, confusability, levels of expertise in different domains, and feelings of progress during problem solving. For example, Bobrow (1975) describes a robot who is planning a trip; its knowledge that it can read the street signs to find out where it is along the way illustrates meta-knowledge.

The questions of whether these kinds of knowledge are distinguishable or whether there are other varieties of knowledge are interesting psychological issues. For now, however, we will ignore the psychological aspects of the problem of knowledge. In this article we will discuss some of the features of the AI knowledge representation schemes that make it possible, sometimes, for computer programs to exhibit behaviors indicating these four different types of knowledge.

Using Knowledge

The most important consideration in examining and comparing knowledge representation schemes is the eventual *use* of the knowledge. The goals of AI systems can be described in terms of cognitive tasks like recognizing objects, answering questions, and manipulating robotic devices. But the actual use of the knowledge in these programs involves three stages: acquiring more knowledge, retrieving facts relevant to the problem at hand, and reasoning about these facts in search of a solution.

Acquisition. We usually think of *learning* as the accumulation of knowledge, but it involves more than the addition of new facts to our brains. Indeed, *knowledge acquisition* involves relating something new to what we already know in a psychologically complex and still mysterious way. AI systems often *classify* a new data structure before it is added to the database, so that it later can be retrieved when it is relevant. Also, in many kinds of systems, new structures can *interact* with old, sometimes interfering with tasks that had previously been performed properly. Finally, some representation schemes are concerned with acquiring knowledge in a form that is *natural* to humans, who serve as the source of new knowledge (see article Applications.B). If these integrative processes do not occur during acquisition, the system would accumulate new facts or data structures without really improving its knowledgeable behavior.

Retrieval. Determining what knowledge is relevant to a given problem becomes crucial when the system "knows" many different things. Humans are incredibly proficient at this task, and most representation schemes that have been directly concerned with this issue have been based on ideas about human memory (see article C3 on semantic nets, C7 on frames, and the Information Processing Psychology chapter on the use of AI methods in building psychological models). The fundamental ideas about retrieval that have been developed in AI might be termed *linking* and *grouping*: if you know that one data structure is going to entail another in an expected reasoning task, put in an explicit link between the two; and if several data structures are typically going to be used together, group them into a larger structure.

Reasoning. When the system is required to do something that it has not been told explicitly how to do, it must reason, i.e., it must "figure out" what it needs to know from what it already knows. For instance, suppose an information retrieval program "knows" only that *Robins are birds* and that *All birds have wings*. Keep in mind that all that it means for a system to "know" these facts is that there are data structures and procedures which would allow it to answer the questions:

Are Robins birds?	Yes
Do all birds have wings?	Yes

If we then ask it, *Do robins have wings?*, the program must "reason" to answer the query. In problems of any complexity, this ability becomes increasingly important. The system must be able to deduce and verify a multitude of new facts beyond those it has been told explicitly.

For a given knowledge representation scheme, we must ask "What kind of reasoning is possible, easy, natural, etc., in this formalism?" There are many different kinds of reasoning one might imagine:

Formal reasoning involves the syntactic manipulation of data structures to deduce new ones following prespecified *rules of inference*. Mathematical logic is the archetypical formal representation (see article C1).

Procedural reasoning uses simulation to answer questions and solve problems. When we use a program to answer *What is the sum of 3 and 4?*, it uses, or "runs," a procedural model of arithmetic (article C2).

Analogical reasoning seems to be a very natural mode of thought for humans,

but, so far, difficult to accomplish in AI programs. The idea is that when you ask a question *Can robins fly?* the system might reason that "Robins are like sparrows, and I know that sparrows can fly, so robins probably can fly." (See article *Problem Solving.C2* for a review of AI attempts to achieve this kind of reasoning.)

Generalization and Abstraction are also natural reasoning processes for humans that are difficult to pin down well enough to implement in a program. If you know that *Robins have wings*, that *Sparrows have wings*, and that *Bluejays have wings*, eventually you will believe that *All birds have wings*. This capability may be at the core of most human learning, but it has not yet become a useful technique in AI (however, see the *Learning and Information Processing Psychology* chapters for current research.)

Meta-level reasoning is demonstrated by how one answers the question *What is Paul Newman's telephone number?* You might reason that "If I knew Paul Newman's number, I would know that I knew it, because it is a notable fact." This involves using "knowledge about what you know," in particular about the extent of your knowledge and about the importance of certain facts. Recent research in psychology and AI indicates that meta-level reasoning may play a central role in human cognitive processing (Gentner and Collins, 1980; Flavell, 1979); some work on implementing this kind of inference mechanism in AI systems has begun (Davis, 1980; Bobrow and Winograd, 1977a; Brachman, 1978).

Two things need to be said about the uses of knowledge described here. First, they are *interrelated*: when acquiring new knowledge, the system must be concerned with how that knowledge will be retrieved and used later in reasoning. Second, when you get right down to it, *efficacy* is the primary consideration for knowledge-based AI systems. Although there is serious concern among AI researchers about the psychological validity of the various representation schemes, we are not yet in a position to prove that one scheme captures some aspect of human memory better than another. There is no *theory of knowledge representation*. We don't yet know why some schemes are good for certain tasks and not others. But each scheme has been successfully used in a variety of programs that do exhibit intelligent behavior.

We will now discuss some of the characteristics of representation schemes that have been used to describe and compare different formalisms.

Scope and Grain Size

What portion of the external world can be represented in a system? In what detail are objects and events represented? And how much of this detail is actually needed by the reasoning mechanisms? Questions like these, concerning the *scope* and *grain size* of a representation scheme, can help determine the suitability of a given formalism for the solution of a particular problem, but they are not easy to answer.

For one thing, of course, the answers depend totally on the particular application intended. A knowledge representation based on *logic*, for instance, might be an extremely fine grain representation in a mathematical reasoning program, but might result in a coarse simplification for a vision program. Exactly how much detail is needed depends, on the

performance desired (see McCarthy and Hayes, 1969). In general, uniformity of detail across the objects and events seems desirable for a given reasoning task (Bobrow, 1975).

Also important in this discussion is the choice of *semantic primitives*, the basic vocabulary used to describe objects and events in the problem domain. For any given formalism--logic, semantic net, procedures, etc.--the choice of the primitive attributes of the domain that are used to build up facts in the database strongly affects the expressive power of the knowledge representation scheme. (This point is discussed thoroughly in article C6).

If one asks, "Can everything that the system must know be represented in the formalism?", the answer is almost always, "Yes, but some things are more easily represented than others." Getting a feeling for what it means "to be represented more easily"--which involves the representation, the domain, and the reasoning strategies--is, at present, part of the art of doing AI research; there is no formal metric for the appropriateness of a formalism along these lines. Bobrow (1975) refers to the process of *mapping* the objects and events in the world into some internal encoding; then one can ask if the mapping in a given situation is "easy," "natural," "psychologically valid," etc.

Modularity and Understandability

If one thinks of the data structures in a program as "pieces of knowledge," then adding new data structures is like adding knowledge to the system. One characteristic that is often used to compare representation schemes is *modularity*, which refers to the ability to add, modify, or delete individual data structures more-or-less independent of the remainder of the database, i.e., with clearly circumscribed effects on what the system "knows."

In general, humans find modular or "decomposable" systems easier to understand and work with (Simon, 1969). To illustrate the kind of difficulty encountered in non-modular systems, consider the complicated interdependence of procedures in a large computer program, like an operating system. The following situation will be familiar to readers who have helped to write and maintain large programs: A large system is composed of many procedures that "call" each other in a complex way that becomes increasingly hard to follow as the system grows. Often modification of Procedure X, so that it will work properly when called by Procedure A, interferes with the proper functioning of X when it is called by Procedure B. In other words, in order to successfully modify a large system, the programmer must understand the interactions of all of its pieces, which can become an impossibly difficult task.

In general terms, the problem with non-modular systems is that the "meaning" of data structures in the knowledge base depends on the *context* in which the knowledge is being used. (Procedures are the data structures in *procedural representations*, which are notoriously non-modular.) Context dependence, in turn, dramatically affects the *modifiability* of the knowledge base; modification is much easier if the "meaning" of a fact can be known when the fact is entered or removed.

On the other hand, some human knowledge just doesn't seem very modular and is very difficult for people to express as *independent* rules or facts. Winograd (1974) generalizes that in modular systems the facts are easy to recognize but the reasoning process may be

quite opaque, and the opposite is often true in procedural representations. The degree to which the system is *understandable* by humans is important in several phases of its development and performance: design and implementation, acquisition of knowledge from human experts, performance of the task, and interaction with and explanations for the eventual user.

In some representation schemes the data structures (e.g. production rules, logic formulae) seem less inherently intertwined, but the control of the interaction of the various database entries is a very important characteristic of all representation schemes. Winograd (1975) suggests that no system is completely modular--in all systems there is some degree of interaction between the data structures that form the knowledge base--but some formalisms are more inherently modular than others.

Explicit Knowledge and Flexibility

Another issue to keep in mind when examining various representation schemes is what part of the system's knowledge is *explicit*. By this we mean to what knowledge do the programmer and the system have direct, manipulatory access, and what knowledge is "built-in." For example, an operating system has an explicit representation of its "priority queues," but its full knowledge about scheduling jobs (deciding which of several users to serve first) is typically hidden deep in voluminous code. The knowledge is there, of course, since the system behaves in a knowledgeable way, but it is *implicit* in the system's program. If on the other hand, the operating system were designed as a *production system*, for example, which used a list of "priority rules" to decide what kind of jobs to run first, the scheduling knowledge would then be more explicit.

One particular advantage of explicit representation schemes is that, because the facts are in a form that allows a global interpretation, the same fact can be used for multiple purposes. In some large systems this feature has been a significant advantage. For example, in MYCIN (article Medical Applications.C1), the production rules that form the system's knowledge about how to diagnose the possible causes of infectious diseases are used not only by the diagnosis module itself, but also by the routines that *explain* the diagnosis module's reasoning to the consulting physician and that *acquire* new rules from expert physicians (Davis and Buchanan, 1977).

Declarative vs. Procedural Representations

On a closely related subject, the dispute about the relative merits of *declarative* vs. *procedural* knowledge representations is a historically important battle from which much of current representation theory was painfully developed (Winograd, 1975). Many of the issues discussed in this article were identified during the declarative/procedural debate. The declarative systems were typified by resolution-based *theorem provers* (see article C1 on logic and the chapter on Theorem Proving), and the procedural systems by Winograd's PLANNER-based SHRDLU (article Natural Language.F4). The Declarativists talked about the *flexibility* and *economy* of their representation schemes, their *completeness* and the *certainty* of the deductions, and about the *ease of modifiability* of the systems. The Proceduralists stressed the *directness* of the line of inference (using *domain-specific heuristics* to avoid using irrelevant knowledge and following unnatural lines of reasoning) and the *ease of coding* and *understandability* of the reasoning process itself.

Although in retrospect these positions seem somewhat arbitrarily chosen over the space of possible features of representation schemes, the declarative/procedural battle was an important one in AI. It dissolved, rather than resolved, and the result was a much greater respect for the importance of knowledge representation in current AI work.

Final Remarks

This article has not been about representation formalisms *per se*, but rather about the pragmatics of *epistemology*, the study of the nature of knowledge. The intention has been to lay the groundwork for an appreciation of the problems inherent in representing knowledge in AI programs. The discussion may also guide a critical comparison of the representation methods described in the articles to follow.

There are many open questions, indeed serious problems, in knowledge representation research. For example, *quantification*, the ability to specify properties of arbitrarily defined sets, is an area of active theoretical research. Other current problems include how to represent people's beliefs (which may or may not be true), mass nouns, degrees of certainty, time and tense information, and processes that consist of sequenced actions taking place over time.

The article which follows immediately is a quick overview of the knowledge representation schemes used in AI. Articles C1-C7 go into substantial detail about individual representation schemes, discussing their development, their technical features, their use in AI systems, and their shortcomings.

References

The best recent review of knowledge representation research in AI is Winograd (1980b). Earlier excellent discussions include the papers in (Bobrow and Collins, 1975), especially Bobrow (1975) and Winograd (1975). Other recent general discussions of knowledge representation include Boden (1977) and the papers in Findler (1979). Brachman and Smith (1980) report on a survey of knowledge representation researchers, showing the wide diversity of goals and approaches among workers in the field.

B. Survey of Representation Techniques

As stated in the preceding article, AI research deals in *experimental epistemology*; in order to create programs that exhibit intelligent behavior, researchers in AI develop schemes for incorporating "knowledge about the world" in their programs. These *representation schemes* involve routines for manipulating specialized data structures to make intelligent inferences. Although some aspects of each knowledge representation technique are incidental and will seem unmotivated, in its way each scheme touches on concerns central to the study of cognition and intelligence.

This survey article presents sketches of the representation schemes that have been used in AI programs that play chess, converse in English, operate robots, etc. Hopefully, seeing simple examples of the major techniques will help the reader get a clearer idea of what a knowledge representation is. Most of this research assumes that *what* needs to be represented is known, a priori; the AI researcher's job is just figuring out *how* to encode the information in the system's data structures and procedures.

State-Space Search

Perhaps the earliest representation formalism used extensively in AI programs was the *state-space representation*, developed for problem-solving and game-playing programs. The search space is not a representation of knowledge, per se: what it represents is the structure of a problem in terms of the *alternatives* available at each possible state of the problem, for example the alternative moves available on each turn of a game. The basic idea is that from a given state in a problem, all possible next states can be determined using a small set of rules, called *transition operators* (or *legal move generators* in game playing programs). For example, in a chess game, the original state is the board position at the beginning of the game. The legal move generators correspond to the rules for moving each piece. So, all of the next states of the game (i.e., the board configurations after each of White's possible first moves) can be generated by applying the move generators to the original positions of the pieces. Similarly, all of the possible states after Black's first response can be generated.

One rather straightforward way to find the winning move is to try all of the alternative moves, then try all of the opponents responses to these moves, then all of the possible responses to those, until all of the possible continuations of the game have been exhausted and it is clear which was optimal. The problem with this solution is that, for interesting problems like chess, there are far too many possible combinations of moves to try in a reasonable amount of time on a machine of conceivable computational power. This problem, called the *combinatorial explosion*, is an important general difficulty for AI systems in all domains (see the Search chapter).

The solution adopted in AI research is to limit the number of alternatives searched at each stage of the look-ahead process to the *best* possibilities. And in order to determine which alternatives are best, programs must *reason* using large amounts *knowledge* about the world. Whatever domain the systems deal with, chess or organic chemistry or pizza parlor scenarios, the goal of research in knowledge representation is to enable AI programs to behave like they *know* something about the problems they solve.

Logic

The classical approach to representing the knowledge about the world contained in sentences like

All birds have wings

is the predicate calculus, developed by philosophers and mathematicians as a formalization of the process of making inferences from facts. The example about birds' wings would be translated into the mathematical formula

$$\forall x. \text{Bird}(x) \supset \text{HasWings}(x)$$

which reads, *For any object, x, in the world, if x is a bird, then x has wings.* The advantage of using a formal representation is that there is a set of rules associated with the predicate calculus, called the *rules of inference*, by which facts that are known to be true can be used to derive other facts which *must* also be true. Furthermore, the truth of any new statement can be checked, in a well specified manner, against the facts that are already known to be true.

For example, suppose we add another fact to our database

$$\forall x. \text{Robin}(x) \supset \text{Bird}(x)$$

which reads, *For any object, x, in the world, if x is a Robin, then x is a Bird.* Then from these two facts, we can conclude, using the rules of inference, that the following fact *must* be true:

$$\forall x. \text{Robin}(x) \supset \text{HasWings}(x)$$

i.e., that *All robins have wings.* Note that there is a specific rule of inference that allows this deduction based on the superficial structure, or *syntax*, of the first two formulae, independent of whether they dealt with birds or battleships, and that new facts derived through application of the rules of inference are always true so long as the original facts were true.

The most important feature of the predicate calculus and related formal systems are that deductions are "guaranteed correct" to an extent that other representation schemes have not yet reached. The semantic *entailment* of a set of logic statements (i.e., the set of inferences or conclusions that can be drawn from those statements) is completely specified by the rules of inference. Theoretically, the database can be kept logically consistent and all conclusions can be guaranteed correct. Other representation schemes are still striving for such a definition and guarantee of logical consistency.

One reason that logic-based representations have been so popular in AI research is that the derivation of new facts from old can be mechanized. Using automated versions of *theorem proving* techniques, programs have been written to automatically determine the validity of a new statement in a logic database by attempting to *prove* it from the existing statements (see the *Theorem Proving* chapter). Although mechanistic theorem provers of this sort have been used with some success in programs with relatively small databases (Green, 1969), when the number of facts becomes large there is a *combinatorial explosion* in the possibilities of which rules to apply to which facts at each step of the proof. More

knowledge about what facts are relevant to what situations is needed, and, again, incorporating additional knowledge is the goal of continuing work in representation theory.

Procedural Representation

The idea of procedural representation of knowledge first appeared as an attempt to encode some explicit control of the theorem-proving process within a logic-based system. (This refers to research on the PLANNER programming language, article A1 Programming Languages.C2). In a procedural representation, knowledge about the world is contained in procedures, small programs that know how to do specific things, how to proceed in well-specified situations. For instance, in a parser for a natural language understanding system, the knowledge that a *noun phrase* may contain articles, adjectives, and nouns is represented in the program by calls (within the NP procedure) to routines that know how to process articles, nouns, and adjectives.

The underlying knowledge, the permissible grammar for a noun phrase in our example, is not stated explicitly, and thus is not typically extractable in a form that humans can easily understand. The consequent difficulty that humans have in verifying and changing procedural representations is the major flaw of these systems. Nevertheless, all AI systems use a procedural representation at some level of their operation, and general consensus gives a legitimate role for procedural representation in AI programs (Winograd, 1975). The advantages and disadvantages of procedural knowledge representation are discussed fully in article C2. Recent work has emphasized *procedural attachment* in frame-based systems (article C7).

Semantic Nets

The semantic net, developed by Quillian (1968) and others, was invented as an explicitly psychological model of human associative memory. A net consists of *nodes* representing objects, concepts and events, and *links* between the nodes, representing their interrelations. Consider, for example, the simple net:



where BIRD and WINGS are nodes representing sets or concepts, and HAS-PART is the name of the link specifying their relationship. Among the many possible interpretations of this net fragment is the statement

All birds have wings.

As illustrated earlier, statements of this sort also have a natural representation in logic-based representation systems. One key feature of the semantic net representation is that important associations can be made explicitly and succinctly: relevant facts about an object or concept can be inferred from the nodes to which they are directly linked, without a search through a large database.

The ability to point directly to relevant facts is particularly salient with respect to ISA and SUBSET links, which establish an *property inheritance hierarchy* in the net. For example, the net segment



might be interpreted to mean that since robins are birds, and birds have wings, then robins have wings. The interpretation (semantics) of net structures, however, depends solely on the program that manipulates them; there are no conventions about their meaning. Therefore, inferences drawn by manipulation of the net are not assuredly *valid*, in the sense they are assured to be valid in a logic-based representation scheme.

Production Systems

Production systems, developed by Newell and Simon (1972) for their models of human cognition (see *Information Processing Psychology*), are a *modular* knowledge representation scheme that is finding increasing popularity in large AI programs. The basic idea of these systems is that the database consists of rules, called *productions*, in the form of condition/action pairs: "If this condition occurs, then do this action." For example,

IF stoplight is red AND you have stopped THEN right turn OK.

The utility of the formalism comes from the facts that the conditions in which each rule is applicable are made *explicit*, and that, in theory at least, the interactions between rules are minimized (one rule doesn't "call" another).

Production systems have been found useful as a mechanism for controlling the interaction between statements of declarative and procedural knowledge. Because they facilitate human understanding and modification of systems with large amounts of knowledge, productions have been used in several recent large applications systems like DENDRAL, MYCIN, PROSPECTOR, and AM (see *Science and Math Applications*). Current work on production systems has emphasized the control aspects of the formalism and the ability to develop self-modifying (learning) systems.

Special Purpose Representation Techniques

Some of the domains that AI researchers work in seem to suggest natural representations for the knowledge required to solve problems. For example, a visual scene from a robot's camera is often encoded as an array representing a grid over the scene: the values of the elements of the array represent the average brightness over the corresponding area of the scene (see *Vision*). This *direct representation* is useful for some tasks, like finding the boundaries of the objects in the scene, but is clumsy for other tasks, like counting the number of objects. In the latter case a list, each element of which represents one object indicating its location, orientation and size, might be a more useful representation. (See the discussion in Bobrow, 1975).

This example illustrates a very important principle to realize when comparing representation techniques. In some sense, these two (and all other) knowledge representation methods are interchangeable: If we know one representation in enough detail, we could for the most part construct the other one. It is the intended *use* of the knowledge about the scene that recommends one representation scheme over another. In a big AI system, like the *speech understanding* programs, multiple representations of the same information may be used simultaneously for different purposes.

Other special purpose representation schemes of particular interest are those used in the early natural language understanding programs, like SAD-SAM and SIR (see article Natural Language.F1), and the *discrimination net* used in the EPAM program (article Information Processing Psychology.B2).

Semantic Primitives

A general problem in knowledge representation schemes, first confronted in semantic-net-based systems, is that there are generally many possible representations in the net for a single fact or event. For instance, the fact that robins have wings could be represented by the example net discussed above, *All robins are birds*, *All birds have wings*, or more simply by linking the WINGS node directly to the ROBINS:



This inherent ambiguity is a general problem in network construction, and the programs that manipulate the net to make deductions must be able to handle alternative structures. Although this feature might be used to great advantage, allowing redundant storage of information with an eye to future relevance, for example, the representational ambiguity generally causes confusion and expense, since the system doesn't know what exactly to look for when it comes time to retrieve a given fact.

One particular task where the problem of non-specificity of representation was critical was the *paraphrase* task popular in natural language research. Problems encountered in trying to build programs that could rephrase an input sentence, or check whether two sentences have the same "meaning," or translate a sentence from one language into another led researchers like Norman and Rumelhart (1975), Schank and Abelson (1977), and Wilks (1977b) to use canonical internal representations based on *semantic primitives*, fundamental concepts from which all others are built. Thus all representation structures are built from the set of primitives in such a way that two structures that mean the same thing reduce to the same network of primitive nodes. The selection of primitive elements for the expression of knowledge in a given domain is a basic problem in all representation schemes, whether the primitives are represented as nodes in a net, predicates in logic formulae, or slots in a frame.

Frames

The most recently developed AI knowledge representation scheme is the *frame*, still in its early development stage. Researchers have different ideas about what exactly a frame is, but basically, a frame is a data structure that includes declarative and procedural information in predefined internal relations. Thus a generic frame for a dog might have knowledge hooks, or *slots* for facts that are typically known about dogs, like the BREED, OWNER, NAME, and an "attached procedure" for finding out who the owner is if that is not known. In the frame-like language KRL (Bobrow and Winograd, 1977a), a dog-frame might look like this:

Generic DOG Frame

```
Self:  an ANIMAL; a PET
Breed:
Owner:  a PERSON
       (If-Needed: find a PERSON with pet=myself)
Name:  a PROPER NAME (DEFAULT=Rover)
```

DOG-NEXT-DOOR Frame

```
Self:  a DOG
Breed: mutt
Owner:  Jimmy
Name:  Fido
```

The semantics of this example, as well as the ideas being developed in frame-based formalisms are discussed in article C7.

An interesting, much discussed feature of frame-based processing is a frame's ability to determine whether it is applicable in a given situation. The idea is that a likely frame is selected to aid in the process of understanding the current situation (dialogue, scene, problem) and this frame in turn tries to "match" itself to the data it discovers. If it finds that it is not applicable, it could transfer control to a more appropriate frame (Minsky, 1975). Although many issues about the possible implementations of frame-based systems are unresolved, and others may not have surfaced yet, the basic idea of frame-like structuring of knowledge seems promising.

Conclusion

This brief summary of knowledge representation indicates the variety of techniques being used in AI projects. The remaining articles in this Chapter go into most of these schemes in greater detail. Many researchers feel that the representation of knowledge is the key issue at this point in the development of AI. Knowledge representation is also one area where AI and cognitive psychology share fundamental concerns, for the brain's operation is sometimes best described in terms of one representation formalism, sometimes by another, and sometimes not very well by any we have thought up. The interested reader should peruse the chapter on Information Processing Psychology.

References

The best current survey of knowledge representation in AI is in Winograd (1980b).
References on each of the schemes are given after the corresponding article below.

C. Representation Schemes

C1. Logic

Philosophers have been grappling with the nature of reasoning and knowledge since the time of the ancient Greeks. This tradition, formalized in the last half of the nineteenth century with the work of Boole, Frege, and Russell, and expanded and amplified in the current century by philosophers such as Quine, Carnap, and Tarski, is an important part of western intellectual history and has developed into the philosophical and mathematical study of *logic*.

This article is about logic, about how the formal treatment of knowledge and thought, as developed in philosophy, has been applied to the development of computer programs that can reason. The first two sections of the article, dealing with the *propositional* and *predicate calculi*, are an introduction to formal logic. This particular introduction has been written with the Artificial Intelligence applications of logic in mind. It is followed by an illustration of the way that a simple problem, the famous *Tower of Hanoi puzzle*, might be formalized in the predicate calculus. Then, after a survey of some of the important AI systems that have used logic for a representation, we discuss the advantages and problems of this representational formalism in AI.

The Propositional Calculus

Logic, one of the first representation schemes used in AI, has two important and interlocking branches. The first is consideration of *what can be said*; what relationships and implications one can formalize, the *axioms* of a system. The second is the deductive structure, the *rules of inference* that determine what can be inferred if certain axioms are taken to be true. Logic is quite literally a formal endeavor: it is concerned with the form, or *syntax*, of statements and with the determination of truth by *syntactic* manipulation of formulae. The expressive power of a logic-based representational system results from building. One starts with a simple notion, like truth and falsehood, and, by inclusion of additional notions, like conjunction and predication, develops a more expressive logic--one in which more subtle ideas can be represented.

The most fundamental notion in logic is that of *truth*. A properly formed statement or *proposition* has one of two different possible *truth values*. TRUE or FALSE. Typical propositions are "Bob's car is blue," "Seven plus six equals twelve," and "John is Mary's uncle." Note that each of the quoted sentences is a proposition, not to be broken down into its constituent parts. Thus, we could assign truth value TRUE to the proposition "John is Mary's uncle," with no regard for the *meaning* of "John is Mary's uncle," i.e. that John is the brother of one of Mary's parents. Propositions are those things which we can call true or false. Terms such as "Mary's uncle" and "seven plus four" would not be propositions, as we cannot assign a truth value to them.

Pure, disjoint propositions aren't very interesting. Many more of the things we say and think about can be represented in propositions which use *sentential connectives* to combine simple propositions. There are five commonly employed connectives:

And	\wedge	or	\vee
Or	\vee		
Not	\neg		
Implies	\supset	or	\rightarrow
Equivalent	\equiv		

The use of the sentential connectives in the syntax of propositions brings us to the simplest logic, the *propositional calculus*, in which we can express statements like *The book is on the table or it is on the chair*, and *If Socrates is a man, then he is mortal*. In fact, the meanings of the sentential connectives are intended to keep their *natural* interpretations, so that if X and Y are any two propositions,

$X \wedge Y$ is TRUE if X is TRUE and Y is TRUE; otherwise $X \wedge Y$ is FALSE.

$X \vee Y$ is TRUE if either X is TRUE or Y is TRUE or both.

$\neg X$ is TRUE if X is FALSE, and FALSE if X is TRUE.

$X \supset Y$ is meant to be the propositional calculus rendition of the notion, *If we assume that X is true, then Y must be so*, i.e. the truth of X implies that Y is true. We use this concept in everyday speech with statements like *If Jenny is nine months old, then she can't do calculus*. The truth value of $X \supset Y$ is defined to be TRUE if Y is TRUE or X is FALSE.

$X \equiv Y$ is TRUE if both X and Y are TRUE, or both X and Y are FALSE; $X \equiv Y$ is FALSE if X and Y have different truth values.

The following table, a compressed *truth table* summarizes these definitions.

X	Y	$X \wedge Y$	$X \vee Y$	$X \supset Y$	$\neg X$	$X \equiv Y$
T	T	T	T	T	F	T
T	F	F	T	F	F	F
F	T	F	T	T	T	F
F	F	F	F	T	T	T

From syntactic combinations of variables and connectives, we can build sentences of propositional logic, just like the expressions of mathematics. Parentheses are used here just as in ordinary algebra. Typical sentences are:

$$(X \supset (Y \wedge Z)) \equiv ((X \supset Y) \wedge (X \supset Z)) \quad (1)$$

$$\neg(X \vee Y) \equiv (\neg X \wedge \neg Y) \quad (2)$$

$$(X \wedge Y) \vee (\neg Y \wedge Z) \quad (3)$$

Sentence (1) is a *tautology*; it states "Saying X implies Y and Z is the same as saying that X implies Y and X implies Z." This is a tautology because it is true no matter what propositions are substituted for the *sentential constants* X, Y, and Z. Sentence (2) is a *fallacy* or

contradiction. No matter what assignment of values is used, the sentence is always false. (It states "Saying X or Y is false is the same as saying that 'X is false and Y is false' is false.") Sentence (3) is neither a tautology or a fallacy. Its truth value depends on what propositions are substituted for X, Y, and Z.

In the propositional calculus, we also encounter the first *rules of inference*. An inference rule allows the deduction of a new sentence from previously given sentences. The power of logic lies in the fact that the new sentence is assured to be true if the original sentences were true. The most well known inference rule is *modus ponens*. It states that if we know that two sentences of the form X and $X \supset Y$ are true, then we can *infer* that the sentence Y is true. For example, if we know that the sentence *John is an uncle* is true and we also know that *If John is an uncle then John is male* is true, then we can conclude that *John is male* is true. More formally, the *modus ponens* rule would be expressed as:

$$(X \wedge (X \supset Y)) \supset Y$$

Note that if we think of X and $X \supset Y$ as two entries in a database, the *modus ponens* rule allows us to replace them with the single statement, Y, thus *eliminating* one occurrence of the connective " \supset ". In what are called *natural deduction* systems of logic, there are typically two rules of inference for each connective, one that *introduces* it into expressions and one that eliminates it. *Modus ponens* is therefore called the \supset -*elimination* rule.

The Predicate Calculus

For the purposes of AI, propositional logic is not very useful. In order to adequately capture in a formalism our knowledge of the world, we not only need to be able to express true or false propositions, but we must also be able to speak of objects, to postulate relationships between these objects, and to generalize these relationships over classes of objects. We turn to the *predicate calculus* to accomplish these objectives.

The predicate calculus is an extension of the notions of the propositional calculus. The meanings of the connectives (\wedge , \vee , \supset , \neg , and \equiv) are retained, but the focus of the logic is changed. Instead of looking at sentences that are of interest merely for their truth value, predicate calculus is used to represent statements about specific objects, or *individuals*. Examples of individuals are *you*, *this sheet of paper*, *the number 1*, *the queen of hearts*, *Socrates*, and *that coke can*.

Predicates. Statements about individuals, both by themselves and in relation to other individuals, are called *predicates*. A predicate is applied to a specific number of arguments, and has a value of either TRUE or FALSE when individuals are used as the arguments. An example of a predicate of one argument is the predicate *is-red*. Of the individuals mentioned in the previous paragraph, the predicate *is-red* has the value TRUE when applied to the individuals *the queen of hearts* and *that coke can*, and FALSE when the individual *this paper* is used as the argument. Other examples of predicates are *less-than-zero*, *Greek*, *mortal*, and *made-of-paper*.

Predicates can have more than one argument. An example of a two-place predicate from mathematics is *is-greater-than*, e.g., *is-greater-than (7,4)*. Physical objects could be compared by the two-place predicate *is-lighter-than*. A three-place predicate from geometry

might be *Pythagorean*, which takes three line-segments as arguments and would be TRUE whenever two were the sides of a right triangle with the third as its hypotenuse. One very important two place predicate is *equals*.

Each one-place predicate defines what is called a *set* or *sort*. That is, for any one place predicate, P, all individuals X can be sorted into two disjoint groups, those objects that *satisfy* P (for which P(X) is TRUE) forming one group, and those that don't satisfy P in the other. Some sorts include other sorts; all men are animals, all knaves are playing-cards.

Quantifiers. We shall often have occasion to refer to facts that we know to be true of all or some of the members of a sort. For this, we introduce two new notions, those of *variable* and *quantifier*. A variable is a place holder, one that is to be filled in by some constant, as X has been used in this article. There are two quantifiers, \forall , meaning *For all...*, and \exists , meaning *There exists...*. The English language sentence "All men are mortal" is thus expressed in predicate calculus using the variable X as

$$\forall X. \text{Man}(X) \supset \text{Mortal}(X) ,$$

which is loosely rendered "For all individuals, X, if X is a man (i.e., Man(X) is true) then X is mortal." The English sentence "There is a playing card that is red and is a knave" becomes the predicate calculus statement

$$\exists X. \text{Playing-card}(X) \wedge \text{Knave}(X) \wedge \text{Is-red}(X) .$$

More complicated expressions or *well-formed formulae* (WFFs) are created with syntactically allowed combinations of the connectives, predicates, constants, variables, and quantifiers.

Inference rules for quantifiers. In a typical natural deduction system, use of the quantifiers implies the introduction of four more *inference rules*, one for the introduction and elimination of each of the two quantifiers. For example, the \forall -elimination, or *universal specialization*, rule states that, for any well-formed expression α , that mentions a variable X, if we have

$$\forall X. \alpha(X)$$

we can conclude

$$\alpha(A)$$

for any individual A. In other words, if we know, for example,

$$\forall X. \text{Man}(X) \supset \text{Mortal}(X)$$

we can apply this to the individual Socrates, using the \forall -elimination rule, to get:

$$\text{Man}(\text{Socrates}) \supset \text{Mortal}(\text{Socrates}).$$

The rules of the propositional calculus, extended by predicates, quantification, and the inference rules for quantifiers, result in the *predicate calculus*.

First-order logic. Predicate calculus, as we've described it, is very general, and often quite clumsy. Two other additions to the logic will make some things easier to say, without really extending the range of what can be expressed. The first of these is the the notion of operators, or *functions*. Functions, like predicates, have a fixed number of arguments; but functions are different from predicates in that they do not just have the values TRUE or FALSE, but they "return" objects related to their arguments. For example, the function *uncle-of* when applied to the individual *Mary* would return the value *John*. Other examples of functions are *absolute-value*, *plus*, and *left-arm-of*. Each of the arguments of a function can be either a variable, a constant, or a function (with its arguments). Functions can, of course, be combined; we can speak of the *father-of* (*father-of* (*John*)), who would, of course, be John's paternal grandfather.

The second important addition is that of the predicate *equals*. Two individuals X and Y are equal if and only if they are indistinguishable under all predicates and functions. More formally, $X=Y$ if and only if, for all predicates P, $P(X) \equiv P(Y)$, and also for all functions F, $F(X)=F(Y)$. What we arrive at with these additions is no longer pure predicate calculus; it is a variety of *first-order logic*. (A logic is of first order if it permits quantification over individuals, but *not* over predicates and functions. For example, a statement like *All predicates have only one argument* cannot be expressed in a first-order theory.) First-order logic is both *sound* (it is impossible to prove a false statement) and *complete* (any true statement has a proof). The utility of these properties in AI systems will be discussed after we present the *axiomatization* of a sample problem, i.e., its formal expression in sentences of first-order logic.

A Sample Axiomatic System

So far we have sketched the language of logic, its parts of speech, grammar, etc. We have not talked about how to express a problem to be solved in this language. Deciding how to express the notions he needs is up to the user of logic, just as a programmer must construct programs from the elements presented in the programming language manual. However, a good programming manual ought to present sample programs; we present a sample *axiomatization* of the famous Tower of Hanoi problem (see article Search.B2). One common version of this puzzle involves three pegs, 1, 2, and 3, and three disks, A, B, and C of graduated sizes. Initially the disks are stacked on peg 1, with A, the smallest, on top and C, the largest, at the bottom. The problem is to transfer the stack to peg 3, as in Figure 1, given that (a) only one disk can be moved at a time, and it must be "free", i.e., have no other disks on top of it, and (b) no disk may ever be placed on top of a smaller disk.



Figure 1. The Tower of Hanoi puzzle.

Problem expression and solution in logic has several parts. We must first specify the vocabulary of our domain; what are the variables, constants, predicates and functions.

Secondly, we define *axioms*, expressions which we assert state the necessary relationships between the objects needed to model our domain.

Obvious objects (constants) for this axiomatization are the disks, A, B, and C, and the pegs, 1, 2, and 3; obvious predicates are the *sorts* DISK and PEG. DISK(A) is TRUE, since A is a disk; PEG (C) is FALSE.

We need also to be able to compare disk size; for that, we have a binary predicate, SMALLER. We define SMALLER (A,B) to be TRUE if and only if disk A is smaller than disk B. If we have variables X, Y, and Z denoting disks, we can have our first axiom express the transitivity of SMALLER:

$$\forall X Y Z. (\text{SMALLER}(X,Y) \wedge \text{SMALLER}(Y,Z)) \supset \text{SMALLER}(X,Z)$$

In other words, *If disk X is smaller than Y, and Y is smaller than Z, then X is smaller than Z.* The given size relationships between the disks is stated by the premise

$$\text{SMALLER}(A,B) \wedge \text{SMALLER}(B,C)$$

Note that by using the preceding two expressions, we can establish, or *prove*, SMALLER (A,C).

We need to be able to talk about the status of our problem solving as we work through a solution and to be able to compare the status after a series of moves. A common strategy to deal with this difficulty is that of introducing a *situational* constant. A situation is a "snap shot" of where things are at any given point. Thus, in the Tower of Hanoi problem, we might have disk C under disk A at some point called, say, situation SIT12.

The vertical relationships of the disks and the pegs are, after all, the primary predicate of this problem. Hence, we need a three-place predicate ON (X,Y,S), which asserts that disk X is on disk (or peg) Y in situation S. The axiomatization of the fact that a disk is "free," i.e., has no other disk on it, in a situation S becomes:

$$\forall X S. \text{FREE}(X,S) \equiv \neg \exists Y. (\text{ON}(Y,X,S))$$

which is read "For all disks X and situations S, X is free in situation S if and only if there does not exist a disk Y such that Y is ON X in situation S." Notice how specific and concise the formal statement of these relationships can be.

To describe the idea that moving a disk, X, onto Y is "legal" in a given situation only if both X and Y are free and Y is bigger, we create the predicate LEGAL:

$$\forall X Y S. \text{LEGAL}(X,Y,S) \equiv (\text{FREE}(X,S) \wedge \text{FREE}(Y,S) \wedge \text{DISK}(X) \wedge \text{SMALLER}(X,Y))$$

Now all we lack is a way of generating new situations. The *function* MOVE, defined on two objects and a situation, produces a new situation where the first object is on top of the second. And what does this new situation, S', look like? Well, X is on Y, and nothing else has changed:

$$\begin{aligned}
 & \forall S S' X Y. \\
 & S' = \text{MOVE}(X, Y, S) \supset (\text{ON}(X, Y, S') \\
 & \quad \wedge \forall Z Z1. ((\neg Z=X \wedge \neg Z1=Y) \\
 & \quad \quad \supset (\text{ON}(Z, Z1, S) = \text{ON}(Z, Z1, S')))) \\
 & \wedge \forall Z. (\text{ON}(X, Z, S) \supset \text{FREE}(Z, S'))
 \end{aligned}$$

S' will be a SITUATION if that MOVE is LEGAL:

$$\forall X Y S. \text{LEGAL}(X, Y, S) = \text{SITUATION}(\text{MOVE}(X, Y, S))$$

This example gives an idea of how to express in first-order logic the notions involved in the Tower of Hanoi problem. The solution of the problem involves *proving* that the goal state can be reached from the original state. More precisely, one proves a theorem which states that, given the problem's premises expressed like the ones above, such a goal state exists. There are very general methods for automated *theorem proving* in first-order logic, involving programs which manipulate internally stored logical expressions using rules of inference supplied as procedures. The details of this process are discussed in the Theorem Proving chapter. The AI systems described in the next section all use some kind of theorem prover to make deductions based on facts expressed as logic formulae.

Applications of Logic to Artificial Intelligence

In this section we shall survey various Artificial Intelligence systems which use logic to represent knowledge. We shall also mention the different processes they use to make deductions, since this is an equally important aspect of the system, but we will not describe the various alternatives in detail.

QA3 (Green, 1969) was a general-purpose *question-answering* system, which solved simple problems in a number of domains. Deductions were performed using the *resolution method* of inference, with simple general heuristics for control. The system could solve problems in chemistry, robot movement, puzzles (like the Tower of Hanoi), and automatic programming. An example from chemistry is, given the following facts (among others) about ferrous sulfide (FeS):

FeS is a sulfide, it is a dark-gray compound, and it is brittle,

represented as the first-order logic formula

$$\text{sulfide}(\text{FeS}) \wedge \text{compound}(\text{FeS}) \wedge \text{darkgray}(\text{FeS}) \wedge \text{brittle}(\text{FeS}),$$

QA3 could then answer questions like, "Is it true that no dark-gray thing is a sulfide?" i.e.,

$$\neg \exists X. \text{darkgray}(X) \wedge \text{sulfide}(X) ?$$

Despite its generality and its success on simple problems, QA3 could not handle really hard problems. The fault lay in the method of deduction, resolution theorem proving, which became impossibly slow as the number of facts it knew about a particular domain increased beyond just a few. In AI jargon, there was a *combinatorial explosion* in the number of ways to

combine facts to make inferences, and the resolution method of proving theorems was too indiscriminate in which combinations it tried. Although unrestricted resolution is *complete*, in the sense that it will always return an answer if one exists, it is too *indirect* for non-trivial problems. Even the use of *heuristic* rules to suggest which alternatives might be most profitably followed at various points of the proof did not constrain the search sufficiently to make QA3's approach feasible.

STRIPS (Fikes, Hart, and Nilsson, 1972), short for Stanford Research Institute Problem Solver, was designed to solve the *planning* problems faced by a robot in rearranging objects and navigating in a cluttered environment. Since the representation of the world must include a large number of facts dealing with the position of the robot, objects, open spaces, etc., simpler methods often used for puzzles or games would not suffice. The representation scheme chosen for STRIPS was again the first-order predicate calculus.

A simple problem was:

Given a robot at point A, and boxes at points B, C, and D, gather the boxes together.

The current situation is described as:

ATR (A)
 AT (BOX1, B)
 AT (BOX2, C)
 AT (BOX3, D)

and the goal as

$\exists X. AT(BOX1,X) \wedge AT(BOX2,X) \wedge AT(BOX3,X)$

i.e., *get all the boxes together at some place, X*. Problem-solving in a robot domain such as this involves two types of processes: (a) deduction in a particular world model, to find out whether a certain fact is true, and (b) searching through a space of world models, to find one in which the given condition is satisfied (e.g., how can we get the three blocks together?)

The former process is usually called *question answering*; the latter, *planning*. STRIPS used different methods to solve these two types of problems. Question answering was done via resolution theorem proving, as in Green's QA3 system; planning was performed via *means-ends analysis*, as in the GPS system of Newell and Simon (1972). This dual approach allowed world models more complex and general than in GPS, and provided more powerful search heuristics than those found in theorem-proving programs. GPS, STRIPS, and its successor ABSTRIPS are described in detail in the Search chapter.

FOL (Filman and Weyhrauch, 1976) is, among other things, a very flexible proof checker for proofs stated in first-order logic. Deduction is done using the *natural deduction* system of (Prawitz, 1965), which includes the *introduction* and *elimination* rules of inference discussed above. FOL can more properly be viewed as a sophisticated, interactive environment for using logic to study epistemological questions (see Weyhrauch, 1978).

Logic and Representation

First-order logic, as we have described it, demands a clean syntax, clear semantics, and, above all, the notions of *truth* and *inference*. Clarity about what is being expressed and about the consequences of, or possible inferences from, a set of facts is perhaps the most important quality of this formalism.

In a classic paper, McCarthy and Hayes (1969) differentiate two parts of the AI problem. The *epistemological* part was defined as determining "what kinds of facts about the world are available to an observer with given opportunities to observe, how these facts can be represented in the memory of a computer, and what rules permit legitimate conclusions to be drawn from these facts" (McCarthy, 1977). The issue of processing, of using the knowledge once represented, what McCarthy and Hayes called the *heuristic* part of the AI problem, was separated from the representation issue. Given this distinction, logic can be a useful means for exploring the epistemological problems for several reasons.

1. Logic often seems a *natural* way to express certain notions. As McCarthy (1977) and Filman (1979) point out, the expression of a problem in logic often corresponds to our intuitive understanding of the domain. Green (1969) also indicated that a logical representation was easier to reformulate; thus, experimentation is made easier.
2. Logic is *precise*. There are standard methods of determining the *meaning* of an expression in a logical formalism. Hayes (1977a) presents a complete discussion on this issue, and argues for the advantages of logic over other representation systems on these grounds.
3. Logic is *flexible*. Since logic makes no commitment to the kinds of processes which will actually make deductions, a particular fact can be represented in a single way, without having to consider its possible use.
4. Logic is *modular*. Logical assertions can be entered in a data base independently of each other; knowledge can grow incrementally, as new facts are discovered and added. In other representational systems, the addition of a new fact might sometimes adversely affect the kinds of deductions which can be made.

The major disadvantage of logic also stems from the separation of *representation* from *processing*. The difficulty with most current Artificial Intelligence systems lies in the heuristic part of the system, i.e., in determining how to use the facts stored in the system's data structures, not in deciding how to store them (for example, QA3's failure with large databases). Thus, separating the two aspects and concentrating on epistemological questions merely postpones addressing the problem. Work on *procedural representation* schemes such as PLANNER (article C2) and on *frame-based* schemes (article C7) are efforts to incorporate the heuristic aspect into the epistemological; systems like GOLUX (Hayes, 1977b) and FOL (Weyhrauch, 1978) are attempts to formalize control of processing, while retaining the logical precision.

References

Nilsson (1971) gives a brief, elementary introduction to the use of logic in Artificial Intelligence in chapters six and seven, including an introduction to automated theorem proving techniques. More thorough discussions can be found in any of the introductory logic texts, like Manna (1973) or Suppes (1957).

C2. Procedural Representations of Knowledge

The distinction between *declarative* and *procedural* representations of knowledge has had a key role in the historical development of AI ideas. Declarative representations stress the "static" aspects of knowledge--facts about objects, events, their relationships, and states of the world. The proponents of procedural representations pointed out that AI systems had to *know how to use* their knowledge--find relevant facts, make inferences, etc.--and that this aspect of knowledgeable behavior was best captured in procedures. (Hayes, 1977a, discusses the different kinds of knowledge amenable to different types of representation schemes.)

As a simple example of what it means to represent knowledge procedurally, consider what a typical alphabetization program could be said to "know" about its task. The knowledge that "A comes before B in the alphabet," is represented *implicitly* in the body of the alphabetization procedure, which really does an integer comparison of the computer codes for A and B. All computer programs incorporate procedural knowledge of this sort. What the proceduralists pointed out was that, while the knowledge about alphabetical order was implicit in such a system, the knowledge about *how to alphabetize* was represented *explicitly* in the alphabetization procedure. On the other hand, in declarative system, where knowledge about alphabetical order might be explicitly represented as facts like *A comes before B*, *B comes before C*, etc., the knowledge about how to alphabetize is implicit in the programs that manipulate those facts (theorem prover, production system interpreter, etc.).

Before the advent of proceduralism, workers in AI focused on determining what kinds of knowledge could be represented adequately in formalisms like *logic* and *semantic nets*. Questions about how the data structures involved could be manipulated effectively as the databases grew larger were considered a secondary concern. The proceduralists took exception to this view. They argued that the useful knowledge of a domain is intrinsically bound up with the specialized knowledge about how it is to be used (Hewitt, 1975). Through an evolving series of new systems and *AI programming languages*, the proponents of procedural knowledge representation brought concerns about the relevance and utility of knowledge into the center of knowledge representation research.

Early Procedural Systems

The first AI systems that might be called procedural were not extreme in their stance: their factual knowledge was stored in a database similar to those used by the then popular theorem-proving programs, but their *reasoning* was structured in a new way. (See Winograd, 1972, for a discussion of the development of proceduralist ideas.)

Such an approach was used in Raphael's (1968) early *question-answering* system. SIR (see article *Natural Language.F1*). SIR could answer questions about simple logical relationships, such as *Is a finger part of a person?* Its knowledge was stored in two forms: facts about the parts of things were represented as properties "linked" to the nodes representing the objects and the inference-making mechanism was represented as specialized procedures. Thus, to answer the question about a finger, SIR would use two stored facts, that a finger is part of a hand and that a hand is part of a person, and one procedure, a specialized induction procedure that traced *part-of* links between nodes. The inference routines were specialized in that they had to be custom-built for each new type of inference and link in the database.

However, the most characteristically procedural quality of the SIR system was that the *meaning* of an input sentence or question, the final result of SIR's *parsing* stage, was a procedure. When executed, this routine performed the desired action--either adding to the database or printing information found therein. In other words, when the sentence *A finger is part of a hand* was entered into the system, SIR produced and immediately executed a procedure which added a PART-OF link to the database between the FINGER node and the HAND node.

Woods (1968) implemented the most sophisticated of the early procedural systems. His program handled questions about airline flight schedules. The questions were translated into functions which, when run over the system's database, produced the correct response. For example, the question *What American Airlines flights go from Boston to Chicago?* would be translated into the query language as:

```
(FOR-EVERY X1/FLIGHT;
      EQUAL (OWNER(X1), AMERICAN-AIRLINES)
      AND CONNECT (X1, BOSTON, CHICAGO);
      LIST(X1))
```

This expression of the question is a function (in the LISP programming language) built from other specialized-knowledge procedures like FOR-EVERY and CONNECT. When evaluated, this function would retrieve a list of all of the flights in the data base, then find out which of those were owned by American Airlines, then which of those went from Boston to Chicago, and finally print the resulting list.

Representing How to Use Knowledge

The great advantage of the early procedural systems was that they were *directed* in their problem solving activity in the sense that they did not use irrelevant knowledge or follow unnatural lines of reasoning. These inefficient types of behavior, characteristic of the early declarative systems that blindly tried to apply anything they knew to the problem at hand until something worked, were eliminated by the specialized inference procedures. But this solution created problems of its own. In general, as procedural systems become very complex, they become very hard for people to understand and modify.

Thus, in the late 1960's there was an effort to merge the two types of representation, seeking the ease of modification of the declarative systems (especially *logic*) and the directedness of the earlier procedural systems. The essence of this approach was to represent declarative knowledge of the type typically encoded in logic expressions along with *instructions for its use*. The thrust of the later work in procedural representations was to try to find better ways of expressing this *control* information.

Information about how to use a piece of knowledge might concern various aspects of processing. One form of control is to indicate the *direction* in which an implication can be used. For example, to encode the idea that to prove that something flies you might want to show that it is a bird, from which the desired conclusion follows, one might write something like this:

(IF-NEEDED FLIES (X)
TRY-SHOWING BIRD (X))

Thus, if we are trying to prove that Clyde can fly, this *heuristic* tells us to first try to prove that he is a bird. Note that the knowledge that *All birds can fly*, as represented here, is not usable in the other direction--if we learn that Fred is a bird, we will not be able to immediately conclude anything about Fred being able to fly.

Another use of procedural knowledge occurs when we try to specify what knowledge will be relevant to achieving a specific goal. For example, if expect that we might want to prove that something is a bird, and that two facts called, say, THEOREM1 and THEOREM2 might be useful in proving birdhood, we could write:

(GOAL BIRD (TRY-USING THEOREM1 THEOREM2)) .

In essence, what has been done here is to embellish the straightforward deduction provided by *resolution* or *natural-deduction* theorem proving in a logic-based representation. The inferences here are *controlled*; we have told the system how and when it can use the knowledge that it has. Three major methods of specifying control information have been tried:

- 1) Specify control by the way in which one states the facts; this is the approach used in the examples above and in PLANNER, discussed below.
- 2) Encode the representation language at a lower level, so that the user has access to the set of mechanisms for specifying the reasoning process. This is the approach taken in the CONNIVER programming language (Sussman and McDermott, 1972, and see article AI Programming Languages.C3).
- 3) Define an additional language, used to express control information, which works together with the representation language. This idea was the foundation for the GOLUX project (Hayes, 1973) and for Kowalski's (1974) predicate calculus programming.

Of the three approaches, the work on PLANNER was widely used and was seminal for later work on procedural representations.

PLANNER: Guided Inference and Extended Logic

PLANNER (Hewitt, 1972) was an *AI programming language* designed to implement both representational and control information. The features of the language are described in article AI Programming Languages.C2, and we discuss here only those aspects relevant to knowledge representation. The specific concern of the PLANNER research was not to facilitate the class of inferences which were logically *possible*, as would be the focus in *theorem proving* work, but to expedite the inferences which were *expected* to actually be needed. This approach created its own problems; there are some quite straightforward deductions which PLANNER is unable to make, as will be discussed later.

The relevant features of the PLANNER language include the ability to specify whether theorems should be used in a forward or backward direction and the ability to recommend the use of specific theorems in given situations, as described in the preceding section. In fact, the ability to recommend pieces of knowledge was somewhat more general than indicated previously. Besides recommending possibly useful theorems by name, general *classes* of theorems could be suggested via the use of *filters*. For example, the idea that the most promising way to prove that something was a bird was to use theorems about zoology could be expressed in PLANNER as:

(GOAL BIRD (FILTER ABOUT-ZOOLOGY)) .

The implementation of these *inference guiding* control features did not change the nature of the possible inferences themselves. However, other procedural knowledge implemented in PLANNER did allow inferences beyond those found in classical logical systems, particularly the various forms of *default reasoning* (Reiter, 1978). One form of default, as implemented in PLANNER, is the THNOT primitive. For example, the expression

(THNOT OSTRICH (X) ASSUME FLIES (X))

refers to all birds, X, and means that unless it can be shown that X is an ostrich, assume that it can fly.

THNOT can only function correctly if certain aspects of our knowledge are *complete*. In the above example, we assume that, if X were an ostrich, we would either know that fact or have some way to deduce it. If the knowledge base is not complete in this sense, the system might make incorrect inferences. This is not necessarily a serious problem; there might be times that we want the system to "jump to conclusions." This will be discussed later.

THNOT and similar functions take us beyond the realm of ordinary logic, since they violate the property of *monotonicity*. Monotonicity states that if a conclusion is derivable from a certain collection of facts, the same conclusion remains derivable if more facts are added. Thus, procedural and declarative systems implement *different logics* (Reiter, 1978). As Hewitt (1972) points out, the logic of PLANNER is a combination of classical logic, intuitionistic logic, and recursive function theory. Winograd (1980a) outlines a taxonomy of *extended inference modes* which are outside the provision of ordinary logic.

PLANNER thus serves as a programming language in which knowledge about both the problem to be solved and the methods of solution can be stated in a modular, flexible style reminiscent of logic. The intent is that the user be able to state as much or as little *domain-specific knowledge* as required. The most extensive use of PLANNER was in Winograd's (1972) SHRDLU system (see article Natural Language.F4). A number of AI programming language projects followed PLANNER, including CONNIVER (Sussman and McDermott, 1972), QA4 (Rulifson, Derkson, and Waldinger, 1972), POPLER (Davies, 1972), and QLISP (Reboh et al., 1976). For further discussion of many of these languages see the AI Programming Languages chapter.

Advantages and Disadvantages of Procedural Representations

Chief among the advantages of using procedures to represent knowledge is their facility for representing *heuristic* knowledge, especially domain-specific information which might enable more *directed* deduction processes. This includes information about whether a theorem should be used in a backward or forward direction, about what knowledge should be used in a given situation, or about which subgoals should be tried first. The most important result of this ability to encode heuristic knowledge is the *directedness* realized by such systems, which of course is crucial in large systems which would get bogged down if the problem solving was not efficient. Efficiency in this sense was the motivation behind most of the work on procedural representations.

A related advantage is the ability to perform *extended-logical* inferences, like the default reasoning described in the preceding section. There are efforts to try to achieve this kind of informal or *plausible reasoning* in more formal logical systems (McCarthy, 1977). Winograd (1980a) discusses this issue further, arguing that these types of inferences are necessary in a system which attempts to model human reasoning.

Procedural representation may also be at an advantage with regard to what is called *modeling*, in particular in relation to the *frame problem* as identified by McCarthy and Hayes (1969). This difficulty, common to all representation formalisms, refers to their inability to model *side effects* of actions taken in the world by making corresponding modifications in the database representing the state of the world. (Note that the *frame problem* has nothing to do with the *frame* as a representation formalism, discussed in article C7.) For example, suppose we are reasoning about a robot with a key moving from ROOM-1 to ROOM-2 to find and unlock a safe. The initial situation, S0, might be represented in the database with assertions like:

1. IN (ROOM-1, ROBOT, S0)
2. IN (ROOM-1, KEY, S0)
3. IN (ROOM-2, SAFE, S0)

After the robot has moved from ROOM-1 to ROOM-2, the system must somehow know that assertions (1) and (2) are now false, while assertion (3) is still true.

In a large system with many facts keeping track of these changes, especially the *side effects* of actions, like the moving of the key to the safe, can be very tricky. The propagation of those facts which have not changed is sometimes much easier in a procedural system: the procedure which *performs* the actions can update the database immediately. (See also the discussion a similar way of dealing with the the frame problem in a *direct representation*, article C5).

Two problems of the procedural approach in relation to more formal representational schemes concern *completeness* and *consistency*. Many procedural systems are not complete, meaning that there are cases in which a system like PLANNER could know all the facts required to reach a certain conclusion, but not be powerful enough to make the required deductions (Moore, 1975). Of course, completeness is not necessarily always desirable. There are cases when we might want the system to work quickly and not spend a long time finding a particular answer or concluding that it cannot find the answer.

A deductive system is consistent if all its deductions are correct--that is, if the

conclusion necessarily follows from the premises. Again, most theorem-proving systems have this property, but procedural systems often do not. For example, the use of default reasoning can introduce inconsistency in the presence of incomplete knowledge. Thus, if we use the fact that *Fred is a bird* to conclude that he can fly, and later discover that he is an ostrich, we will have inconsistency. Hewitt (1975) refers to this as the "Garbage In--Garbage Out" principle.

Like completeness, consistency is not necessarily always desirable. McDermott and Doyle (1980) argue that much of our reasoning is done by revising our beliefs in the presence of new information. Similarly, Hewitt points out that most of our knowledge is not absolute; we regularly accept caveats and exceptions. If we control the reasoning sufficiently tightly in the presence of inconsistency, the Garbage In--Garbage Out effect can be avoided.

Another drawback of procedural representations in their current form is that the control information sometimes gets in the way. For example, if we want to prove that both statements A and B are true, PLANNER allow us to express this as a goal: (THAND A B). But this expression really means, prove A and then prove B--there is no way to state the goal without including some control information.

Another feature which is sacrificed in the procedural approach is the *modularity* of knowledge in the database that was so advantageous in logic and other declarative schemes. In a procedural representation, the interaction between various facts is unavoidable because of the heuristic information itself. Therefore, a change or addition to the knowledge base might have more far-reaching effects than a similar change in a base of logic assertions. In essence, this is the price that is paid for the greater degree of control permitted using procedures.

Two specific criticisms have been directed at PLANNER's method of specifying control. First, it is too *local*: PLANNER is unable to consider the overall shape of the problem's solution and therefore can make only local problem-solving decisions. Second, PLANNER is unable to reason about its control information; ideally, it should be able to make decisions on the basis of facts about control, as it can now make decisions on the basis of facts about the world.

Conclusions

The consensus among AI researchers is that there should be ways to embed control in a deductive system, but that the methods tried thus far have many flaws in them (see, for example, Moore, 1975, Ch. 5). Current research, especially on frame systems (article C7), emphasizes a somewhat different approach to the problem of organizing knowledge with special regard to its expected use, called *procedural attachment*.

References

The most readable discussion of the issues involved in procedural knowledge representation is Winograd (1975). Hayes (1977a) presents these issues from the perspective of a declarativist. The original statement of what procedural representation was all about is Hewitt (1972), and, more readably, Winograd (1972). Winograd (1980b) offers a recent study of knowledge representation in which procedural ideas are fully discussed.

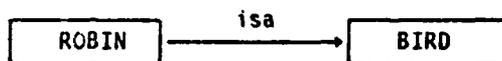
C3. Semantic Networks

Many of the recent systems developed in Artificial Intelligence research use a class of knowledge representation formalisms called *semantic networks*. These representation formalisms are grouped together because they share a common notation, consisting of *nodes* (drawn as dots, circles, or boxes in illustrations) and *arcs*, or *links*, (drawn as arrows) connecting the nodes. Both the nodes and the arcs can have labels. Nodes are usually used to represent *objects, concepts, or situations* in the domain, and the arcs are used to represent the *relationships* between them.

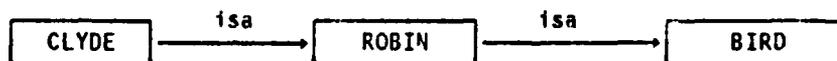
The superficial similarity of this notation is all that some semantic network systems have in common. For example, researchers in psychology, such as Quillian (1968), Norman and Rumelhart (1975), and Anderson and Bower (1973), have developed semantic network systems primarily as psychological models of human memory. Researchers in computer science have been more concerned with developing functional representations for the variety of types of knowledge needed in their systems. Because of these diverse goals, there is no simple set of unifying principles to apply across all semantic network systems. This article, however, will attempt to characterize some of the most common network schemes. We will present a description of how simple concepts are represented in semantic networks and then review some AI systems that use semantic networks. Finally, some more difficult problems in semantic net representation will be mentioned and some of the proposed solutions reviewed.

A Basic Description of the Representation Scheme

Suppose we wish to represent a simple fact like *All robins are birds* in a semantic network. We might do this by creating two nodes to designate "robins" and "birds" with a link connecting them, as follows:



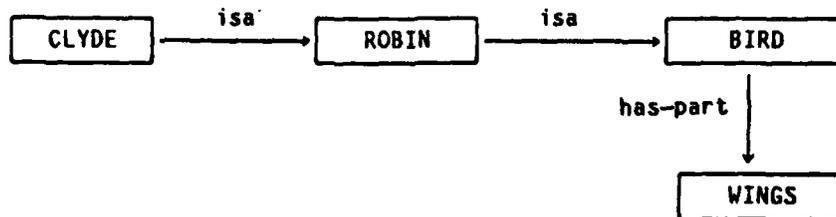
If Clyde is a particular individual whom we wished to assert is a robin, we could add a node for Clyde to the network as follows:



Notice that in this example we have not only represented the two facts we initially intended to represent, but we have also made it very easy to deduce a third fact, namely, that Clyde is a bird, simply by following the ISA links: *Clyde is a robin. Robins are birds. So, Clyde is a bird.* The ease with which it is possible to make deductions about *inheritance hierarchies* such as this one is one reason for the popularity of semantic networks as a knowledge representation. In a domain where much of the reasoning is based on a very complicated *taxonomy*, a semantic network is a natural representation scheme (see, for example, the PROSPECTOR system, article Applications.D2).

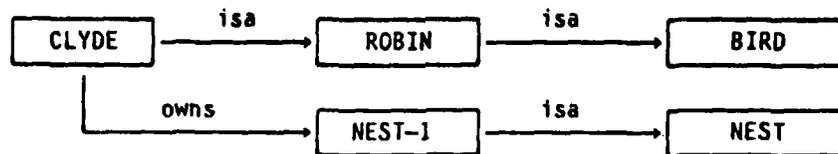
Besides their taxonomic classification, one usually needs to represent knowledge about

the properties of objects. For example, one might wish to express the fact *birds have wings* in the network. We could do this as follows:



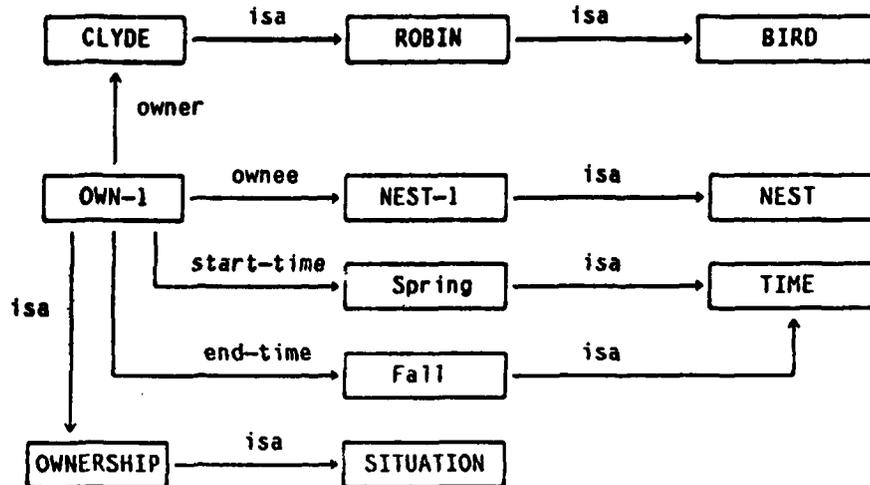
As in the previous example, our choice of representation has made it very easy to write a procedure to make the deductions that robins have wings and that Clyde has wings. All that is necessary is to trace up the ISA-hierarchy, assuming any facts asserted about higher nodes on the hierarchy can be considered assertions about the lower ones also, without having to represent these assertions explicitly in the net. In AI jargon this type of reasoning is called *property inheritance*, and the ISA link is often referred to as a *property inheritance link*.

Suppose we wish to represent the fact *Clyde owns a nest*. Our first impulse may be to encode this fact using an ownership link to a node representing Clyde's nest:



In the above example, NEST-1 is the nest that Clyde owns. It is an *instance* of NEST, i.e., the NEST node represents a general class of objects of which the NEST-1 node represents an example. The above representation may be adequate for some purposes, but it has shortcomings. Suppose one wanted to encode the additional information that Clyde owned NEST-1 from Spring until Fall. This is impossible to do in the current network because the ownership relation is encoded as a link, and links, by their nature, can only encode binary relationships. What is needed is the semantic net equivalent of a four-place *predicate* in logic, that would note the start-time and end-time of the ownership relationship, as well as the owner and the object owned.

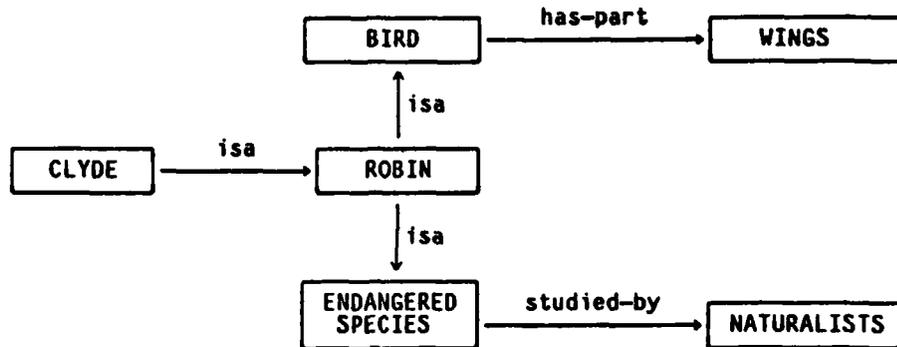
A solution to this problem was proposed by Simmons and Slocum (1972) and later adopted in many semantic net systems: to allow nodes to represent situations and actions, as well as objects and sets of objects. Each situation node can have a set of outgoing arcs, called a *case frame*, which specify the various arguments to the situation predicate. For example, using a situation node with case arcs, the network representation of the fact *Clyde owned a nest from Spring until Fall* becomes



The node NEST-1 is created to represent Clyde's nest, which, of course, ISA nest, as shown. The OWN-1 node represents a particular instance of OWNERSHIP, namely, Clyde owning his nest. And like all nodes which are instances of OWNERSHIP, OWN-1 inherits case arcs to OWNER, OWNEE, START-TIME, and END-TIME nodes. Many semantic network systems use sets of case arcs motivated by linguistic considerations, e.g., general case structures for agent, object, etc. (see article Natural Language.C4). One important use of the case-frame structure for nodes is the possibility of allowing instance nodes, like OWN-1, to inherit *expectations* about, and even *default values* for, certain of their attributes (see article C7 for a discussion of inheritance in *frame systems*).

One more thing to note about the representation scheme described above is that it lends itself to the expression of states and actions in terms of a small number of primitive concepts. For example, FLYING might be considered a type of MOTION, and could be represented by a FLYING node having an ISA arc to the MOTION node and case arcs that describe how flying is a *specialization* of moving. The use of a small number of *semantic primitives* as the basis of a system's knowledge representation has both advantages and disadvantages, and is discussed fully in article C6.

There are still some serious problems with our semantic net representation as it has been developed so far. Suppose one wished to make the assertion *The robin is an endangered species*. The simplest thing to do would be to create the following representation for this fact in our net structure:



This structure indicates that ROBINS are an ENDANGERED SPECIES, and that ENDANGERED SPECIES are studied by NATURALISTS. The problem that is illustrated in this simple example involves inheritance. Since the reasoning procedures, as they've been defined, treat the ISA link as a property inheritance link, the instance node CLYDE inherits all the properties of ENDANGERED SPECIES, just as it inherits the property of *having wings* from the BIRD node. In this way, one might conclude from the fact that *Naturalists study endangered species* that *Naturalists study Clyde*, which may or may not be true.

The source of the problem is that there is as yet no distinction in our network formalism between an *individual* and a *class* of individuals. Furthermore, some things said about a class are meant to be true of all members of a class, like *Robins are birds*, while some refer to the class itself, e.g. *Robins are an endangered species*. Recent semantic net research has explored various ways of making the *semantics* of network structures more precise and of specifying different property inheritance strategies (Woods, 1975; Hendrix, 1976; Brachman, 1979; Stefik, 1980).

A Brief History of Semantic Network-based Systems

The node and link formalism of semantic networks has found use in many AI systems in different application domains. It is impossible to mention every one of these systems here, but we will try to note the highlights and point out where in the AI literature these and related systems are described more fully.

Ross Quillian (1968) designed two early semantic network-based systems that were intended primarily as psychological models of associative memory. Quillian was the first to make use of the node and link formalism, but his networks were simpler than those in the bird's nest example above: They consisted of a number of groups of nodes, called *planes*. Each plane encoded knowledge about a particular concept through interconnections of nodes representing other concepts in the system. These interconnecting links allowed a few simple ways of combining concepts: conjunction, disjunction, and the modification of one concept by another.

Quillian wrote procedures that manipulated the network to make inferences about a pair of concepts by finding connections between the nodes that represented them. The method, called *spreading activation* started from the two nodes and would then "activate" all nodes connected to each of them. Then all of the nodes connected to each of those would in turn be activated, forming an expanding sphere of activation around each of the original concepts. When some concept was activated simultaneously from two directions, a connection had been found. The program would then try to describe the connecting route through the net in a stylized version of English (see article Natural Language.E for an example).

Quillian's second system, called the Teachable Language Comprehender (1969) attempted to solve some of the problems with the original system, like the lack of *directedness* in the net search, and was a bit more complex. Other semantic network-based computer programs that were designed as psychological models of memory, including the HAM program (Anderson and Bower, 1973) and the Active Structural Network system (Norman and Rumelhart, 1975), are described fully in the Information Processing Psychology chapter. Bertram Raphael's (1968) early AI system, SIR, mentioned earlier in article C2, was one of the first programs to use semantic network techniques. SIR could *answer questions* requiring a variety of simple reasoning tasks, such as "A finger is part of a hand, a hand is part of an arm, therefore a finger is part of an arm." Although he did not claim to use a node and link formalism, Raphael's use of binary predicates, such as PART-OF(FINGER, HAND), and reasoning procedures was much like the early semantic net systems and faced many of the same problems that these systems faced. (SIR is described in article Natural Language.F1 on early natural language understanding systems).

In the early seventies, Robert Simmons designed a semantic network representation for use in his natural language understanding research. As mentioned in the previous section, Simmons's system used a linguistically motivated *case frame* approach for choosing arc types. The system could *parse* sentences (using an ATN grammar), translate their meaning into network structures, and finally generate answers to questions using the semantic network (Simmons and Slocum, 1972; Simmons, 1973).

Around the same time, Jaime Carbonell used a semantic network as the basis of his tutoring program, SCHOLAR, which answered questions about the geographical information stored in the net. In a *mixed-initiative* dialogue on the subject of South American geography, SCHOLAR answered questions posed by the student and also generated appropriate questions on its own initiative, giving timely hints when necessary (Carbonell, 1970; Carbonell and Collins, 1974; SCHOLAR is described fully in article Education Applications.C1).

Two of the AI *speech understanding* systems, the systems developed at BBN (Woods et al., 1976) and SRI (Walker, 1976), used semantic networks to represent knowledge about their subject domains (see the Speech chapter). In connection with the SRI speech understanding research, Hendrix (1976) developed the idea of *network partitioning*, which provides a mechanism for dealing with a variety of difficult representation problems including representing logical connectives, quantification, and hypothetical worlds.

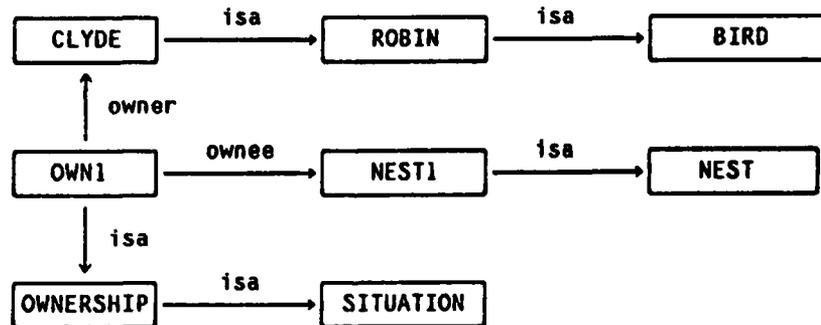
Recent network research involves "structuring" of the nodes and links in the network, and is related to the work on *frame* systems described in article C7. For example, Myooolous, Cohen, Borgida, and Sugar (1975) designed a system for grouping related parts of a semantic network into units called "scenarios." The network was used by the TORUS

system, a program to provide natural language access to a database management system. Hayes (1977b) also designed a system that incorporates higher level structures similar to scenarios, which he calls "depictions."

Reasoning with Semantic Networks

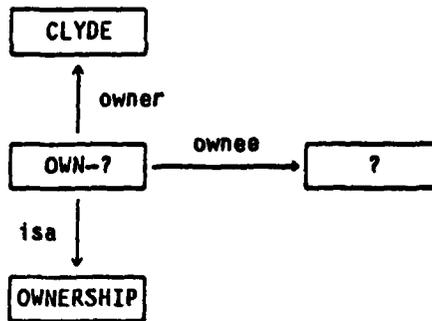
In semantic network representations, there is no *formal semantics*, no agreed up notion of what a given representational structure "means," as there is in *logic*, for instance. Meaning is assigned to a network structure only by the nature of the procedures that manipulate the network. A wide variety of network-based systems have been designed that use totally different procedures for making inferences.

One example of a network reasoning procedure was Quillian's *spreading activation* model of human memory, described above. The reasoning mechanism used by most semantic network systems is based on *matching* network structures: a *network fragment* is constructed, representing a sought-for object or a query, and then matched against the network database to see if such an object exists. Variable nodes in the fragment are *bound* in the matching process to the values they must have to make the match perfect. For example, suppose we use the following network as a database:



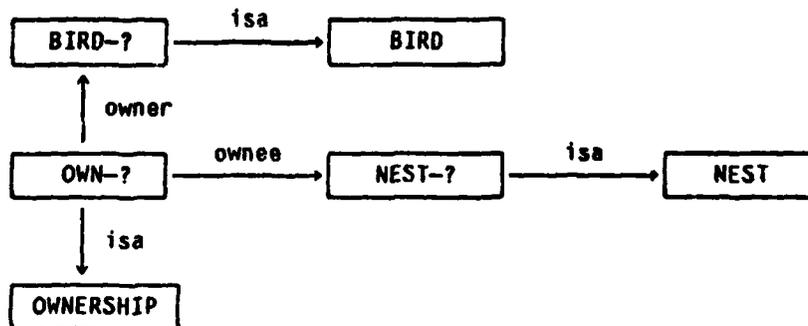
SAMPLE NETWORK DATABASE

and suppose we wish to answer the questions *What does Clyde own?*. We might construct the fragment:



which represents an *instance* of OWNERSHIP in which Clyde is the owner. This fragment is then matched against the network database looking for an OWN node that has an OWNER link to CLYDE. When it is found, the node that the OWNEE link points to is bound in the partial match and is the answer to the question. If no match had been found, of course, the answer would be *Clyde doesn't own anything*.

The matcher can make *inferences* during the matching process to create network structure that is not explicitly present in the network. For example, suppose we wish to answer the question *Is there a bird who owns a nest?* We could translate that question into the following network fragment:



Here, the BIRD-?, NEST-?, and OWN-? nodes represent the yet to be determined bird-owning-nest relationship. Notice that the query network fragment does not match the knowledge database exactly. The deduction procedure would have to construct an ISA link from CLYDE to BIRD to make the match possible. The matcher would bind BIRD-? to the node CLYDE, OWN-? to OWN-1, and NEST-? to NEST-1, and the answer to the question would be *Yes, Clyde*, since the CLYDE node was bound to BIRD-? in order to match the query fragment to the database.

A good example of a network deduction system constructed around this matching paradigm is the SNIFFER system (Fikes and Hendrix, 1977). SNIFFER has the general power

of a *theorem prover* for making deductions from the network database. It is also capable of taking advantage of *heuristic* knowledge embedded in procedures called *selector functions*, which provide advice about which network elements should be matched first and about how to match the selected element. These heuristics allow the system to proceed in a *direct* and sensible way when the amount of information in the database becomes very large and blind retrieval strategies, like spreading activation or systematic matching, are useless because they take too long to retrieve an answer.

Status of Network Representations

Semantic networks are a very popular representation scheme in AI. Node and link structure captures something essential about symbols and pointers in symbolic computation and about association in the psychology of memory. Most current work on the representation of knowledge involves elaboration of the semantic net idea, in particular work on aggregate network structures called *frames*. But like most of the efforts to deal with knowledge in AI, the simple idea of having nodes that stand for things in the world and links that represent the relations between things can't be pushed too far. Besides computational problems that occur when network databases become large enough to represent non-trivial amounts of knowledge, there are many more subtle problems involving the semantics of the network structures. What does a node really "mean," is there a unique way to represent an idea, how is the passage of time to be represented, how do you represent things which are not facts about the world but rather ideas or beliefs, what are the rules about inheritance of properties in networks, etc.? Current research on network representation schemes attempts to deal with these concerns.

References

Introductory discussions of semantic networks can be found in Simmons (1973), Anderson and Bower (1973), Norman and Rumelhart (1975) and Winograd (1980b). Current semantic network research is surveyed in the articles in Findler (1979).

C4. Production Systems

Production systems were first proposed by Post (1943) but have since undergone such theoretical and application-oriented development in AI that the current systems have little in common with Post's formulation. In fact, just as the term *semantic net* refers to a variety of different knowledge representation schemes based on the node and link formalism, so *production system* is used to describe a variety of different kinds of systems based on one very general underlying idea, the notion of condition/action pairs, called *production rules*, or just *productions*. In this article we illustrate the basics of a production system (PS) with an elementary example and discuss some of the design decisions that result in the variety PS architectures. We also describe some of the important AI systems that have been built using PS architectures and discuss current issues in PS design.

A Sample Production System

A production system consists of three parts: (1) a *rulebase* composed of a set of production rules, (2) a special, buffer-like data structure which we shall call the *context*, and (3) an *interpreter* that controls the system's activity. After briefly describing each of these components, we'll go step-by-step through an example to show how a production system works.

A *production rule* is a statement cast in the form "if this *condition* holds, then this *action* is appropriate." For example, the rule

Always punt on 4th down with long yardage required.

might be encoded as the production rule

IF *it is fourth down* AND *long yardage is required* THEN *punt*.

The "IF" part of the productions, called the *condition-part* or *left-hand side*, states the conditions that must occur for the production to be applicable, and the "THEN" part, called the *action-part* or *right-hand side*, is the appropriate action to take. During the execution of the production system, a production whose condition-part is satisfied can *fire*, i.e. can have its action-part executed by the interpreter. Although there are only a few productions in the rulebase of our example below, typical AI systems nowadays contain hundreds of productions in their rulebases.

The *context*, which is sometimes called the *data* or *short-term memory buffer*, is the focus of attention of the production rules. The left-hand side of each production in the rulebase represents a condition that must occur *in the context* data structure before the production can fire. For example, the condition might specify that some symbol is *in the context*, or that something is *not in* it. The actions of the production rules can change the *context* so that other rules will have their condition-parts satisfied. The context data structure may be a simple list, a very large array, or, more typically, a medium size buffer with some internal structure of its own.

Finally there is the *interpreter* which, like the interpreters in all computer systems, is a program whose job is to decide what to do next. In a production system, the interpreter has the special task of deciding which production to fire next.

Consider a production system that might be used to identify food items given a few hints, using a process similar to that used in the game of "20 Questions." The context data structure for this system is a simple list of symbols, called CL. "On-CL X" means that the symbol X is currently in the context. The system has the following rulebase and interpreter.

PRODUCTIONS:

- P1. IF On-CL *green* THEN Put-On-CL *produce*
- P2. IF On-CL *packed in small container* THEN Put-On-CL *delicacy*
- P3. IF On-CL *refrigerated* OR On-CL *produce* THEN Put-On-CL *perishable*
- P4. IF On-CL *weighs 15 lbs* AND On-CL *inexpensive*
AND NOT On-CL *perishable* THEN Put-On-CL *staple*
- P5. IF On-CL *perishable* AND On-CL *weighs 15 lbs* THEN Put-On-CL *turkey*
- P6. IF On-CL *weighs 15 lbs* AND On-CL *produce* THEN Put-On-CL *watermelon*

INTERPRETER:

1. Find all productions whose condition parts are TRUE and make "applicable."
2. If more than one production is applicable, then deactivate any production whose action adds a duplicate symbol to the CL.
3. Execute the action of the lowest numbered (or only) applicable production. If no productions are applicable, then quit.
4. Reset "applicability" of all productions, and return to S1.

The condition-part of each of the productions corresponds to a question one might ask in the 20-Questions game: "Is the item green?," "Does it come in a small container?," etc. The action-parts of the productions represent additions to our knowledge about the unknown item.

Production systems operate in cycles. In each cycle, the productions are examined in a manner specified by the interpreter to see which are appropriate and could fire. Then, if more than one is found appropriate, one production is selected from among them. Finally, the production is fired. These three phases of each cycle are called *matching*, *conflict resolution*, and *action*, respectively.

Suppose the the original knowledge about the mystery food item is that it is green and weighs 15 pounds; in other words, the value of the context list before the beginning of the first cycle is:

CL = (*green, weighs 15 lbs*).

The cycle starts with Step 1 of the interpreter algorithm: finding all the applicable productions by testing their condition-parts. Since only P1 is applicable, Step 2 is not necessary, and Step 3 causes the action-part of P1 to be executed. This adds the symbol *produce* to the CL, representing a new fact about the unknown food item:

CL = (*produce, green, weighs 15 lbs*).

Step 4 ends the first cycle and brings us back to Step 1--finding all the applicable productions. In the second cycle, productions P1, P3, and P6 are all applicable. So in Step 2 we must check if any of these three add a duplicate symbol to the CL. P1 adds *produce*, which is a duplication, so it is eliminated. Then in Step 3 we select P3 to be executed (because it has a lower number than P6), resulting in:

CL = (*perishable, produce, green, weighs 15 lbs*).

In the third cycle, we find that productions P1, P3, P5, and P6 are applicable. Checking, in Step 2, for redundant entries, we eliminate P1 and P3 from consideration. In Step 3, we decide to execute P5, once again because it comes before P6. This results in the context list:

CL = (*turkey, perishable, produce, green, weighs 15 lbs*).

Clearly this last step was a mistake--we wouldn't want to ascribe to something that we know is *green produce* the attribute *turkey*.

In its next two cycles of execution, our example production system will finish. In cycle four, the symbol *watermelon* is added to the context list, and in the last cycle, finding no non-redundant productions to fire, the interpreter finally quits, leaving the context list:

CL = (*watermelon, turkey, perishable, produce, green, weighs 15 lbs*).

If we define the system's "answer" to be the first symbol on the context list, we can ignore the suspicious attribute *turkey*. The reader can probably think of more satisfying ways to "fix up" the rulebase, or the interpreter, such as changing the productions (particularly adding conditions to the condition-part of P5), switching the order of the productions in the rulebase around, adding new productions, and so on. This feeling of *manageability* of the rulebase is perhaps one of the strongest attractions of production systems as a knowledge representation scheme.

Advantages and Disadvantages of Productions Systems

Production systems have most often been used in Artificial Intelligence programs to represent a body of knowledge about how people do a specific, real-world task, like speech understanding, medical diagnosis, or mineral exploration. In psychology, production systems have also been a popular tool for modelling human behavior, perhaps because of the stimulus/response character of production rules (Anderson and Bower, 1973; Newell, 1973). These psychological models are described in the Information Processing Psychology chapter. The AI systems based on productions have been quite diverse in most respects, but there are some features of the production system formalism, both good and bad, that can be generalized.

Modularity. One obvious quality of production systems is that the individual productions in the rulebase can be added, deleted, or changed independently. They behave much like independent *pieces of knowledge*. Changing one rule, although it may change the performance of the system, can be accomplished without worrying about *direct effects* on the other rules, since rules communicate only via the context data structure (looking to see if their conditions are satisfied in the context and then modifying the context); they don't *call* each other directly. This relative *modularity* of the rules is important in building the large rulebases used in current AI systems--knowing what a *proposed new rule* will mean, in whatever situation it is used, makes the creation of the database much easier. There are indications, however, that modularity is harder to maintain as one moves to larger systems (Rychener, 1976), and, even if modularity can be preserved, strongly constraining interaction among rules leads to *inefficiencies* which might become important problems in large systems (see below).

Uniformity. Another general attribute of production systems is the uniform structure imposed on the knowledge in the rulebase. Since all information must be encoded within the rigid structure of production rules, it can often be more easily understood, by another person or by another part of the system itself, than would be possible in the relatively free form of *semantic net* or *procedural* representation schemes, for example. Production systems that examine and automatically modify their own rules are exemplified by those of Waterman, Davis, and Anderson (see below).

Naturalness. A further advantage of the production system formalism is the ease with which one can express certain important types of knowledge. In particular, statements about *what to do* in predetermined situations are naturally encoded into production rules. Furthermore, it is these kinds of statements that are most frequently used by human experts to explain how they do their jobs.

Inefficiency. There are, however, significant disadvantages inherent in the production system formalism. One of these is *inefficiency* of program execution. The strong modularity and uniformity of the productions results in high overhead in their use in problem solving. For example, since PSs perform *every* action via the match/action cycle and convey all information via the *context* data structure, it is difficult to make them efficiently responsive to predetermined sequences of situations or to take *larger steps* in their reasoning when the situation demands it (see Barstow, 1979). Lenat and McDermott (1977) proposes solutions to this type of problem, sacrificing some of the advantages of production systems.

Opacity. A second disadvantage of the production system formalism is that it is hard to follow the flow of control in problem solving--*algorithms* are less apparent than they would be if they were expressed in a programming language. In other words, although situation/action knowledge can be expressed naturally in production systems, algorithmic knowledge is not expressed naturally. Two factors that contribute to this problem are the isolation of productions (they don't *call* each other) and the uniform *size* of productions (there is nothing like a *subroutine hierarchy* where one production can be composed of several sub-productions). Function calls and subroutines, common features of all programming languages, would help to make the *flow of control* easier to follow.

Appropriate Domains for Production Systems

The features of PSs described in the previous section can be seen as having good and bad consequences. A more fruitful way to evaluate the utility of PSs is to characterize the domains for which production rules might be a useful knowledge representation scheme. Davis and King (1977) proposed the following characteristics for appropriate domains:

- a) domains in which the knowledge is diffuse, consisting of many facts (e.g., clinical medicine), as opposed to domains in which there is a concise, unified theory (physics);
- b) domains in which processes can be represented as a set of independent actions (a medical patient-monitoring system), as opposed to domains with dependent sub-processes (a payroll program);
- c) domains in which knowledge can be easily separated from the manner in which it is to be used (e.g., a classificatory taxonomy, like those used in biology), as opposed to cases in which representation and control are merged (e.g., a recipe).

Rychener (1976) rephrased this characterization of appropriate domains in AI terms: if we can view the task at hand as a sequence of transitions from one state to another in a *problem space* (see article Search.Overview), we can model this behavior with production systems, since each transition can be effectively represented by one or more production firings. The following examples of important AI production systems may help demonstrate their utility.

Waterman (1970) implemented an *adaptive production system* to play the game of draw poker (see article Learning.B1). The program was adaptive in that it automatically changed the productions in its rulebase--it started with a set of fairly simple heuristics for playing poker (when to raise, when to bluff, etc.) and extended and improved these rules as it gained experience in actually playing the game. The fact that knowledge in production systems is represented in a constrained, modular fashion facilitated the learning aspect of the system, since the program needed to analyze and manipulate its own representation. Other examples of production systems which model human learning are those of Hedrick (1976), Vere (1977), and Anderson, Kline, and Beasley (1979).

The MYCIN system (Shortliffe, 1976; Davis, Buchanan, and Shortliffe, 1977) acts as a medical consultant, aiding in the diagnosis and selection of therapy for a patients with bacteremia or meningitis infections (see article Medical Applications.C1). It carries out an interactive dialogue with a physician and is capable of *explaining* its reasoning. It also includes a *knowledge acquisition* subsystem, TEIRESIAS, which helps expert physicians to expand or modify the rulebase (article Applications.B). MYCIN's rulebase contains several hundred production rules representing human-expert-level knowledge about the domain. The system is distinguished by its use of a *backward chaining* control structure (see below) and *inexact reasoning*, involving confidence factors which are attached to the conclusion part of each production to help determine the relative strengths of alternative diagnoses.

Lenat (1980) modeled the process of *discovery* in mathematics, viewed as heuristic search, in his AM production system (see article Applications.C2). AM started with a minimal

knowledge of mathematical concepts and used heuristics, represented as production rules, to expand its knowledge about these concepts and *learn* new ones. In the course of its operation, AM discovered a large number of important mathematical concepts, such as prime numbers, the arithmetic functions, etc., and also two mathematical concepts which had not been discovered before. AM is especially important because of its sophisticated data structures and control mechanisms.

Rychener (1976) built several production systems to reimplement a number of AI systems that had been previously developed using other techniques, including Bobrow's STUDENT, Newell and Simon's GPS, Feigenbaum's EPAM, Winograd's SHRDLU, and Berliner's CAPS. Rychener's intent was to show that the production system formalism was a natural one for programming. His primary problem was the difficulty of building very complex control structures (Rychener, 1977).

Current Issues in Production System Design

Complexity of Left- and Right-hand Sides. The structure of the two sides of the productions in the rulebase has been progressively extended as the size and complexity of systems has increased, so that in many current systems the left-hand side (LHS) is a LISP function that can evaluate an arbitrarily complex condition. In some systems, the testing of the LHS can even have *side effects*, so that the rule can alter the context or change the control sequence without ever being fired. Similarly, the form of the RHS has been extended to include *variables*, whose values are bound during the test phase of the cycle, and to allow running arbitrary programs rather than just making changes in the context. These programs usually specify actions in terms of a set of domain-specific conceptual primitives. In some systems (Riesbeck, 1975; Rychener, 1976) these actions could include activation or deactivation of sets of other productions. Again, this represents a radical extension of the original production system formalism.

Structure of the Rulebase and Context. Of the three phases of each production system cycle, matching, conflict resolution, and action, the matching process uses by far the most computational resources. As PSs have become bigger and more complex, questions of efficiency have necessitated making both the rulebase and the context into more complex data structures. For example, in order to allow rapid determination of which productions are applicable in a given situation without checking through all of the rulebase, the productions are often *indexed* or *partitioned* according to conditions that will make them fire (see Davis, 1980, and Lenat and McDermott, 1977). The *context* data structure has increased in internal complexity both for reasons of efficiency and in order to allow it to represent more complicated situations. MYCIN's *context tree* (Shortliffe, 1976), HEARSAY's *blackboard* (Erman and Lesser, 1975), and PROSPECTOR's *semantic net* (Duda, Hart, Nilsson, and Sutherland, 1978) are examples of complex context data structures. A good example of the work in organizing the rulebase and database is that of McDermott, Newell, and Moore (1978).

Conflict Resolution. In practice, it is often the case that more than one rule could fire in each cycle of the operation of a typical large production system; the system is required to choose one from among this set (called the *conflict set*). This *conflict resolution* phase of each cycle is where basic cognitive traits like action sequencing, attention focusing, interruptability, and control of instability are realized. Several different approaches to conflict resolution have been tried, including choosing:

- The *first* rule that matches the context, where *first* is defined in terms of some explicit linear order of the rulebase.
- The highest *priority* rule, where *priority* is defined by the programmer according to the demands and characteristics of the task (as in DENDRAL).
- The most *specific* rule, i.e., the one with the most detailed condition-part that matches the current context.
- The rule which references the element most *recently* added to the context.
- A *new* rule, i.e., a rule/binding instantiation that has not occurred previously.
- A rule *arbitrarily*.
- Not to choose--exploring *all* the applicable rules in parallel (as in MYCIN).

Different systems use different combinations of these simple conflict resolution methods, some of which become quite complicated *scheduling algorithms* (see, for example, AM and HEARSAY). Good discussions of conflict resolution can be found in Davis and King (1977). Also, McDermott and Forgy (1978) discuss the way conflict resolution strategies affect two important characteristics for production systems: *sensitivity*, the ability to respond quickly to changes in the environment, and *stability*, the ability to carry out relatively long sequences of actions. They conclude that no simple conflict resolution strategy can be completely satisfactory.

Direction of Inference. Research on deductive inference has recognized two fundamentally different ways that people reason. Sometimes we work in a *data-driven, event-driven, or bottom-up* direction, starting from the available information as it comes in and trying to draw conclusions that are appropriate to our goals. This is how our sample production system worked, for example. In PS research this is called a *forward chaining* method of inference. On the other hand, we sometimes work the other way, starting from a *goal or expectation* of what is to happen and working backwards, looking for evidence that supports or contradicts our hunch. This is called *goal-driven, expectation-driven, or top-down* thinking, and in production systems it is referred to as *backward chaining*, since it requires looking at the action-parts of rules to find ones that would *conclude* the current goal, then looking at the left-hand sides of those rules to find out what conditions would make them fire, then finding other rules whose action-parts conclude these conditions, and so on. MYCIN's use of backward-chaining is described fully in article *Medical Applications.C1*.

Primitive Vocabulary. As the complexity of the condition- and action-parts of the productions in the rulebase increases, there has been greater concern about the nature of the expressions allowed--the kinds of conditions and actions that can be expressed. A significant aspect of the representation language issue concerns the choice of vocabulary or *semantic primitives*, i.e., the functions or predicates in terms of which the rules and context elements are expressed (see article C6). Different systems will define their vocabulary at higher or lower levels, depending upon the task to be accomplished.

Conclusion

Perhaps the final word on production systems is that they capture in a *manageable* representation scheme a certain type of problem-solving knowledge--knowledge about what to do in a specific situation. Although this kind of knowledge is basically *procedural*, the production system formalism has many of the advantages of declarative representation schemes, most importantly *modularity* of the rules. Furthermore, the way that the productions themselves are structured is very similar to the way that people *talk* about how they solve problems. For this reason, production systems have been used as the backbone for expert AI systems like DENDRAL, MYCIN, and PROSPECTOR (see Applications chapters). Research on these knowledge-based expert systems, called *knowledge engineering* (Feigenbaum, 1977; Bernstein, 1977) is concerned not only with expert-level performance, but also with the *interactive transfer of expertise*--acquisition of knowledge from human experts and explanation of reasoning to human users (Davis, 1980).

References

Winston (1977) gives a basic introduction to production systems with examples. An excellent review of production systems and the issues involved in their design was prepared by Davis and King (1977). Current research in production system design and applications is reported in the collection of papers edited by Waterman and Hayes-Roth (1978).

C5. Direct (Analogical) Representations

There is a class of representation schemes, called analogical or *direct* representations, like maps, models, diagrams, and sheet music, that can represent knowledge about certain aspects of the world in especially natural ways. This type of knowledge representation is central to many AI tasks but seems at first quite different from the usual *propositional* representation schemes like logic and semantic nets. Understanding the sense in which direct and propositional representations are the same may help clarify the meaning of concepts like representation, data structure, and interpretative procedure.

Direct representations have been defined as schemes in which "properties of and relations between parts of the representing configuration represent properties and relations of parts in a complex represented configuration, so that the structure of the representation gives information about the structure of what is represented" (Sloman, 1971). The significant point here is the requirement of correspondence between relations in the representational data structures and relations in the represented situation. For example, a street map is a *direct* representation of a city in the sense that the distance between two points on the map must *correspond* with the distance between the places they represent in town. Hayes (1974) calls this type of connection between the representation and the situation one of *homomorphism* (structural similarity) rather than just *denotation*.

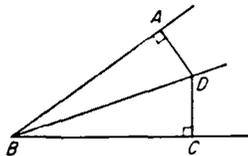
Direct representations may be contrasted with the more prevalent *propositional* or *Fregean* forms (so called after Frege, who invented the *predicate calculus*), which do not require this homomorphic correspondence between relations in the representation and relations in the situation--proximity of assertions in the database of a *logic* system, for instance, indicates nothing about the location of objects in the world. Note that the propositional and direct representations may actually use the same data structures, but differ in *how* they use them--which properties of the data structures are used in what way by the routines which operate on the representation to make inferences. Continuing with the map example, if a routine for examining a map retrieved all distances from an internal table, rather than by looking at the map, it would be pointless to say that the map was *direct* with respect to distance.

Thus, it is the combination of the data structures and the *semantic interpretation function* (SIF) that manipulates them that should be referred to as *direct*, and only with respect to certain properties (Pylyshyn, 1975, 1978). For example, a map (with a reasonable SIF) is direct with respect to location and hence distance, but not, usually, with respect to elevation. For some problems, direct representation has significant advantages. In particular, the problem of *updating* the representation to reflect changes in the world is simpler. For example, if we add a new city to a map, we need only put it in the right place. It is not necessary to explicitly state its distance from all the old cities, since the distance on the map accurately represents the distance in the world. See the discussion of the *j:ame* problem (McCarthy and Hayes, 1969) in article C2.

The distinction between direct and propositional representations has also been the subject of discussions in psychology concerning the character of human memory, which seems to have properties of both types (Pylyshyn, 1973). The next section of this article presents some AI systems which use direct representations. The final section returns to a discussion of their advantages and disadvantages.

Systems Using Analogical Representations

The Geometry Theorem Prover (Gelernter, 1963) was one of the earliest automated theorem provers and was distinguished by its reliance on a *diagram* to guide the proof. The system proved simple, high-school level theorems in Euclidean geometry like the following:



Given: Angle ABD equals angle DBC.
 Segment AD perpendicular segment AB.
 Segment DC perpendicular segment BC.

Prove: Segment AD equals segment CD.

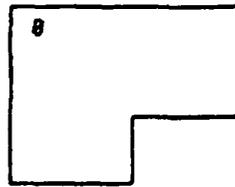
Using *problem reduction* techniques, the system worked backward from the goal to be proved.

Gelernter's system, which is described fully in article Search.D3, is mentioned here because of its use of *diagrams* like the one above. The *problem diagram* was used in two ways. The most important of these was the *pruning heuristic*: "Reject as false any hypothesis (goal) which is not true in the diagram." In other words, those subgoals which were obviously false in the diagram were not pursued via the formal proof methods. This use of the diagram to guide the solution search resulted in the pruning of about 995 out of every 1000 subgoals at each level of search.

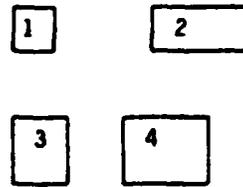
The other use of the diagram was to establish obvious facts concerning, for example, the order that points fall on a given line and the intersection properties of lines in a plane. Many of these are self-evident from the diagram, but would be tiresome to prove from fundamental axioms. In certain such circumstances, the program would assume the fact true if it were true in the diagram, while explicitly noting that it had made such an assumption. The program was also able to add additional lines to the diagram, when necessary, to facilitate the proof.

Work on the General Space Planner (Eastman, 1970, 1973) addressed the task of arranging things in a space, e.g., furniture in a room, subject to given *constraints* that must be satisfied, e.g., room for walkways, no overlapping, etc. A simple problem is:

Given the space



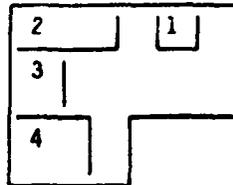
and the objects



and the constraints:

- {3} must be adjacent to {4}
- {2} must be adjacent to {3}
- {1} must be visible from {3}
- {1} must not be adjacent to any other objects,

one solution is:



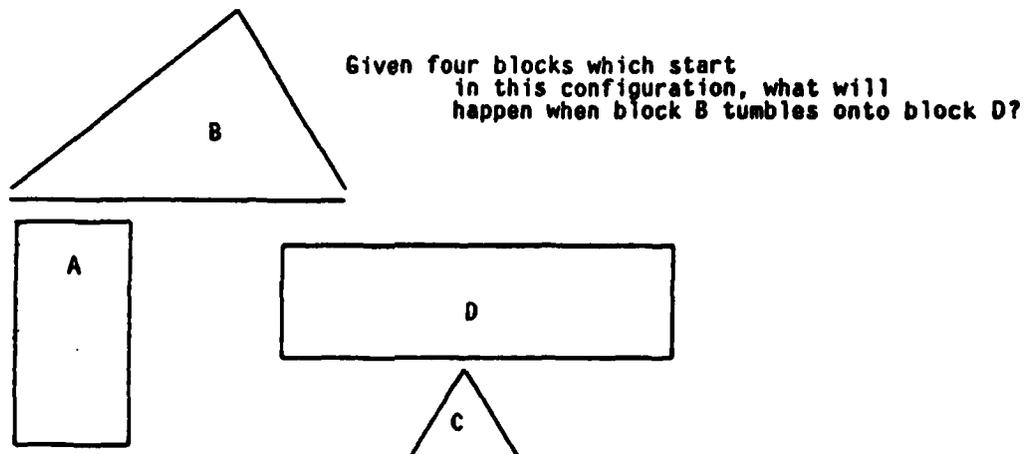
The system used a direct representation, called a *variable domain array*, which was a specialization of the sort of two dimensional "diagram" used by Gelernter. Since the structure of the representation reflected the structure of the space, with respect to the properties of size, shape, and position, the system could be described as analogical for those properties. Space was partitioned into a set of rectangles, and in addition to the above properties, two others that are particularly important for the space planning task were easily detectable from the variable domain array representation: filled vs. empty space and overlapping objects.

The system solved the problems via a *depth-first search* algorithm, finding locations for successive objects and *backing up* when it couldn't proceed without violating some constraint. The search was facilitated by a *constraint graph* which represented, via restrictions on the amount of area left, the effects of constraints between pairs of objects. Thus, by attacking the most restrictive constraint first, the search was relatively efficient. This method has been called *constraint structured planning*.

Note that Eastman's work is in one sense the reverse of Gelernter's. Gelernter's system performed search in a propositional space (sets of formal statements) using an analogical representation (the diagram) for guidance. Eastman's performed search in an analogical space (the diagrammatic array) using a propositional form for heuristic guidance (the constraint graph).

WHISPER (Funt, 1976, 1977) was a system designed to reason exclusively via the

analogical representation. WHISPER operated in a simplified blocks-world environment, solving problems like the following:



The system consisted of three components:

1. The *diagram*, an array which represented the 2-dimensional scene in the obvious way, as shown above.
2. The *retina*, used to "view" the diagram, consisted of a set of parallel receptors, arranged in concentric circles; each receptor viewed a small part of the diagram.
3. The *high-level reasoner*, containing qualitative physical knowledge, the domain-dependent part of the system; it employed information regarding the behavior of rigid bodies when acted upon by gravity.

The significance of the diagram to WHISPER lay in the fact that there were two types of analogs present: between static states of the diagram and the world and also between dynamic behavior of objects in the diagram and of objects in the world.

The correspondences between the diagram and the world were simple and well-defined; no complicated processes were required to map from one to the other. A number of properties, such as position, orientation, and size of blocks, were represented analogically. For these properties, it was not necessary to perform complicated deductions, since the desired information "fell out" of the diagrams. For example, as in Eastman's space planner, to test whether or not a particular area of the world was "empty" (i.e., not occupied by any block), the system had only to "look at" the corresponding area of the diagram. With most propositional representations, it would be necessary to examine each block individually, testing whether or not that block overlapped the space in question (e.g., Fahman, 1974). The retina also provided a number of *perceptual primitives*, including center of area, contact finding, similarity testing, etc. The high-level reasoner never looked at the diagram directly. Note that certain properties (color, weight) were not represented in the analog. To reason about these, normal inference-making processes would be necessary.

Baker (1973) had earlier suggested a similar representational formalism. Like Funt, he envisioned a 2-dimensional array to represent the diagram; however, he also discussed the possibility of retaining spatial "smoothing" information *within* each cell of the array, to remove some of the error induced by the coarseness of the array. Both Funt and Baker suggested that the *parallelism* of their systems, pursuing several goals simultaneously, coupled well with the analogical representations. Individual elements of their processors (in Funt's case, the *retina*) could operate autonomously, with connections only to their (spatially) neighboring cells. In this sort of network, arbitrary transformations, via combinations of translation and rotation, could be represented.

Issues Concerning Direct Representations

The work done to date on direct representations raises a number of questions. First, following Sloman (1975), we should clarify some common misconceptions about direct representations. Analogical representations need not be continuous, nor need they be 2-dimensional; an ordered list of numbers, for example, can be analogical with respect to size. Also, like propositional representations, analogical ones may have a *grammar* which defines what data structures are well-formed or "legal." The difference between the two types of representation schemes lies in the nature of the correspondence between aspects of the structure of the representation and the structure of the represented situation.

One of the advantages of analogical representations over their propositional counterparts relates to the difference between *observation* and *deduction*. In some situations, the former can be accomplished relatively cheaply in terms of the computation involved, and direct representations often facilitate observation since important properties are "directly observable" (Funt, 1976). For example, determining whether three points are } much easier using a direct representation (a diagram) than it would be to calculate analytically using their coordinates. As another example, Filman (1979) implemented a chess reasoning system which relied on both inference (searching several moves ahead) and observation (looking at a *semantic model* of the current state of the chess board).

Funt (1976) relates a more abstract justification for the use of analogical representations. A propositional representation of a situation, e.g., a set of statements in the predicate calculus, will often admit to several *models*. In other words, there might be many situations of the world which would be represented by the same statements, since they are distinguished in aspects that are not captured in the representation. Direct representations, on the other hand, are usually more exhaustive and specific, admitting fewer models, and in turn making for more efficient problem solving.

Additionally, as illustrated by Gelernter's work, the use of analogical representations can facilitate search. Constraints in the problem situation are represented by constraints on the types of transformations applied to the representation, so that impossible strategies are rejected immediately.

There are, however, some disadvantages to the use of these direct representations. First, the tendency toward more specific inference schemes mentioned earlier has its drawbacks--as Sloman (1975) points out, there are times when generality is needed. For example, consider the problem "If I start in room A and then move back and forth between room A and room B, which room will I be in after exactly 377 moves?" For this case, the best

approach is not to simulate the action, but to generalize the effects of odd and even numbers of moves.

Second, Funt notes that some features of the analog may not hold in the actual situation, and we might not know which ones these are. This is related to the general problem of knowing the limits of the representation.

Third, analogical representations become unwieldy for certain types of incomplete information. That is, if a new city is added to a map, its distance from other cities is obtained easily. But, suppose that its location is known only indirectly, e.g., that it is equidistant from cities Y and Z. Then the distance to other cities must be represented as equations, and the power of the analog has been lost.

To conclude, direct representations are *analogous with respect to some properties* to the situation being represented. Some properties (especially physical ones) may be relatively easily represented analogically, resulting in significant savings in computation for certain types of inferences.

References

General discussions of the research on direct representations in AI are Funt (1976), Sloman (1971), and Sloman (1975). Related psychological concerns are discussed by Pylyshyn (1976, 1978).

C6. Semantic Primitives

The knowledge representation formalisms described in this chapter, logic, procedures, semantic nets, productions, direct representations, and frames, are all ways of expressing the kinds of things we express in English--the kinds of things we "know." Having chosen a representation technique, another major question in the design of an AI system concerns the vocabulary to be used within that formalism. In a logic-based representation, for example, what predicates are to be used? In a semantic net, what node and link types should be provided? Research on *semantic primitives* is concerned with this problem of establishing the representational vocabulary. This article, then, is not about a knowledge representation technique per se, but rather about a representational issue that concerns all of the techniques used in AI.

The term *semantic primitive* has no clear-cut definition. As a starting point, one may think of a primitive as any symbol that is used but not defined within the system. The term is so used by Wilks, for example, who accordingly concludes that "primitives are to be found in all natural language understanding systems--even those . . . that argue vigorously against them" (Wilks, 1977c). A second and narrower usage takes semantic primitives to be elements of meaning into which the meanings of words and sentences can be broken down; examples of such work come from linguistics (e.g., Jackendoff, 1975, 1976) and psychology (Miller, 1975; Miller and Johnson-Laird, 1976; Norman and Rumelhart, 1975) as well as from AI.

Additional issues exist about what primitives really are, how they may be used in reasoning, and what alternatives there are to using primitives. Winograd (1978) provides a general survey and analysis of such questions. Some of these are illustrated in the following discussion of the two major AI systems for natural language understanding that are characterized by their authors as using semantic primitives.

Wilks's System

Yorick Wilks, now of the University of Essex, has been developing a natural language system for machine translation since 1968 (described fully in article Natural Language.F2). The system accepts paragraphs of English text, producing from them an internal representation that is a data structure composed of nodes representing semantic primitives. From this structure, a French translation of the input is generated. The translation serves as a test of whether the English has been "understood," a more objective test than just inspection of the internal representation. The translation task also has the advantage, Wilks suggests, that correct translation of the input may often require a shallower understanding than would the ability to answer arbitrary questions about it. Consistent with these reasons for the choice of translation as a task, most of the effort in Wilks's system is spent in converting the English input to the internal representation.

The first major problem that Wilks addressed was the resolution of *word-sense ambiguity*, for this was the problem on which earlier machine translation efforts had foundered (see article Natural Language.B). For example, in the sentence "The old salt was damp," it is necessary to determine from the surrounding context whether "salt" means a chemical compound or a sailor. Wilks's system also addressed problems involving other types of ambiguity and extended word senses, as in "my car *drinks* gas."

The general idea of Wilks's approach, which he calls *preference semantics*, is to use knowledge of possible word meanings to disambiguate other words. For example, part of the meaning of "drink" is that it prefers a fluid object; and part of the meanings of "wine" and "gas" is that they are fluids. If the best fit among possible word senses does not satisfy all preferences (such as the preference of "drink" for an animate subject), then an extended word sense can be accepted. The formalism within which preferences are expressed, Wilks suggests, is closer to a *frame* representation than to a *semantic net*.

As the description above should make clear, a central requirement in Wilks's system is a dictionary distinguishing among the various senses of words that can appear in the input text. Definitions in the dictionary use a vocabulary of *semantic primitives*, grouped into five classes. Examples from each class are given below:

Substantives

MAN	a human
STUFF	a substance
PART	a part of an entity

Actions

CAUSE	causing something to happen
BE	being as equivalence or predication
FLOW	moving as liquids do

Cases

TO	direction toward something
IN	containment

Qualifiers

GOOD	morally correct or approved
MUCH	much, applied to a substance

Type indicators

HOW	being a type of action--for adverbial constructions
KIND	being a quality--for adjectival constructions

In addition to the primitive elements, of which there are currently over eighty, Wilks uses several elements, distinguished by names beginning with an asterisk, that are defined as equivalent to a class of primitives. For example, *ANI (animate) encompasses MAN, FOLK (a human group), BEAST (a nonhuman animal), and certain others. A typical definition using the primitives is the following definition for one sense of the word "break":

```
(BREAK: (*HUM SUBJ)
          (*PHYSOB OBJE)
          (((NOTWHOLE KIND) BE) CAUSE) GOAL)
          (THING INST)
          STRIK)
```

This says roughly that "break" means a STRIKing, done preferably by a HUMAN SUBJECT and preferably with an INSTRUMENT that is a THING, with the GOAL of CAUSing a PHYSICAL OBJECT to BE NOT WHOLE. Words other than verbs are also defined by such structured formulas. A

detailed description of the syntax of such word-sense definitions, or *semantic formulas*, is given in Wilks (1977c).

The completed representation of a text is a structure made up of such word-sense formulas. At a level corresponding to the clause or simple sentence, formulas are arranged into triples, or *templates*, standing for an agent, an action, and an object; any of the three may itself be qualified by other formulas. For example, "Small men sometimes father big sons" would be structured as follows:



Here the bracketed English words should be imagined as being replaced by the semantic formulas representing their appropriate sense. Relationships among templates are indicated at a still higher level of structure.

What is the status of the primitive vocabulary in Wilks's system? First, he argues, primitives are not essentially different from natural language words. A semantic description in terms of primitives is just a description in "a reduced micro-language, with all the normal weaknesses and vagueness of a natural language" (Wilks, 1977c). The justification for using a language of primitives, then, is that it provides "a useful organizing hypothesis . . . for an AI natural language system."

Second, Wilks claims that individual primitives have their meaning in the same way that English words do: neither by direct reference to things, nor by correspondence to non-linguistic psychological entities, but only by their function within the overall language.

Third, in light of the nature of primitives, there is no one correct vocabulary for a primitive language, any more than there is a correct vocabulary for English. The test of the adequacy of a particular set of primitives is an operational one: the success or failure of the linguistic computations that use it. As suggestive evidence that Wilks's own set of primitives will indeed turn out to be adequate, he observes that it is very similar to the eighty words that are most frequently used in definitions in Webster's dictionary.

Finally, there are some general considerations to be taken into account in choosing a set of primitives. Wilks (1977c) identifies the following properties as desirable:

1. **Finitude:** The number of primitives should be finite and should be smaller than the number of words whose meanings it is to encode.
2. **Comprehensiveness:** The set should be adequate to express and distinguish among the senses of the word set whose meanings it is to encode.
3. **Independence:** No primitive should be definable in terms of other primitives.
4. **Noncircularity:** No two primitives should be definable in terms of each other.
5. **Primitiveness:** No subset of the primitives should be replaceable by a smaller set.

A qualification should be noted concerning the property of comprehensiveness: the definition in primitives of a word-sense is not required to be exhaustive of meaning. Wilks cites "hammer," "mallet," and "axe" as terms among which a representation in primitives cannot be expected to distinguish. In addition, the definition of a term is not expected to say everything; Wilks distinguishes between word meanings, which definitions express, and facts about things. The definition of "water," for example, might say that water is a liquid substance, but not that water freezes into ice. Facts of the latter sort are expressed in Wilks's system as in common-sense inference rules, which are separate from the dictionary and are used only as a last resort in disambiguation.

Schank's Conceptual Dependency Theory

The Conceptual Dependency theory of Roger Schank, now of Yale University, has been under development since 1969. Its most distinctive feature, the attempt to provide a representation of all actions using a small number of primitives, was first introduced in 1972. (See articles Natural Language.F5 and Natural Language.F6 on Schank's natural language understanding systems.)

There are significant differences between the systems of Schank and Wilks, both in the general outline of their systems and in their views of primitives. Wilks's system, for example, is oriented toward the task of machine translation, whereas Conceptual Dependency theory makes broader claims. First, Schank emphasizes task independence. In fact, the theory has been used as the basis of programs that, among other things, can *paraphrase* an input text, *translate* it to another language, *draw inferences* from it, or *answer questions* about it. Second, the theory is offered not only as a basis computer programs that understand language, but also as an intuitive theory of human language processing.

Consistent with these claims, Schank holds that it is the business of an adequate representation of natural language utterances to capture their underlying conceptual structure. A first requirement is that the representation be unambiguous, even though the input may contain syntactic ambiguity, as in "I saw the Grand Canyon flying to New York," or semantic ambiguity, as in "The old man's glasses were filled with sherry." The speaker of an ambiguous sentence usually intends an unambiguous meaning, so the representation is expected to reflect only the most likely version of what was intended.

A second requirement is that the representation be unique--that is, that distinct sentences with the same conceptual content should have the same representation. Some examples of groups of sentences that are all represented the same way are

I want a book.
I want to get a book.
I want to have a book.

and

Don't let John out of the room.
Prevent John from getting out of the room.

The principle of uniqueness of representation has been characterized as the basic axiom of the system. It has also been identified as accounting for human abilities to paraphrase and translate text. The problem of paraphrase--"how sentences which were constructed

differently lexically could be identical in meaning"--is a major theme throughout Schank's work (Schank, 1975c).

To obtain unique, unambiguous representations of meaning, Schank's system relies principally on a set of eleven primitive ACTs (Schank, 1975a; Schank and Abelson, 1977):

Physical acts

PROPEL	apply a force to a physical object
MOVE	move a body part
INGEST	take something to the inside of an animate object
EXPEL	force something out from inside an animate object
GRASP	grasp an object physically

Acts characterized by resulting state changes

PTRANS	change the location of a physical object
ATRANS	change an abstract relationship, such as possession or ownership, with respect to an object

Acts used mainly as instruments for other acts

SPEAK	produce a sound
ATTEND	direct a sense organ toward a stimulus

Mental acts

MTRANS	transfer information
MBUILD	construct new information from old information

There are several other categories, or concept types, besides the primitive ACTs in the representational system. They are:

Picture Producers (PPs), which are physical objects. Some special cases included among the PPs are natural forces like wind and three postulated divisions of human memory: the Conceptual Processor, where conscious thought takes place; the Intermediate Memory; and the Long Term Memory.

Picture Aiders (PAs), which are attributes of objects.

Times.

Locations.

Action Aiders (AAs), which are attributes of ACTs.

Only a little work has been done on reducing these latter categories to a primitive set; see Russell (1972) and Lehnert (1978) on the analysis of PPs, and Schank (1975a) on the treatment of PAs as attribute-value pairs.

Detailed rules are provided for the ways that elements of these categories can be combined into representations of meaning. There are two basic kinds of combinations, or

conceptualizations. One involves an actor (a PP) doing a primitive ACT; the other involves an object (a PP) and a description of its state (a PA). Conceptualizations can be tied together by relationships of instrumentality or causation, among others.

The primitive elements that occur in conceptualizations are not words, according to Schank, but concepts; they reflect a level of thought underlying language rather than language itself. Consequently, representations of text in Conceptual Dependency are said to be language-free. The task of translation, then, becomes only one task among many; it is accomplished by parsing from one language into Conceptual Dependency, and then generating text in the second language from the Conceptual Dependency representation.

The notion of language-free primitive concepts requires explication. For Schank, as for Wilks, the justification for using primitives is functional. Schank differs from Wilks, however, in his choice of the sort of function to be optimized, as well as in his view of primitives as language-free and psychologically plausible. Schank particularly emphasizes the computational advantages, to both programs and people, of storing propositions in a canonical form (Schank, 1975b). This requires, in Schank's view, that information implicit in a sentence be made explicit (Schank, 1975a; Schank and Abelson, 1977). Obtaining the implicit information in turn requires inferencing, and it is as an aid to making inferences that the use of primitives receives its most important justification:

Rather than stating that if you see something then you know it and if you hear something then you know it and if you read something then you know it and so on, we simply state that whenever an MTRANS exists, a likely inference is that the MTRANSed information is in the mental location LTM [Long Term Memory] (our representation for "know"). This is a tremendous savings of time and space. (Schank, 1975b)

Each primitive ACT, then, entails its own set of inferences. As a fuller example, the following are the main inferences from the fact that X PTRANSed Y from W to Z:

- 1) Y is now located at Z.
- 2) Y is no longer at location W.
- 3) If $Z = X$, or Z is human and requested the PTRANS, then Z will probably do whatever one ordinarily does with Y. Moreover, Z probably will become pleased by doing this. (Schank, 1975a)

Such inferences provide both the criterion for choosing a set of primitives and the definition of what primitives are. The primitive ACTs, Schank and Abelson (1977) state, are no more than the sets of inferences to which they give rise. Moreover:

The theoretical decision for what constitutes a primitive ACT is based on whether the proposed new ACT carries with it a set of inferences not already accounted for by an ACT that is already in use. Similarly, a primitive ACT is dropped when we find that its inferences are already handled by another ACT. (Schank, 1975b)

In his earlier work, Schank (1973a) claimed that the primitive ACTs of Conceptual

Dependency, together with some set of possible states of objects, were sufficient to represent the meaning of any English verb. It soon became clear, however, that additional mechanisms would be needed for a general-purpose language-understanding system. For example, there are problems of quantification and of metaphor, which have not yet been addressed (Schank and Abelson, 1977). There are problems raised by the fact that natural-language communications often presuppose a great deal of background knowledge, some of which has to do with the typical course of events in commonplace situations like eating in a restaurant or taking a bus (see article C7 on *frame* systems). Finally, of particular importance with respect to the use of primitives, there are problems arising from the fact that Conceptual Dependency generally expresses the meaning of an action *verbe* only in terms of its physical realization. One example is the reduction of "kiss" to "MOVE lips to lips" (Schank and Abelson, 1977). The inadequacy of this representation becomes especially apparent in light of the claim that no information is lost by the use of primitive ACTs to represent actions.

Recently Schank has added several new devices to his representational system to reflect the purposive aspects of actions as well as their physical descriptions. These include *goals*, which can be realized by appropriate sequences of acts; *scripts*, which provide such sequences in simple stereotyped situations; *plans*, which provide a more flexible way of specifying the appropriate action sequences, including the treatment of a whole set of alternative subsequences as a single subgoal; and *themes*, which include people's occupations (e.g., lawyer), their relationships with others (e.g., love), and their general aims (e.g., getting rich), and which are offered as the source of their goals. The representation of a piece of text is thus extended to try to take into account not only what caused what but also what was intended to cause what and why the actor might have had such an intention in the first place. In addition, Schank has recently supplemented the primitive ACTs with several social ACTs--AUTHORIZE, ORDER, DISPUTE, and PETITION--in order to represent yet another dimension of human actions more readily. None of these devices, however, is characterized as introducing a new set of primitives.

References

The best descriptions of the two systems using semantic primitives in AI discussed here are Wilks (1975a) and Schank and Abelson (1977). Norman and Rumelhart (1975) describe a computer model of human memory, MEMOD, and discuss the psychological basis of the semantic primitives used in their model (see also article Information Processing Psychology.B4). See Wilks (1977c), Wilks (1977a), and Winograd (1978) for further discussions of the semantic primitives issue.

C7. Frames and Scripts

There is abundant psychological evidence that people use a large, well-coordinated body of knowledge based on previous experiences to actively interpret new situations in their everyday cognitive activity (Bartlett, 1932). For example, when we visit a restaurant where we have never been before, we have an extensive array of *expectations* based on experience in other restaurants about what we will find: menus, tables, waiters, etc. In addition to these expectations about the *objects* in a typical restaurant, we have strong expectations about the *sequences of events* that are likely to take place. Representing knowledge about the objects and events *typical* to specific situations has been the focus of the AI knowledge representation ideas called *frames* and *scripts*. Frames were originally proposed by Minsky (1975) as a basis for understanding visual perception, natural language dialogues, and other complex behaviors. *Scripts*, frame-like structures specifically designed for representing sequences of events, have been developed by Schank and Abelson (1977) and their colleagues. Both refer to methods of *organizing* the knowledge representation in a way that directs attention and facilitates recall and inference.

Organizing Knowledge and Expectations

Frames provide a structure, a framework, in which new data is interpreted in terms of concepts acquired through previous experience. Furthermore, the organization of this knowledge facilitates *expectation-driven processing*, looking for things that are expected based on the context one thinks one is in. The representational mechanism that enables this kind of reasoning is the *slot*, the place where knowledge fits within the larger context created by the frame. For example, a simple frame for the generic concept of chair might have slots for number of legs, style of back, etc. A frame for a particular chair has the same slots, they are *inherited* from the CHAIR frame, but the contents of the slots are more fully specified:

CHAIR Frame

Specialization-of: FURNITURE
 Number-of-legs: an integer (DEFAULT=4)
 Style-of-back: straight, cushioned, ...
 Number-of-arms: 0, 1, or 2

JOHN'S-CHAIR Frame

Specialization-of: CHAIR
 Number-of-legs: 4
 Style-of-back: cushioned
 Number-of-arms: 0

By supplying a place for knowledge, and thus creating the possibility of missing or incompletely specified knowledge, the slot mechanism enables reasoning based on seeking confirmation of expectations--"filling in slots."

To illustrate some of the current ideas about slots and frames and how they might be used by a frame-based reasoning system, consider the following example of a Restaurant Frame. The terminology used for this example is intended only to give the flavor of the structure of knowledge in a frame, and does not follow any of the varied formalisms

developed by the different researchers in this area (e.g., Minsky, 1976; Bobrow and Winograd, 1977a; Schank and Abelson, 1977; Szolovitz, Hawkinson, and Martin, 1977; Goldstein and Roberts, 1977; Brachman, 1978; Aikins, 1979; Stefik, 1980).

Generic RESTAURANT Frame

Specialization-of: Business-Establishment

Types:

range: (Cafeteria, Seat-Yourself, Wait-To-Be-Seated)
 default: Wait-to-be-Seated
 if-needed: IF plastic-orange-counter THEN Fast-Food,
 IF stack-of-trays THEN Cafeteria,
 IF wait-for-waitress-sign or reservations-made
 THEN Wait-To-Be-Seated,
 OTHERWISE Seat-Yourself.

Location:

range: an ADDRESS
 if-needed: (Look at the MENU)

Name:

if-needed: (Look at the MENU)

Food-Style:

range: (Burgers, Chinese, American, Seafood, French)
 default: American
 if-added: (Update Alternatives of Restaurant)

Times-of-Operation:

range: a Time-of-Day
 default: open evenings except Mondays

Payment-Form:

range: (Cash, CreditCard, Check, Washing-Dishes-Script)

Event-Sequence:

default: Eat-at-Restaurant Script

Alternatives:

range: all restaurants with same FoodStyle
 if-needed: (Find all Restaurants with the same FoodStyle)

There are several different kinds of knowledge represented in this example. The "Specialization-of" slot is used to establish a *property inheritance hierarchy* among the frames, which in turn allows information about the parent frame to be *inherited* by its children, much like the ISA link in *semantic net* representations (see article C3). Note that the "Location" slot has sub-slots of its own--slots can have complex, frame-like structure in some systems. The contents of the "Range" slot in this generic restaurant example is an *expectation* about what kinds of things the Location of a restaurant might be. And the "If-Needed" slot contains an *attached procedure* which can be used to determine the slot's value if necessary (see discussion of procedural attachment below). Another important slot type is "Default," which suggests a value for the slot unless there is contradictory evidence. Many other types of slots are used in the various frame systems, and the descriptions here are quite incomplete, but they will at least give an idea of the kinds of knowledge represented in frames.

As indicated in the "Event-Sequence" slot, knowledge about what typically happens at a restaurant might be represented in a script, like the one below:

EAT-AT-RESTAURANT Script

Props: (Restaurant, Money, Food, Menu, Tables, Chairs)
 Roles: (Hungry-Persons, Wait-Persons, Chef-Persons)
 Point-of-View: Hungry-Persons
 Time-of-Occurrence: (Times-of-Operation of Restaurant)
 Place-of-Occurrence: (Location of Restaurant)
 Event-Sequence:
 first: Enter-Restaurant Script
 then: if (Wait-To-Be-Seated-Sign or Reservations)
 then Get-Maitre-d's-Attention Script
 then: Please-Be-Seated Script
 then: Order-Food-Script
 then: Eat-Food-Script unless (Long-Wait) when
 Exit-Restaurant-Angry Script
 then: if (Food-Quality was better than Palatable)
 then Complements-To-The-Chef Script
 then: Pay-For-It-Script
 finally: Leave-Restaurant Script

This is a rough rendition in English of the type of Restaurant script described by Schank and Abelson (1977). The script specifies a normal or default sequence of events as well as exceptions and possible error situations. The script also requires the use of a few static descriptions such as "Props" and "Roles" that refer to other frames. Scripts are described more fully in article Natural Language.F6.

Procedural Knowledge in Frames and Scripts

Underlying the *declarative* structure of frames and scripts--the way that they organize the representation of static facts--is an important dynamic or *procedural* aspect of frame-based systems. In particular, procedures can be *attached* to slots to drive the reasoning or problem-solving behavior of the system. (See the general discussion of procedural representation of knowledge in article C2). In some frame-based systems, attached procedures are the principal mechanism for *directing* the reasoning process, being activated to fill in slots "if-needed" or being "triggered" when a slot is filled (Bobrow, Kaplan, Kay, Norman, Thompson, and Winograd, 1977).

Filling in Slots. After a particular frame or script has been selected to represent the current context or situation, the primary process in a frame-based reasoning system is often just filling in the details called for by its slots. For example, after selecting the Generic Restaurant Frame above, one of the first things we might want the system to do is to determine the value of the "Type" slot. This could be accomplished in one of several ways. Sometimes the type is directly inherited, but in this case there are several alternatives. For instance, the *default* restaurant type can be used if there are no contraindications, or the attached "If-Needed" procedure can be used to decide.

Default and inherited values are relatively inexpensive methods of filling in slots; they don't require powerful reasoning processes. These methods account for a large part of the power of frames--any new frames interpreting the situation can make use of values determined by prior experience, without having to recompute them. When the needed

information must be derived, attached procedures provide a means of specifying appropriate methods that can take advantage of the current context, *slot-specific heuristics*. In other words, general problem-solving methods can be augmented by domain-specific knowledge about how to accomplish specific, slot-sized goals.

Besides directing the gathering of further information, filling in the slots provides confirmation that the frame or script is appropriate for understanding the scene or event. For example, Schank's script-based story understander, SAM, can be said to have understood a written story when each slot in the appropriate script has been filled by an event in the story, either explicit in the text or implied (see article *Natural Language.F6*). Should the frame or script be found to be inappropriate, attached procedures can "trigger" transfer of control to other frames.

Triggers. Another frequently used form of procedural attachment is routines that are activated when the value of a slot is found or changed. These "trigger" procedures implement *event or data-driven processing*, since they take over control only when certain events or data occur (see article C4). For example, the "If-Added" procedure in the "Food-Style" slot of the Generic Restaurant Frame is used to modify the list of alternative restaurants once a particular cuisine is chosen.

In some systems, trigger procedures attached to special slots in the frame are used to decide what to do in the event that the frame is found not to match the current situation. For instance, if the Eat-at-Restaurant script were to discover a food line and a stack of trays, it might trigger the Eat-at-Cafeteria script as being more appropriate. This triggering procedure has been used in various frame-driven systems for medical diagnosis (Szolovitz, Hawkinson, and Martin, 1977; Aikins, 1979). Since a number of related diseases might share a core set of signs and symptoms, the ability to make a differential diagnosis depends heavily on the ability to detect those factors that rule out or confirm a particular diagnosis. Typically, in medicine, when one diagnosis is ruled out, a similar but more likely disease is indicated.

Current Research on Frames and Scripts

A number of experimental prototype systems have been implemented to explore the idea of frame-based processing introduced by Minsky (1975). The following descriptions are intended to give an indication of the domains and problems researchers in this area address.

Bobrow, Kaplan, Kay, Norman, Thompson, and Winograd (1977) have experimented with frame-based natural language understanding in their GUS system, and their article includes clear examples of how frames might be used to control a system's reasoning. Designed as a prototype automated airline reservation assistant, the system attempted to demonstrate how various aspects of dialogue understanding--such as handling mixed-initiative dialogues, indirect answers, and anaphoric references--could be facilitated by the ability to provide expectations and defaults available with frames. This system was also used to explore procedural attachment issues.

Concurrently with the design of GUS, a frame-based programming/representation language called KRL (Knowledge Representation Language) was developed to explore frame-based processing (Bobrow and Winograd, 1977a). Many of the specific ideas about how

frame-based systems might work were first suggested by the KRL research group. As part of their early design work, the KRL group implemented several AI systems in the first version of the language (Bobrow and Winograd, 1977b). The report details a number of difficulties and shortcomings encountered, some of which are inherent in frame-based processing.

Other work with frame-based systems includes the NUDGE system, developed by Goldstein and Roberts (1977), which was used to understand incomplete and possibly inconsistent management-scheduling requests and to provide a complete specification for a conventional scheduling algorithm. Implemented in their FRL-O language, the system also used a frame-based semantics to resolve anaphoric requests (Bullwinkle, 1977).

A program that solves physics problems stated in English was developed by Novak (1977). It used a set of canonical object frames, such as Point, Mass, and Pivot, to interpret the actual objects and their relations in a number of statics problems. These canonical object frames were used to construct a view of an actual object as an abstract object, thereby simplifying the problem representation.

The UNITS package, developed by Stefik (1980), is a useful implementation of a variety of ideas about frame systems in an exportable programming package. The UNITS package has been used to build working AI systems for scientific applications. Finally, work on KLONE (Brachman, 1978) represents current research in the theory and design of frame-based systems.

Work on script-based processing in AI has for the most part been carried on by Schank and Abelson (1977) and their colleagues. They have used scripts to investigate the notions of causality and the understanding of sequences of events. In particular, the SAM program (article Natural Language.F6) attempts to understand short stories using a script to guide the interpretation of occurrences in the story. After establishing the appropriate script and filling some of its slots with information from the story, SAM can make inferences from script-based information about similar events.

Summary

Frames and scripts are recent attempts by AI researchers to provide a method for organizing the large amounts of knowledge needed to perform cognitive tasks. Much of the work in this area is quite conjectural, and there are many fundamental differences in approach among the researchers who have designed frame-based systems. The development of large-scale organizations of knowledge and the concomitant ability of these structures to provide direction for active cognitive processing is the current direction of AI research in knowledge representation. A number of serious problems must be solved before the conjectured benefits of frames will be realized.

References

Minsky (1975) coined the word "frame" and set off the recent flurry of AI work in the area. The clearest, detailed descriptions of frame-based reasoning systems are Kuipers (1975) and Bobrow and Winograd (1977a). Fahlman (1975), Charniak (1978), and Brachman (1978) discuss some important current research issues in this area. Schank and Abelson (1977) is an excellent discussion of AI research on scripts.

References

- Aikins, J. S. 1979. Prototypes and production rules: An approach to knowledge representation for hypothesis formation. *IJCAI 6*, 1-3.
- Anderson, J. 1976. *Language, memory, and thought*. Hillsdale, N.J.: Lawrence Erlbaum.
- Anderson, J., and Bower, G. 1973. *Human associative memory*. Washington, D.C.: Winston.
- Anderson, J., Kline, P., and Beasley, C. 1979. A general learning theory and its application to schema abstraction. In G.H. Bower (Ed.), *The psychology of learning and motivation*, Volume 13, New York: Academic Press, 277-318.
- Baker, R. 1973. A spatially-oriented information processor which simulates the motions of rigid objects. *Artificial Intelligence*, 4:29-40.
- Barstow, D. R. 1979. *Knowledge-based program construction*. New York: North Holland.
- Bartlett, F. C. 1932. *Remembering*. Cambridge, England: Cambridge University Press. Reprinted in 1977.
- Bernstein, M. I. 1977. Knowledge-based systems: A tutorial. Systems Development Corp., Santa Monica, Calif., Report No. TM-(L)-5903/000/00A.
- Bobrow, D. G. 1975. Dimensions of representation. In Bobrow and Collins, 1-34.
- Bobrow, D. G., and Collins, A. (Eds.) 1975. *Representation and understanding: Studies in cognitive science*. New York: Academic Press.
- Bobrow, D. G., Kaplan, R. M., Kay, M., Norman, D. A., Thompson, H., and Winograd T. 1977. GUS, a frame-driven dialog system. *Artificial Intelligence*, 8:155-173.
- Bobrow, D. G., and Winograd, T. 1977a. An overview of KRL, a knowledge representation language. *Cognitive Science*, 1(1):3-46.
- Bobrow, D. G., and Winograd, T. 1977b. Experience with KRL-0, one cycle of a knowledge representation language. *IJCAI 5*, 213-222.
- Boden, M. 1977. *Artificial intelligence and natural man*. New York: Basic Books.
- Brachman, R. J. 1978. A structural paradigm for representing knowledge. Bolt Beranek and Newman, Inc., Cambridge, Mass., Report No. 3605.
- Brachman, R. J. 1979. What's in a concept: Structural foundations for semantic networks. In Findler, 3-50.
- Brachman, R. J., and Smith, B. C. 1980. *SIGART Newsletter*, No. 70, Special issue on knowledge representation.

- Brooks, R. 1977. Production systems as control structures for programming languages. *SIGART Newsletter*, No. 63, 33-77.
- Bullwinkle, C. 1977. Levels of complexity in discourse for anaphora disambiguation and speech act interpretation. *IJCAI 5*, 43-49.
- Carbonell, J. R. 1970. AI in CAI: An artificial intelligence approach to computer-assisted instruction. *IEEE Transactions on Man-Machine Systems*, MMS-11, 190-202.
- Carbonell, J. R., and Collins, A. M. 1974. Natural semantics in AI. *IJCAI 3*, 344-351.
- Charniak, E. 1978. With spoon in hand this must be the Eating frame. *TINLAP 2*, 187-193.
- Davies, D. J. M. 1972. Popler: A POP-2 Planner. School of AI, University of Edinburgh, Report MIP-89.
- Davis, R. 1980. The application of meta-level knowledge to the construction, maintenance, and use of large knowledge bases. In Davis and Lenat.
- Davis, R., and Buchanan, B. G. 1977. Meta-level knowledge: Overview and applications. *IJCAI 5*, 920-927.
- Davis, R., Buchanan, B. G., and Shortliffe, E. H. 1977. Production rules as a representation for a knowledge-based consultation system. *Artificial Intelligence*, 8:15-45.
- Davis, R., and King, J. J. 1977. An overview of production systems. In E. Elcock and D. Michie (Eds.), *Machine Intelligence 8*, Chichester: Ellis Horwood, 300-332.
- Davis, R., and Lenat, D. B. 1980. *Knowledge-based systems in artificial intelligence*. New York: McGraw-Hill.
- Duda, R. O., Hart, P. E., Nilsson, N. J., and Sutherland, G. L. 1978. Semantic network representations in rule-based inference systems. In Waterman and Hayes-Roth, 203-221.
- Eastman, C. M. 1970. Representations for Space Planning. *CACM*, 13(4):242-250.
- Eastman, C. M. 1973. Automated Space Planning. *Artificial Intelligence*, 4:41-64.
- Erman, L. D., and Lesser, V. R. 1975. A multi-level organization for problem solving using many diverse, cooperating sources of knowledge. *IJCAI 4*, 483-490.
- Fahlman, S. E. 1974. A planning system for robot construction tasks. *Artificial Intelligence*, 5:1-49.
- Fahlman, S. E. 1975. Symbol-mapping and frames. *SIGART Newsletter*, No. 53, 7-8.
- Feigenbaum, E. A. 1977. The art of artificial intelligence: Themes and case studies of knowledge engineering. *IJCAI 5*, 1014-1029.

- Feigenbaum, E. A., and Feldman, J. (Eds.) 1963. *Computers and thought*. New York: McGraw-Hill.
- Fikes, R., Hart, P. E., and Nilsson, N. J. 1972. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251-288.
- Fikes, R., and Hendrix, G. 1977. A Network-based knowledge representation and its natural deduction system. *IJCAI 5*, 235-246.
- Filman, R. E. 1979. The interaction of observation and inference in a formal representation system. *IJCAI 6*, 269-274.
- Filman, R. E., and Weyhrauch, R. W. 1976. An FOL primer. AI Lab, Stanford University, Memo 288.
- Findler, N. V. (Ed.) 1979. *Associative networks: The representation and use of knowledge by computers*. New York: Academic Press.
- Flavell, J. 1979. Metacognition and cognitive monitoring: A new area for cognitive-developmental inquiry. *American Psychologist*, 34(10):906-911.
- Forgy, C. and McDermott, J. 1977. OPS, a domain-independent production system language. *IJCAI 5*, 933-939.
- Funt, B. V. 1976. WHISPER: A computer implementation using analogs in reasoning. Computer Science Dept., University of British Columbia, Report No. 76-09.
- Funt, B. V. 1977. WHISPER: A problem-solving system utilizing diagrams and a parallel processing retina. *IJCAI 5*, 459-464.
- Gelernter, H. 1963. Realization of a geometry-theorem proving machine. In Feigenbaum and Feldman, 134-152.
- Gentner, D., and Collins, A. M. 1980. Knowing about knowing: Effects of meta-knowledge on inference. *Memory and Cognition*, in press.
- Goldstein, I. P., and Roberts, R. B. 1977. NUDGE, a knowledge-based scheduling program. *IJCAI 5*, 257-263.
- Green, C. C. 1969. The application of theorem-proving to question-answering systems. *IJCAI 1*, 219-237.
- Hayes, P. J. 1973. Computation and deduction. *Symposium on mathematical foundations of computer science*, Czechoslovakia Academy of Science.
- Hayes, P. J. 1974. Some problems and non-problems in representation theory. *British computer society, AI and simulation of behavior group summer conference*, University of Sussex, 63-79.
- Hayes, P. J. 1977a. In defence of logic. *IJCAI 5*, 559-565.

- Hayes, P. J. 1977b. On semantic nets, frames, and associations. *IJCAI 5*, 99-107.
- Hedrick, C. 1976. Learning production systems from examples. *Artificial Intelligence*, 7:21-49.
- Hendrix, G. 1976. Expanding the utility of semantic networks through partitioning. *Artificial Intelligence*, 7:21-49.
- Hewitt, C. 1972. Description and theoretical analysis (using schemata) of PLANNER, a language for proving theorems and manipulating models in a robot. AI Lab, MIT, TR-258.
- Hewitt, C. 1975. How to use what you know. *IJCAI 4*, 189-198.
- Jackendoff, R. 1975. A system of semantic primitives. *TINLAP 1*, 28-33.
- Jackendoff, R. 1976. Toward an explanatory semantic representation. *Linguistic Inquiry*, 7:89-150.
- Kowalski, R. 1974. Predicate logic as a programming language. *IFIP 74*. Amsterdam: North-Holland, 569-574.
- Kuipers, B. 1975. A frame for frames: Representing knowledge for recognition. In Bobrow and Collins, 151-184.
- Lehnert, W. C. 1978. *The process of question answering: A computer simulation of cognition*. Hillsdale, N.J.: Lawrence Erlbaum.
- Lenat, D. B. 1980. AM: An AI approach to discovery in mathematics. In Davis and Lenat.
- Lenat, D. B., and McDermott, J. 1977. Less than general production system architectures. *IJCAI 5*, 928-932.
- Manna, Z. 1973. *Introduction to the mathematical theory of computation*. New York: McGraw-Hill.
- McCarthy, J. 1977. Epistemological problems of artificial intelligence. *IJCAI 5*, 1038-1044.
- McCarthy, J., and Hayes, P. J. 1969. Some philosophical problems from the standpoint of artificial intelligence. In D. Michie and B. Meltzer (Eds.), *Machine Intelligence 4*, Edinburgh: Edinburgh University Press, 463-502.
- McDermott, D. 1976. Artificial intelligence meets natural stupidity. *SIGART Newsletter*, No. 57, 4-9.
- McDermott, D., and Doyle, J. 1980. Non-monotonic logic - I. *Artificial Intelligence*, 13, in press.
- McDermott, J., and Forgy, C. 1978. Production system conflict resolution strategies. In Waterman and Hayes-Roth, 177-199.
- McDermott, J., Newell, A., and Moore, J. 1978. The efficiency of certain production system implementations. In Waterman and Hayes-Roth, 155-176.

- Miller, G. A. 1975. Comments on lexical analysis. *TINLAP 1*, 34-37.
- Miller, G. A., and Johnson-Laird, P. N. 1976. *Language and perception*. Cambridge, Mass.: Harvard University Press.
- Minsky, M. (Ed.) 1968. *Semantic information processing*. Cambridge, Mass.: MIT Press.
- Minsky, M. 1975. A framework for representing knowledge. In P. Winston (Ed.), *The psychology of computer vision*. New York: McGraw-Hill, 211-277.
- Moore, R. C. 1975. Reasoning from incomplete knowledge in a procedural deductive system. MIT AI Lab, TR-347.
- Myopolous, J., Cohen, P., Borgida, A., and Sugar, L. 1975. Semantic networks and the generation of context. *IJCAI 4*, 134-142.
- Newell, A. 1973. Production systems: Models of control structure. In W. Chase (Ed.), *Visual information processing*. New York: Academic Press, 463-526.
- Newell, A., and Simon, H. 1972. *Human problem solving*. Englewood Cliffs, N.J.: Prentice-Hall.
- Nilsson, N. J. 1971. *Problem-solving methods in artificial intelligence*. New York: McGraw-Hill.
- Norman, D. A., Rumelhart, D. E., and the LNR Research Group. 1975. *Explorations in cognition*. San Francisco: Freeman.
- Novak, G. S. 1977. Representation of knowledge in a program for solving physics problems. *IJCAI 5*, 286-291.
- Post, E. 1943. Formal reductions of the general combinatorial problem. *American Journal of Mathematics*, 65:197-268.
- Prawitz, D. 1965. *Natural deduction, a proof-theoretical study*. Stockholm: Almqvist and Wiksell.
- Pylyshyn, Z. 1973. What the mind's eye tells the mind's brain: a critique of mental imagery. *Psychological Bulletin*, 13:1-24.
- Pylyshyn, Z. 1975. Do we need images and analogs? *TINLAP 1*, 174-177.
- Pylyshyn, Z. 1976. Imagery and artificial intelligence. In C. W. Savage (Ed.), *Perception and cognition: Issues in the foundation of psychology*, University of Minnesota Press.
- Quillian, M. R. 1968. Semantic memory. In Minsky, 227-270.
- Quillian, M. R. 1969. The teachable language comprehender: A simulation program and the theory of language. *CACM*, 12(8):459-476.
- Raphael, B. 1968. SIR: Semantic information retrieval. In Minsky, 33-145.

- Reboh, R., Sacerdoti, E., Fikes, E. R., Sagalowicz, D., Waldinger, R. J., and Wilber, M. 1976. QLISP: A language for the interactive development of complex systems. AI Center, SRI International, Inc., TN-120.
- Reiter, R. 1978. On reasoning by default. *TINLAP* 2, 210-218.
- Riesbeck, C. K. 1975. Conceptual analysis. In Schank, 83-156.
- Rulisfon, J., Derkson, J. A., and Waldinger, R. J. 1972. QA4: A procedural calculus for intuitive reasoning. AI Center, SRI International, Inc., TN-83.
- Russell, S. W. 1972. Semantic categories of nominals for conceptual dependency analysis of natural language. AI Lab, Stanford University, Memo 172.
- Rychener, M. D. 1976. Production systems as a programming language for artificial intelligence applications. Computer Science Dept., Carnegie-Mellon Univ.
- Rychener, M. D. 1977. Control requirements for the design of production system architectures. *ACM symposium on artificial intelligence and programming languages*, Rochester, N.Y., 37-44.
- Schank, R. C. 1972. Conceptual dependency: A theory of natural language understanding. *Cognitive Psychology*, 3:552-631.
- Schank, R. C. 1973a. The fourteen primitive actions and their inferences. AI Lab, Stanford University, Memo 183.
- Schank, R. C. 1973b. Identification of conceptualizations underlying natural language. In Schank and Colby, 187-247.
- Schank, R. C. 1975a. *Conceptual information processing*. New York: North Holland.
- Schank, R. C. 1975b. The primitive ACTs of conceptual dependency. *TINLAP* 1, 38-41.
- Schank, R. C. 1975c. The structure of episodes in memory. In Bobrow and Collins, 237-272.
- Schank, R. C., and Abelson, R. P. 1977. *Scripts, plans, goals, and understanding*. Hillsdale, N.J.: Lawrence Erlbaum.
- Schank, R. C., and Colby, K. M. (Eds.) 1973. *Computer models of thought and language*. San Francisco: W. H. Freeman.
- Schubert, L. K. 1975. Extending the expressive power of semantic networks. *IJCAI* 4, 158-164.
- Shortliffe, E. H. 1976. *Computer-based medical consultations: MYCIN*. New York: North Holland.
- Simmons, R. F. 1973. Semantic networks: Their computation and use for understanding English sentences. In Schank and Colby, 63-113.

- Simmons, R. F., and Slocum, J. 1972. Generating English discourse from semantic nets. *CACM*, 15(10):891-905.
- Simon, H. A. 1969. *The sciences of the artificial*. Cambridge, Mass.: MIT Press.
- Sloman, A. 1971. Interactions between philosophy and AI: The role of intuition and non-logical reasoning in intelligence. *Artificial Intelligence*, 2:209-225.
- Sloman, A. 1975. Afterthoughts on analogical representations. *TINLAP 1*, 178-182.
- Stefik, M. 1980. Planning with constraints. Computer Science Dept., Stanford University, Report No. 784.
- Suppes, P. 1957. *Introduction to logic*. New York: Van Nostrand Reinhold.
- Sussman, G., and McDermott, D. V. 1972. CONNIVER reference manual. MIT AI Lab, Memo 259.
- Sussman, G., Winograd, T., and Charniak, E. 1970. Micro-PLANNER reference manual. AI Lab, MIT, Memo 203.
- Szolovitz, P., Hawkinson, L. B., and Martin, W. A. 1977. An overview of OWL, a language for knowledge representation. Laboratory for Computer Science, MIT, TM-86.
- Vere, S. A. 1977. Relational production systems. *Artificial Intelligence*, 8:47-68.
- Walker, D. E. (Ed.) 1976. *Speech understanding research*. New York: North Holland.
- Waterman, D. A. 1970. Generalization learning techniques for automating the learning of heuristics. *Artificial Intelligence*, 1:121-170.
- Waterman, D. A., and Hayes-Roth, F. (Eds.) 1978. *Pattern-directed inference systems*. New York: Academic Press.
- Weyhrauch, R. W. 1978. Prolegomena to a theory of mechanized formal reasoning. AI Lab, Stanford University, Memo 315.
- Wilks, Y. A. 1975a. A preferential, pattern-seeking semantics for natural language inference. *Artificial Intelligence*, 6:53-74.
- Wilks, Y. A. 1975b. Primitives and words. *TINLAP 2*, 42-45.
- Wilks, Y. A. 1977a. Good and bad arguments about semantic primitives. Dept. of Artificial Intelligence, University of Edinburgh, Research Report No. 42.
- Wilks, Y. A. 1977b. Knowledge structures and language boundaries. *IJCAI 3*, 151-157.
- Wilks, Y. A. 1977c. Methodological questions about artificial intelligence: Approaches to understanding natural language. *Journal of Pragmatics*, 1:69-84.
- Winograd, T. 1972. *Understanding natural language*. New York: Academic Press.

- Winograd, T. 1974. Five lectures on artificial intelligence. AI Lab, Stanford University, Memo 246.
- Winograd, T. 1975. Frame representation and the declarative/procedural controversy. In Bobrow and Collins, 185-210.
- Winograd, T. 1978. On primitives, prototypes, and other semantic anomalies. *TINLAP 2*, 25-32.
- Winograd, T. 1980a. Extended inference modes in computer reasoning systems. *Artificial Intelligence*, 13, in press.
- Winograd, T. 1980b. *Language as a cognitive process*. Reading, Mass.: Addison-Wesley, in preparation.
- Winston, P. H. 1977. *Artificial intelligence*. Reading, Mass.: Addison-Wesley.
- Woods, W. A. 1968. Procedural semantics for a question-answering machine. *Fall Joint Computer Conference*, 33(1):467-471.
- Woods, W. A. 1975. What's in a Link: Foundations for semantic networks. In Bobrow and Collins, 35-82.
- Woods, Bates, Brown, Bruce, Cook, Klovstad, Makhoul, Nash-Webber, Schwartz, Wolf, and Zue. 1976. *Speech understanding systems: Final report*. Bolt Beranek and Newman, Inc., Cambridge, Mass., Report No. 3438.

Index

- Abelson, R. 60, 64
ABSTRIPS 22
acquisition of knowledge 2
ACT 43
active structural network 35
adaptive production system 43
AI programming languages 25, 27
AM 11, 43-45
ambiguity 53-56
analogical reasoning 3
analogical representation 47
Anderson, J. 43
ATN grammar 35
axiomatic system 19
axiomatization 19
- backtracking 49
backward chaining 43, 45
Baker, R. 50
Barstow, D. 42
Bartlett, F. 60
blackboard 44
Bobrow, D. 13, 63
Brachman, R. 64
Bullwinkle, C. 64
- CAPS 44
Carbonell, J. 35
case frame 32, 35
chemistry applications 21
chess 51
Collins, A. 35
combinatorial explosion 8, 9, 21
completeness 29
computer-assisted instruction 35
conceptual dependency 56-59
conceptualization 57
conflict resolution 40, 44
CONNIVER 27, 28
consistency 29
constraint structured planning 49
context 39
context tree 44
control 26, 30, 42, 63
- data-driven processing 45, 63
Davis, R. 43, 46
declarative vs. procedural
representation 6, 62
declarative vs. procedural
representations 25
deduction 3, 51
default reasoning 28-29
default values 33, 60-63
DENDRAL 11, 46
denotation 47
depth-first search 49
diagram 48
dialogue 63
direct representation 11, 29, 47-52
directed reasoning 6, 26-29, 35, 37, 41,
62
discovery 43
discrimination net 12
domain-specific knowledge 6, 28, 63
Duda, R. 44
- Eastman, C. 48
educational applications 35
elimination rule 17, 18, 22
EPAM 12, 44
epistemology 7, 8, 23
Erman, L. 44
event-driven processing 45, 63
expectation-driven processing 33, 45, 60-
61
explanation 43, 46
explicit vs. implicit knowledge 6, 25
extended inference modes 28
- Feigenbaum, E. A. 46
Fikes, R. 22, 37
Filman, R. 22, 51

- first-order logic 18, 19
- focus of attention 39, 44
- FOL 22, 23
- formal reasoning 3
- forward chaining 45
- frame problem 29, 47
- frame systems 10
- frames 10, 13, 23, 33, 35, 38, 54, 58, 60-64
- Fregean representations 47
- FRL-0 64
- functions 19
- Funt, B. 49

- game playing 8
- Gelernter, H. 48
- General Space Planner 48
- Geometry Theorem Prover 48
- goal-driven processing 45
- Goldstein, I. 64
- GOLUX 23, 27
- GPS 22, 44
- grain size 4
- Green, C. C. 21
- GUS 63

- HAM 35
- Hart, P. 44
- Hayes, P. 23, 27, 29
- HEARSAY 44-45
- Hedrick, C. 43
- Hendrix, G. 35, 37
- heuristic 6, 22, 27, 29, 38, 63
- Hewitt, C. 27
- homomorphism 47
- human memory 31, 36, 47

- inexact reasoning 43
- inference 3, 37, 50, 56, 58
- inference rules 17
- inheritance hierarchy 10, 31, 61

- instance 32
- interactive transfer of expertise 46
- interpreter 39
- introduction rule 17, 18, 22
- issues in knowledge representation 2-7

- KLONE 64
- knowledge 2
- knowledge acquisition 3, 42, 43, 46
- knowledge engineering 46
- Kowalski, R. 27
- KRL 13, 63

- learning 3, 11, 42, 43
- legal move generators 8
- Lenat, D. 43
- Lesser, V. 44
- LISP 26
- logic 3, 4, 6, 9, 15-24, 25, 26, 30, 32, 36, 47

- machine translation 53-58
- manageability 41, 46
- MARGIE 56
- matching 13, 36
- mathematical applications 43
- McCarthy, J. 23, 29
- McDermott, D. 27
- McDermott, J. 44
- means-ends analysis 22
- medical applications 43, 63
- MEMOD 59
- meta-knowledge 2
- Minsky, M. 13, 60, 63
- models 51
- modular 11
- modular representation 23
- modularity 5, 11, 30, 41, 46
- modus ponens 17
- Moore, R. 29, 30
- MYCIN 6, 11, 43-46

- natural deduction 17, 18, 22, 27
 natural language understanding 12, 35, 53-59, 63, 64
 network fragment 36
 network partitioning 35
 Newell, A. 11
 Norman, D. 59
 Novak, G. 64
 NUDGE 64
- observation 51
- parallelism 51
 paraphrase 12, 56
 parsing 26, 35
 perceptual primitives 50
 PLANNER 6, 10, 23, 27-29
 planning 22, 49
 plausible reasoning 29
 POPLER 28
 predicate 17, 32
 predicate calculus 9, 17, 47
 preference semantics 54
 problem reduction 48
 problem solving 8
 problem space 43
 procedural attachment 10, 13, 30, 61, 62, 63
 procedural knowledge 42, 46, 62
 procedural representation 3, 5, 10, 23, 25-30, 62
 production rules 11, 39
 production system 6, 11, 39-46
 property inheritance 10, 31-34, 60, 61
 propositional calculus 15-17
 propositional representation 47
 PROSPECTOR 11, 31, 44, 46
 pruning heuristic 48
 psychology 11, 31, 41, 47
- QA4 28
 QLISP 28
 quantification 7
 quantifier 18
 question answering 21, 22, 25, 35, 56
 Quillian, R. 34
- Raphael, B. 25, 35
 reasoning 3
 representation of knowledge 1-64
 resolution method 21, 27
 retrieval 3
 Roberts, R. 64
 rulebase 39
 rules of inference 3, 9, 15, 17-18
 Rumelhart, D. 59
 Rychener, M. 44
- SAD-SAM 12
 SAM 56, 60, 63
 Schank, R. 56, 60, 63
 SCHOLAR 35
 scientific applications 64
 scope 4
 scripts 60-64
 semantic interpretation function 47
 semantic net 10, 25, 31-38, 42, 44, 54, 61
 semantic primitives 5, 12, 33, 45, 53-59
 semantics 34, 36, 38
 sentential connectives 15
 short-term memory 39
 Shortliffe, E. 43
 SHRDLU 6, 28, 44
 Simmons, R. F. 32, 35
 Simon, H. 11
 SIR 12, 25, 35
 Slocum, J. 32
 slot 13, 60
 SNIFFER 37
 sort 18, 20
 space planning 48
- 22

speech understanding 12, 35
spreading activation 35, 36, 38
state-space search 8, 43
Stefik, M. 64
STRIPS 22
STUDENT 44
Sussman, G. 27
syntax 9

tautology 16
taxonomy 31
Teachable Language Comprehender 35
TEIRESIAS 2, 43-46
theorem proving 6, 9, 21, 24, 27, 37
THNOT 28
top-down vs. bottom-up reasoning 45
TORUS 35
Tower of Hanoi puzzle 15, 19
transition operators 8
truth table 16
truth values 15

understandability 5, 10, 11, 26, 42
UNITS 64
universal specialization 18

variable 18
variable domain array 49
Vere, S. 43

Waterman, D. 43
well-formed formulae 18
Weyhrauch, R. 22, 23
WHISPER 49
Wilks, Y. 53
Winograd, T. 6, 13, 28, 63
Woods, W. 26