System
Development
Corporation

KNOWLEDGE-BASED SYSTEMS:
A TUTORIAL

30 JUNE 1977

TM-(L)-5903/000/00

System Development Corporation

KNOWLEDGE-BASED SYSTEMS: A TUTORIAL

J. A. BARNETT
M. I. BERNSTEIN

30 JUNE 1977

THE WORK REPORTED HEREIN WAS PERFORMED UNDER CONTRACT MDA 904-76-C-0343 FOR THE MARYLAND PROCUREMENT OFFICE.

TM-(L)-5903/000/00

MODIFICATION TO:

TM-(L)-5903/000/00, "Knowledge-Based Systems: A Tutorial," dated 30 June 1977

author's name
M. I. Bernstein

approval signature
[illegible]

System Development Corporation
2500 Colorado Avenue • Santa Monica, California 90406

CURRENT MODIFICATIONS

ERRATA

Listing is by page number and line. Positive line numbers count from top of the page, negative numbers from the bottom. "X ==: Y" means "X should read as Y."

2-3 +2	of the vehicle ==: of a vehicle
2-10 +2	method or operation ==: method of operation
2-14 -6	[NILSSON71] ==: [NILSSON71])
2-20 -11	whole line ==: ment institution system which provides interactive support to
3-5 +9	below under the ==: below with the
3-6 +6	set of four special ==: set of special
3-12 +2	48 QUESTION 48 ==: QUESTION 48
4-1 +8	systems. (KBS). ==: systems (KBS).
4-4 -6	a techniqal option ==: a technical option
4-5 +9	and plausible ==: and principles of plausible
4-6 +1	[FREUD 60 and 55] ==: [FREUD60, 55]
4-6 +3	[BARTLETT 32] ==: [BARTLETT32]
4-6 -3	[BOBROW 75b and 75c], [BROWN 75b], ==: [BOBROWD75b, 75c], [BROWN75b],
4-6 -2	whole line ==: [CHARNIAK75], [COLLINS76], [HAWKINSON75], [MOOREJ73], [SIROVICH72], and

4-6 -1	[WEISS 61] ==: [WEISS61]
4-9 +6	form which ==: from which
4-10 +5	intelligent perfromance ==: intelligent performance
4-10 +12	to product acceptable ==: to produce acceptable
4-11 +12	[SHORTLIFFE 76] ==: [SHORTLIFFE76]
4-11 +12	[ZADEH 75, ==: [ZADEH75,
4-11 +13	74, and 65] and [GOGUEN 68]. ==: 74, 65] and [GOGUEN68].
4-11 +14	[CARNAP 50], [HEMPEL 45], and [HARRE 70]. ==: [CARNAP50], [HEMPEL45], and [HARRE70].
4-11 +15	[TVERSKY 72] and [LUCE 65]. Also see [TÖRNEBOHM 66] ==: [TVERSKY72] and [LUCE65]. Also see [TÖRNEBOHM66]
4-11 -9	The first data ==: the first, data
4-12 +13	procedural,* ==: procedural*,
4-12 -1	[WINOGRAD 75] ==: [WINOGRAD75]
4-16 +4	not "how ==: not "How
4-18 -5	end *state* ==: *end state*
4-20 +9	power,* ==: power*,
4-20 -6	"pot empty," "satisfied," and "thirsty." ==: "pot empty", "satisfied", and "thirsty".
4-20 -1	[FISHER 70] ==: [FISHER70]
4-21 +10	and user to ==: and used to
4-21 -8	"A." ==: "A".
4-23 +11	[MINSKY 67] ==: [MINSKY67]
4-24 -12	effective program ==: effective procedure
4-25 -2	temperature, because ==: temperature because
4-29 -10	if a human ==: if human

4-31 +8	[BOBROW75d] ==: [BOBROWD75d]
4-31 +9	[BOBROW73] ==: [BOBROWD73]
4-31 +12	[R. MOORE75] ==: [MOORER75]
4-36	add at bottom of page $(\exists y)X \equiv \sim((\forall y)(\sim X))$
4-37 -6	predicate calculate ==: predicate calculus
4-39 +1	[P. KLAHR77] ==: [KLAHRP77]
4-40 +4	(4) $(\forall x_3)$ (x_4) ==: (4) $(\forall x_3)(\forall x_4)$
4-41 -5	the box is x_3 ==: the box if x_3
4-44 -11	At such ==: As such
4-45 +10	and W in ==: and ~W in
4-45 -2	[P. KLAHR77 ==: [KLAHRP77,
4-45 -1	and 75], ==: 75],
4-46 -2	7 to 12, ==: 7 to 12, and
4-48 +6	insert new lines <u>best match</u>--Execute the rule whose LHS is the best match to the workspace.
4-48 +8	[Post 36] ==: [POST36]
4-49 +2	$\#\$_1\#\$_2\Longrightarrow$ ==: $\#\$_1\#\$_2\#\Longrightarrow$
4-49 +3	$\#1\$_1\#\$_2\#\$_3\Longrightarrow\#\$_1\#\$_2\#\$_2\#\$_3\#$ ==: $\#1\$_1\#\$_2\#\$_3\#\Longrightarrow\#\$_1\#\$_2\#\$_2\$_3\#$
4-49 +4	$\#\#\$_1\#\$_2\Longrightarrow$ ==: $\#\#\$_1\#\$_2\#\Longrightarrow$
4-51 -5	|e1 ==: |a1
4-51 -3	|e2 ==: |a2
4-51 -2	exp[(* A (* B C))] ==: exp [(+ A (* B C))]
4-53 +5	service manager ==: service representative

4-55 +11	service manager (SvrM) ==: service representative (SrvR)
4-55 -13	service manager ==: service representative
4-58 +10	rules' derived conditions. ==: rules, "derived conditions".
4-60 +10	charge. This is an ==: charge, an
4-61 -10	discussion in showing ==: discussion by showing
4-62 -10	and procedures to ==: and programs to
4-65 -6	whole line ==: [CHOMSKY63], [DAVIS76, 75], [GALLER70], [HEDRICK76, 74], [McDERMOTTJ76a,
4-65 -5	whole line ==: 76b], [MINSKY67], [MOOREJ73], [NEWELL76a], [POST43, 36], and [VERE77].
4-65 -3	[WATERMAN75, 74 and 70] ==: [WATERMAN75, 74, 70]
4-65 -1	[DAVIS77 and 76], and [SHORTLIFFE76, 75a, 75b, and 73] ==: [DAVIS77, 76], and [SHORTLIFFE76, 75a, 75b, 73]
4-66 +1	72, and 69] ==: 72, 69]
4-66 +2	[MARTIN75] ==: [STEFIK77]
4-66 +3	[ANDERSON76a and 76b] ==: [ANDERSON76a, 76b]
4-66 +4	[RYCHENER76 ==: [RYCHENER76,
4-66 +5	and 75]. ==: 75].
4-67 -1	in the figure. ==: in the figure--the arrows connect the relation's arguments in their order of occurrence.
4-68	DOG $\longrightarrow$ BOWSER ==: DOG $\xrightarrow{\text{ISA}}$ BOWSER
4-69 +4	CAT3 ==: CAT
4-69 -10	WFFS ==: WFFs
4-72 +4	MARY's ==: MARY'S
4-72 +6	BOB's ==: BOB'S
4-73 +7	[MYLOPAULOS75a and 75b] ==: [MYLOPAULOS75a, 75b]

4-76 -8	lines. A general ==: lines: a general
4-78 +11	can become more ==: can be incorporated into one that is more
4-78 -3	[DBOBROW77a, 77b, and 75c] ==: [BOBROWD77a, 77b, 75c]
4-78 -2	(GOLDSTEIN76] ==: [GOLDSTEIN76]
4-79 +10	[KAHN 75] ==: [KAHN75]
4-84 +8	more knowledge ==: more independent knowledge
4-86 +4	[ERMAN 75] ==: [ERMAN75]
4-89 -3	performed are the ==: performed is the
4-90 +3	[NILSSON 71] ==: [NILSSON71]
4-95 -11	[HEWITT 72] ==: [HEWITT72]
4-95 -6	4.1.2.1 ==: 4.1.2.6
4-95 -5	[BOBROW 77a] ==: [BOBROWD77a]
4-95 -3	[WOODS 75] ==: [WOODS75]
4-95 -1	whole line ==: described in: [BARNETT75a], [BERNSTEIN76], [MOOREJ73] and [WOODS76].
4-96 -12	[J. MOORE 73] ==: [MOOREJ73]
4-97 -5	finding it. ==: finding it though these two costs are often related.
4-98 +2	some weaken ==: some weakened
4-104 +3	system backs up ==: system could back up
4-104 -6	graph was ==: graphs were
4-105 +11	[P. KLAHR77] ==: [KLAHRP77]
4-107 -4	if either ==: if either function
4-109 -6	has been proved ==: has been proven
4-110 +11	$f <$ old f ==: $\hat{f} <$ old $\hat{f}$

4-110 +12	f with new f ==: $\hat{f}$ with new $\hat{f}$
4-110 +14	f ==: $\hat{f}$
4-113 +10	[NILSSON71 and 69] ==: [NILSSON71, 69]
4-113 +13	an entity. ==: a domain.
4-114 +7	of the entity ==: of the domain
4-116 +9	statistical technique. ==: stochastic technique, e.g., Monte Carlo.
4-117 -2	[BOBROW75b] ==: [BOBROWD75b]
4-122 -11	about x?" ==: about X?"
4-122 -3	[Weizenbaum 66] ==: [WEIZENBAUM66]
4-123 +6	$\$_2$ ==: $\$_1$
4-123 +6	"Mary," ==: "Mary",
4-123 -12	[WOODS73 and 70] ==: [WOODS73, 70]
4-125 -12	network) help ==: network) to help
4-127 -6	becaise ==: because
4-127 -3	complaint depart ==: complaint department
4-130 -14	he employe, ==: he employs,
4-132 +7	[MALHOTRA76,75a and 75b] ==: [MALHOTRA76, 75a, 75b]
4-132 -6	true or systems ==: true of systems
4-134 +2	[McDERMOTT74] ==: [McDERMOTTD74b]
4-134 +4	[JMOORE74] ==: [MOOREJ74]
5-1 +3	where it is ==: where it is in common use and the techniques are commonly understood. And, though there is a fair body of literature on a variety of topics related to KBS technology, there is

5-2 +4	technology. On the ==: technology, unless
5-2 +5	other hand, the ==: the
5-2 -5	whole line ==: assure cooperation early in the planning process.
5-3 +9	Meta-DENDRAL ==: META-DENDRAL
5-9 -1	to implement. ==: to implement, though more difficult to modify.
5-13 +10	aggregates, such ==: aggregates such
5-14 +10	user's ==: users'
5-14 +16	user's ==: users'
7-2 +14	[MINSKY75] ==: [MINSKY75])
7-2 +15	(KRL) [BOBROWD77b]. ==: (KRL [BOBROWD77b]).
7-5 -8	Meta-DENDRAL ==: META-DENDRAL
7-5 -5	Meta-DENDRAL ==: META-DENDRAL
8-16 +2	Understanding systems, ==: Understanding understanding systems,
A-1 -13	[KLAHRD75] ==: [KLAHRD74]
A-3 -9	[BOBROWD77a]. ==: [BOBROWD77a]).
A-25 +15	Meta-DENDRAL ==: META-DENDRAL

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

LIST OF FIGURES

LIST OF FIGURES
(Continued)

LIST OF TABLES

KNOWLEDGE-BASED SYSTEMS: A TUTORIAL

1. INTRODUCTION

This report surveys recent and current work in interactive knowledge-based systems (KBS), defining and explaining KBS concepts and technology. Our purpose is to give those who may be interested in the application of such systems a means of assessing their present potential use. We explore both the capabilities and the limitations of KBS technology as they are observable and predictable in existing systems, and we provide references and a bibliography that will permit those who want to further explore the topic on their own.

Section 2 describes a hypothetical KBS application and defines KBS techniques as they are understood and implemented in existing systems. Section 3 contains a case study of MYCIN, which is, we believe, the best representative of KBS technology. In Section 4, we describe in detail and at length the techniques that are used to build KB systems, and in Section 5, we examine the applicability of existing and emerging KBS techniques and discuss what we believe are boundary conditions and limitations. Finally, we offer our conclusions and recommendations on how to best apply KBS technology and we suggest some directions for future KBS research. The annotated bibliography should provide readers with adequate references to the body of KBS literature. Two appendices are included, as well: one is our adaptation of a taxonomy of knowledge and cognitive skills [BLOOM56], and the other is a compilation of the current KBS research and development projects that sets forth the institution, the principal investigator, and a short description of each project.

2. KNOWLEDGE-BASED SYSTEMS

It is necessary to distinguish, at the outset, between knowledge-based systems and other computer-based systems that contain or incorporate knowledge. Almost all computer programs and systems contain knowledge of at least two kinds: knowledge about things and knowledge about what to do with things--that is, how to manipulate or transform them. But we define a knowledge-based system as one in which knowledge is collected in one or more compartments (called knowledge sources) and is of the kind that facilitates problem solving (reasoning) in a single, well-defined problem domain. Problems are solved by applying the kind of reasoning that is used by a practitioner in the domain in which the KBS is applied. Unlike generalized problem-solving systems, knowledge-based systems must accumulate large amounts of knowledge in specific domains and rely on domain-specific problem-solving techniques that can be developed to a high leve of expertise [DAVIS77].

In considering systems for inclusion in (or exclusion from) the category of "knowledge-based" systems, we have imposed some conditions that exclude systems that others have identified as being knowledge based. We have specifically excluded systems that are research prototypes that were not and are not intended to be put to productive use. Many of the systems that we consider to be outside the KBS domain, as we have defined it, do, however, embody technology that is incorporated in the KB systems we do include; in a sense, then, they are included, if only indirectly. None of the speech-understanding systems developed under the sponsorship of the Defense Advanced Research Projects Agency are included, for example, yet much of the technology they embody is incorporated in systems we do include, such as MOLGEN [STEFIK77]. This is a conservative approach, and if it errs, it errs in the proper direction, if only because it does not raise unwarranted expectations. We have made no attempt to define knowledge in other than operational (or utilitarian) terms. Readers who are interested in pursuing the philosophical aspects of the definitions of knowledge should consult Appendix A, which is a modification and summarization of [BLOOM56].

2.1 A HYPOTHETICAL KBS

The following is a brief description and example of a simple, hypothetical KBS application that illustrates most of the capabilities of KB systems. The application is one that should be familiar to most readers and would be feasible to build, although no such system presently exists.

The hypothetical system is an automotive service consultant whose primary purpose is to help ensure the best service at the least cost for automobiles brought to a service agency or garage. The system is to be used both by the service representative, who is the primary interface between the customers and the establishment, and by the mechanics who work on the cars. Though it functions as a simple data management system in routine circumstances, its value lies in its ability to diagnose subtle problems from symptoms and problems presented by the customer (to the service representative) or by the mechanic when he discovers a non-routine problem in performing service and repair. Its benefit is that it ensures a uniformly high level of service to customers and requires of mechanics only that they have good mechanical skills, but not necessarily good diagnostic skills, and of service representatives that they be able to listen carefully to customers' complaints. Both customers and the service agency benefit, because unnecessary repairs are eliminated and it is highly likely that what has been done corrects the problem the first time. Although the reasoning skills required by such a KB system are relatively simple, the amount of knowledge required (exclusive of the normal data base) is large because of the number and variety of automotive subsystems involved and the high degree of their interdependence.

The data base of the system would contain a description of each car serviced at the agency. The description would contain the configuration of the vehicle's engine, running gear, and optional and special equipment, as well as the year of manufacture, model, color, service history, and other pertinent data.

The system would contain two basic kinds of knowledge: (1) generic knowledge about the major elements of the vehicle (such as its cooling, ignition, and fuel systems) and (2) knowledge about the model and its specific options. Diagnostic knowledge would be necessary to relate symptoms to causes and to relate specific knowledge to generic knowledge. For instance, poor engine performance could result from mechanical problems (valves, rings) or from related systems (cooling, ignition, or carburetion).

The intent of the system is to recommend (1) the least expensive tests (where a specific remedy is not obvious) that will produce the greatest amount of specific information about the cause of the problem, and (2) the least expensive repairs for the problem. The system could also recommend the best post-repair testing to verify that the problem has been removed. The system would explain its recommendations or conclusions if asked.

The following is a hypothetical scenario of such a system in operation. The users are the service representative and the mechanic(s) performing the test and repair work. The user language is a stylized English designed to convey a maximum amount of information between the user and the system with a minimum amount of effort.

System: Tuesday, May 3, 1977, 08:32:44

Service rep: Next customer

System: Enter owner's name or vehicle ID number

Service rep: 1743 HBAQ 8197

System: Owner is S.B. Franklin, car is 1974 Ford Gran Torino 4-dr sedan with 351-2V engine, A/C, P/S, P/B and CA smog controls. Last service — April 4, 1977 at 28,412 mi R&R water pump and lower radiator hose, what is present mileage?

Service rep: 29285

System: Please inform owner that the following regular maintenance services are due:
1. Change oil and filter — last changed on Jan 13, 1977 at 24,842 mi.
May we be of other service?

Service rep: Battery not keeping charge, barely able to start car this a.m.

System: Any other symptoms or problems?

Service rep: No

System: Check belt tension on alternator — most likely cause.

Service rep: Why?

System: Last service required removal of fan and belts, very likely that tension was not properly set on reinstallation.

Service rep: More (meaning more things to be checked or tested)

System: Check fluid level in battery — next most likely cause — overcharged battery. If level is low, check alternator output and voltage regulator.

Service rep: Replace battery?

System: No, battery is 6 mos old and should take recharge.

Service rep: Thank you. (Signal to system that session is ended for now)

System: Car assigned number 287 in control system. First test to be performed by J. K. Uzzi in Bay 17.

After the car is moved to the mechanic's work area and the prescribed tests are performed, the following interchange takes place between the mechanic and the system:

Mechanic: JKU, 17, 287 (indicating mechanic, bay, and car being serviced)

System: 1974 Gran Torino, test results please.

Mechanic: Belt tension OK, battery level OK, charge low, alternator, VR output OK. Problem not found.

System: Test for above-normal current flow with ignition off. (Mechanic attaches ammeter and takes reading.)

Mechanic: Reading above normal

System: Check for faulty switch in hood light, trunk light, and courtesy lights.
Mechanic finds faulty switch in trunk light such that light never turns off.
Mechanic: Faulty trunk light switch R&R. Current reading normal.

What specific knowledge must such a system incorporate in order to diagnose an suggest remedies? First, the knowledge must come from experts who have acquired and demonstrated diagnostic skills that are better than random selection. For the system to have suggested checking the belt tension of the alter nator, it would have to know that the earlier removal of the belt could be related to the present problem, that the severity of the problem would depend on how poorly the tension was adjusted, and that one month and about 900 miles before appearance of symptoms (battery failure) is not unreasonable. Since it is a highly probable cause and the easiest to test, it ranks as the first suggestion. By requesting more information, the service representative can tell the owner what else may be required and what will not likely be required, such as a new battery.

The successful KB systems of today function in essentially this manner but in the more esoteric fields of medicine, chemistry, biochemistry, and the like. Functionally, they do not do more, though they solve more difficult problems, in the sense that the reasoning chains may be longer. The knowledge, however, is of a similar variety, and the interactive discourse has the same flavor of naturalness and civility.

2.2 GENERAL CONCEPTS

A knowledge-based system is one that supports practitioners in a specific knowledge domain by incorporating the knowledge acquired from experts in that domain and applying it, in combination with certain reasoning skills, to the solution of problems posed by the practitioners. In other words, a KBS functions as an intelligent assistant--a substitute for the expert human consultant who may be needed but unavailable. A KBS may produce solutions or explanations that are more thorough than those produced by a human expert, and may produce them more rapidly; however, one should not assume that the KBS is inherently better than the human. The human has imagination and the capacity for innovation, which even the most expert KBS does not.

There are two kinds of knowledge-based systems: those called diagnostic (or problem-solving) and those called pedagogic. The diagnostic systems are designed to help their users solve problems in specific areas; the pedagogic systems are designed to convey information about specific topics. The two have much in common, both structurally and technologically. (The problem-solving KBS may, of course, teach its users [SHORTLIFFE76], but that is not its primary function.) The distinction is in the users, who are either practitioners or students. A pedagogical system is likely to have less expert knowledge about an area but considerable knowledge about how an understanding of the content of that area is best taught; a diagnostic system may contain a large collection of knowledge acquired from experts in an area (and may educate users by repeatedly exposing them to this knowledge and to the reasoning that goes with it), but is not designed primarily to assist a human's learning process.*

*The user of a diagnostic or problem-solving KBS must be an experienced, knowledgeable practitioner in the field for which the KBS is designed, since only someone knowledgeable in the area of application can guide the diagnosis and understand the relevance or limitations of the results. MYCIN, for example, is intended for the doctor, not for the patient (assuming that the patient is not a doctor).

A KBS is composed of three components or modules: (1) an interface, (2) a cognitive engine, and (3) a knowledge base (see Figure 2.1). (These labels are not used uniformly in the KBS literature, but they do refer to components that perform similar functions.) The knowledge base--the passive element in the system--contains the knowledge sources and fact files. The existence of a knowledge source is a necessary condition. The knowledge source contains the "expertise"--for example, the knowledge of the cause-and-effect relationships or methods and procedures specific to the knowledge domain. The fact files, if present, contain other relevant facts and data.

The cognitive engine drives the system. It performs the system's problem-solving (inference-making or reasoning) operations. It applies the knowledge in the knowledge sources and uses the fact files in the knowledge base to answer questions or solve the problem posed by the user.

The interface provides interactive communication between the user and the system. It allows for the acquisition of data in a variety of forms--a real-time signal, a file of observations, data provided by the user, etc., and for the addition or modification of knowledge in the knowledge base.

A KBS acts as a special-purpose "intelligent agent" on behalf or at the behest of its user; it is not a general-purpose problem solver. It provides supportive knowledge in a well-defined, clearly bounded problem domain. As the user's agent, it must be invoked by the user, and the user must know when, why, and how to invoke it. No present-day KBS is intended for use by a casual, inexpert user. This is a qualitative discrimination that we make, and it is an important one.

Parenthetically, there are a few systems that are referred to by their developers as being knowledge based but are not invoked by a user, and do not carry on any dialog with a user, but respond to the occurrence of data input from an external source. Thus though a user is the individual recipient of the system

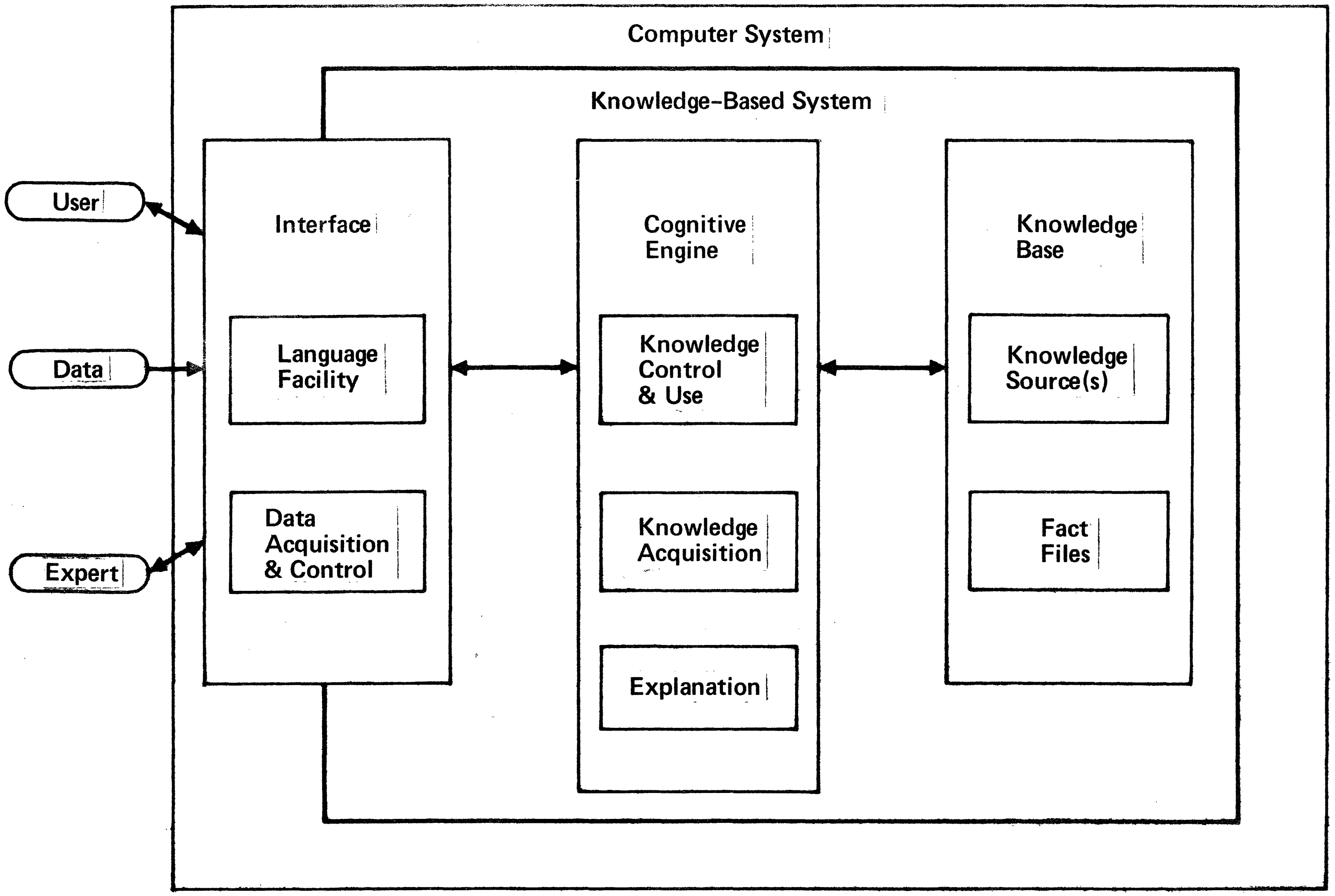


Figure 2-1. KBS Elements and Their Relationship

results, he is a passive rather than an active participant in the problem-solving process. We have not included these systems in this report, but we do not thereby exclude them from consideration as knowledge-based systems.

The Cognitive Engine (CE) provides the central control of the KBS. The CE's principal function is to carry out the plausible reasoning and inference-making that are the heart of the system's problem-solving ability. How the CE does this affects both the power and the performance of the system, but is not the sole determinant. A KBS's ability to solve a particular problem depends on (1) how many paths there are to a solution, (2) the ability of the Cognitive Engine to reduce the number of paths to a minimum, (3) the knowledge in the Knowledge Base, and (4) what information is available within the problem statement. Therefore, although the Cognitive Engine is in command and acts as the driving element, the path to a solution, and the criteria for when to accept a solution or abort a particular path or the entire effort are highly dependent oı the content of the KB and the problem data. The strategies for how to attack a problem and the heuristics of how best to carry on the process are part of the knowledge contained in the CE, and, while these include criteria for selecting from among alternative paths at any point, the CE does not contain sufficient knowledge to determine a priori when an acceptable solution has been found or when the effort should be aborted. These considerations are part of the knowledge in the KB concerning the goal for the problem and the related knowledge about what constitutes a reasonable effort. In this respect, a KBS differs from most systems in that there is no guarantee that a solution exists or, if one exists, that it will be found. In conventional systems, the failure to find a solution or generate an answer implies an error in the data, the problem formulation, or the system itself.

To qualify as a KBS, a system should possess the <u>potential</u> for explaining its actions and reasoning processes with respect to an interaction with the user or to a solution it produces. This is another function of the Cognitive Engine (This means not that every KBS can actually explain its behavior to its users, but that the functions necessary to do so can be added without changing the method or operation of the KBS.) Explanations are given in terms of the content of the Knowledge Base, the problem data, and prior interactions with the user and are related only to past activity; the system cannot explain how it might deal with a hypothetical case or how it will continue in solving a present problem.

The explanation will not indicate why and how the CE took the actions it took; it will indicate only the results of those actions in terms of the Knowledge Base, problem data, and user responses. Even so, the explanation may at times be rich enough for the user to infer what search and inference mechanisms the CE used.

The CE must also provide the mechanisms that facilitate the acquisition of new knowledge, the modification of existing knowledge, and the expunging of erroneous or useless knowledge--all of which are done in cooperation with an expert. A KBS does not generally permit its users to make permanent additions or changes to the Knowledge Base.

In summary, the CE is the controlling, active element of the KBS that directs the problem-solving activities, explains the system's behavior to its users upon request, and manages the Knowledge Base.

2.2.1 The Knowledge Base

The Knowledge Base (KB) of a KBS must, like the Cognitive Engine, be a well-organized, readily identifiable element of the system. Ideally, it should contain all of the knowledge necessary for the KBS to perform as an expert agent. In most present-day KBS systems, some of the knowledge resides in the Cognitive Engine (usually for reasons of efficiency), but the trend is away from this division of the knowledge. The KB will contain such knowledge as stipulations of the existence or non-existences of certain things, simple equivalence relationships, relationships between the concrete and the abstract, knowledge of conventions about the domain, methods of the domain, etc.--in other words, the breadth of knowledge acquired by one who has become expert in solving problems in the domain for which the KBS is designed.

As we have noted earlier, the knowledge in the KB of a pedagogic KBS differs from that in the KB of a diagnostic KBS in two ways: (1) the knowledge of the subject being taught need not be as great, and (2) the knowledge about how to teach is quite rich. Since the intent of a pedagogic KBS is the education of its user, it will not solve problems that would be useful or interesting to workers in the domain. It is the KB, therefore, that determines the overall power of a KBS and its usefulness to its users. This is not true of the knowledge incorporated in the CE, because the user has little cognizance of that knowledge. The CE must be an adequate problem-solving _mechanism_, but it is the knowledge in the KB that determines the breadth, depth, and overall domain of applicability of the KBS. Regardless of its knowledge and power, the CE cannot find solutions to problems for which the KB does not contain adequate knowledge on the other hand, a CE with only weak inference methods may find solutions, albeit inefficiently, if the KB is rich enough.

2.2.2 Separation of KBS Elements

The separation of the elements of a KBS is a necessary condition for including a system in that category, since it permits the changing of the domain of applicability by extending, expanding, or substituting another KB independently of the CE. There are several examples of this. EMYCIN* (for Empty MYCIN) is the CE of MYCIN, to which several different KBs have been experimentally attached for solving different classes of problems. Kellogg's Deductive Processor (DP) [KELLOGG77] is an independent CE to which may be attached various KBs related to specific data-management systems and data bases to provide facilities for extracting implicit information from the explicit facts contained in the data bases. Despite these examples, however, there exists no

*Private communication.

general theory of CEs, and, therefore, no general theory of knowledge-based systems complete enough to permit the facility of substituting a KB from a different domain as a means of creating a new KBS for that domain. That is, while the CE and KB are separate, they are not completely independent of one another.

None of the systems we have studied can modify the content of its CE or KB as a result of having solved a problem. In other words, they do not learn or adapt from experience or examples, even though such a capability would be desirable. A present-day KBS can acquire additional knowledge for permanent incorporation only from expert informants, but even this only at the initiative of its users or informants.

KBS technology arose out of artificial-intelligence (AI) research and is just beginning to attain a separate identity of its own, which may account for some of the confusion about what is and what is not a KBS. KBS technology is rapidly evolving into an engineering-like discipline. A KBS must meet specific structural and organizational specifications. From one point of view, it must incorporate many of the concepts embodied in "software engineering"--specifically, the concept that functions be readily identifiable and incorporated in modules that can retain their unique identity within the integrated whole. Thus, when we say that a KBS must have a CE or a KB, we mean that the functions they provide or the needs they satisfy can be uniquely associated with the modules that implement them, and that the modules associated with one function or need are distinct from those associated with another. This is necessary in the case of a KBS for a number of important reasons. First, the state of the technologies needed for constructing KBS is not so thoroughly understood that all of the correct design decisions can be made prior to implementation and testing. Independence of the various components and modules (as much as can at least realistically be achieved) makes change and modification easier at least up to the point where fundamental design decisions require change (such as the technique selected for representing knowledge).

Of equal consideration is the fact that most KB systems are and will be evolving systems. They will grow and change over time, as does most software, but in somewhat unique ways. A KBS will be expected by its users to acquire additional knowledge over time as experts gain deeper understanding in the domain and as the need and utility to its users grow and change. This kind of evolution cannot be accomplished in a system that is tightly bound together as a monolithic software structure. This implies that it is not possible to determine by external observation of a system's behavior alone whether or not it qualifies as a KBS as we define the area. One must also examine the internal structure for the existence of specific and separable functions or components.

In summary, then, to qualify as a KBS, a system must:

(1) be externally invoked by an expert or student in the domain of applicability;

(2) have an identifiable CE that reasons plausibly using the KB and whose solution path is controlled by the content of the KB and problem data;

(3) have the potential for explaining its behavior;

(4) have an identifiable KB that contains expert domain-specific knowledge (this is the most critical aspect of a KBS); and

(5) be organized and structured so that its KB can be expanded and extended and the system's performance improved.

2.3 THE COGNITIVE ENGINE

The Cognitive Engine (CE) combines and organizes the contents of the KB in inference or search structures in order to perform plausible or common-sense reasoning about the domain as it applies to the problem posed by the user. The intent of the CE is to focus the effort as narrowly as possible on the problem (or subproblem) at hand. In order for the CE to solve problems using search techniques (or "heuristic" search techniques, as they are most frequently referred to), it is necessary that there exist a generator for the hypotheses from which the solution can be constructed (e.g., DENDRAL, [BUCHANAN73]). Solutions in domains for which no such generator is possible must be found by applying inferential or deductive processes over the knowledge of the KSs (e.g., MYCIN, [SHORTLIFFE76]).

The knowledge contained in the CE may be general or meta-knowledge about how to reason (infer or search) as well as domain-specific problem-solving knowledge. The ultimate decision about what kind and what level of knowledge to incorporate in the CE depends on the intent of the system and the complexity of the domain, as well as on considerations about performance, efficiency, growth, and so on.

The depth to which the system will pursue a solution is determined primarily by the content of the KB; it is not a unilateral decision incorporated in the CE. This raises two issues: (1) whether the system will always find an answer or solution if one exists (this is called completeness [NILSSON71] and (2) whether the system will find the "best" answer or solution from the set of legitimate ones (this is called admissibility [NILSSON71]). The combination of knowledge in the CE and in the KB determines whether or not a particular KBS satisfies the completeness and admissibility criteria. This will be discussed in more detail in Section 4.

Through some existing KBSs can explain their lines of reasoning to the users and others cannot, it is a necessary condition, as we have said, that a KBS be capable of accommodating this requirement. The addition of an explanation component must not require that the KBS be redesigned and reimplemented. An explanation is based on the interaction of the CE with the content of the KB. The explanation need not (and probably cannot) include the knowledge that is embedded in the CE, since it is usually not preserved in the solution method and is deeply buried in the system, but may include information acquired from interaction with the user. Despite the fact that the CE's behavior is not incorporated in the explanation, that explanation should be understood by the user because of the knowledge about the domain that the system and the user have in common. (This is another reason why the user should be a worker or student i the domain.) Most KBSs provide the facility for exploring the explanation to various levels of detail.

2.4 THE KNOWLEDGE BASE

As we have said, the Knowledge Base (KB) is a passive element of the KBS. Though passive, the KB determines the performance and utility of the KBS, because the CE depends on the knowledge in the KB. In this section, we describe the characteristics of the KB that are common to all of the KB systems we have studied. They do not necessarily represent the necessary and/or sufficient conditions for a KB.

The KB of a KBS may contain both Knowledge Sources (KSs) and fact files. At least one KS is mandatory; whether fact files are necessary depends on the domain. The fact files that are contained in a KB are equivalent to a data base in that they contain attribute values and the equivalent type of information that may be required for the complete solution or result. A collection of fact files without a Knowledge Source, as we have defined it, is not a KB. Therefore, a system in which all of the expert domain knowledge is embedded in the CE would not, by our definition, qualify as a KBS. A Management Information System constructed from a conventional data-management system and a collection of sophisticated application programs that provides its users with decision-making aids, trend analyses, etc., is not a KBS.

A KS contains what we have been calling knowledge. Whether a KBS has a single or multiple KSs results from system design decisions that are both philosophical and practical. Multiple KSs are usually necessary when there are multiple "levels" of knowledge, such as problem-specific knowledge and knowledge (often called Meta-knowledge) about how the CE can best use the problem-specific knowledge. Since the two kinds of knowledge are used for different purposes, it is reasonable to keep them in different KSs. It is also often true that more than one kind of problem-specific knowledge is acquired from different experts and that there is no efficient single method for representing all of the knowledge. Since different representations are needed, separation into separate KSs is logical.

A KB often contains an indication of the certainty, veracity, or credibility of its representation of knowledge. MYCIN [SHORTLIFFE76] attaches Certainty Factor (CFs) to each item of knowledge it contains, and its CE has a means of combining these CFs to arrive at a certainty value for a conclusion. We have observed no fact files in existing KBSs that contain the equivalent of CFs or other measures of their veracity, but we see no reason why the concept cannot be readily applied to them. It is this property of being able to cope with less than certain knowledge that imbues a KBS with its power and ability to reason plausibly.

We cannot stress too strongly that the expert, domain-specific knowledge in the KSs must not only represent the body of theory about the domain, it must also contain knowledge of how to apply the theory to a given problem or class of problems. This is the kind of knowledge that a person learns from working in the domain and attaining the appellation "expert." For a variety of reasons--efficiency of representation among them--knowledge of how to apply the theory is often incorporated in the CE rather than in a KS.

The knowledge of the KSs in the systems we have examined is of the following types (although the knowledge in no system included all types):

(1) Methods for specifying cause-effect relationships, implications, or inferences, using production rules, predicate-calculus expressions, or other logical representations, depending on the richness of the relationship to be represented.

(2) Plans of action for how one would achieve an end result in the world external to the model that the system represents. For instance, such a procedure may describe how to transform one chemical compound into another for a chemical-synthesis system, or how to purify an intermediate compound. Or it may stipulate the usual logical steps in solving a problem of a particular class or type. It is the equivalent

of the instructions that come with a do-it-yourself kit or unassembled toy or device, in that the user's intent, skill, and some fundamental understanding of content are assumed.

(3) Declaratives that identify objects within the modeled domain and distinguish them from objects that are not within the domain. These declaratives may describe properties of objects, relationships among objects, definitions of terms or constructs, schemata that identify the legal relationships or transformations applicable to the domain, or first-level abstractions, such as class or set memberships for the elements of the domain.

(4) Meta-properties, which are a higher level of abstraction about the domain and the solution space and methods. They are not always embodied in a KS but may be incorporated in the CE, which makes them less readily identifiable. They take the form of meta-rules--that is, rules about using the knowledge in 1, 2, and 3 above. They provide means for determining and assuring the consistency, coherency, and reliability of intermediate results and steps as well as the final solution and answers. They may also restrict the solution space in various ways (such as pruning and ordering a "move" tree) that markedly improves the efficiency of the system.

(5) Advice (sometimes called heuristics) that is similar to meta-properties in intent, but that does not carry the same strength of influence. Advice may be a hint to the CE as to what knowledge is best to use next or how to escape from a possible blind alley or what is the most likely transformation that will yield a useful result. This is the "soft" knowledge that experts acquire from experience in working in the domain and is rarely contained in textbooks and papers. It often consists largely of intuition and has little scientific or theoretical

support, but is highly valuable because it is frequently the knowledg
that gives the system (and the expert) good performance.

There are a variety of techniques that have been used to represent KSs with these characteristics. They will be described in detail in Section 4 below.

2.5 THE INTERFACE

The interface is the communication port between the system and the external world. As such, it provides three functions: (1) interaction with the user (i.e., accepting input and returning results, explanations, or other output, often in English or a stylized natural language of the domain), (2) addition of knowledge to the KB by a domain expert, and (3) acquisition by the KB of problem data (e.g., real-time signals, a file of observations, user-provided data). The interface must logically exist, but its actual realization may make it difficult to identify it as a separate element because of the breadth of function it embodies. Some of its functions may be contained in the CE or be provided by the computer-system environment within which the KBS functions, or both. The three interfaces, when examined individually, have the following properties and underlying rationale, and perform the following functions:

(1) The User Interface should accommodate the jargon or a lexicon specific to the domain of the KBS and may permit a "natural" language. It provides the necessary facilities for the user as a poser of problems and a consumer of results (answers, solutions, termination, explanation, or whatever). It is not the port through which expert knowledge is entered into the system, nor is it intended to support casual, inexpert users. A KBS is analogous to an investment institution in which an interactive system provides support for investment managers. The expert knowledge for such a system is provided by the "back-room" analysts (who also may be users). The users are brokers and portfolio managers; such a system is not available to to the clients of the institution, and would not be particularly useful to them if it were, for a variety of reasons: their lack of fundamental knowledge of local jargon, house rules, government regulations, trading procedures, etc. A system designed for clients would be a different one if it were even feasible, and then one would have to differentiate between the sophisticated or knowledgeable clients and the casual ones.

(2) The Knowledge-Acquisition (Expert) Interface is used by a domain expert (who has gained some feeling for, if not an understanding of, the computer-science aspects of the system) as the provider of knowledge for the KSs. In some systems, the user is able to provide additional temporary knowledge or advice to the KSs through this interface [KELLOGG77]. Still other systems may acquire their knowledge through a quite different mechanism, such as that of Meta-DENDRAL, which is a system that creates the KSs for DENDRAL from observed results of spectrographic experiments. In this case, no human agent is directly involved in providing the knowledge to the KSs of DENDRAL, so the interface must accommodate input of machine-generated knowledge. Associated with the Knowledge-Acquisition Interface is some means of verifying the incoming knowledge, sometimes limited to syntax checking, but often including tests for coherence and consistency with prior knowledge both in the KSs and the CE [SHORTLIFFE76]. It is possible that the Knowledge-Acquisition Interface and the User Interface use some system components in common, such as the language processor, but they are considered logically separate. It is not usual that the CE's knowledge can be supplied, modified, or added to through the Knowledge Acquisition Interface.

(3) The Data Interface is more conventional than the other two. It is similar to that of most other interactive computer systems, in that it incorporates (1) facilities for user input of parameters and data and responses to the system's queries; (2) the mechanism for locating and accessing data sets, files, or data bases; and (3) a capability for accessing real-time (or quasi-real-time) data streams. The Data Interface of each KBS need provide for only those data sources that are meaningful or necessary to its operation. Many of the functions necessary to provide the Data Interface may be drawn directly from the computer-system environment within which the KBS functions.

2.6 SUMMARY

A KBS can, in summary, be said to be a problem-solving or pedagogic agent for its user(s). To qualify as a KBS, a system must have: (1) a Cognitive Engine that reasons plausibly within the domain of application, can accommodate a mechanism for explaining the system's problem-solving behavior, and supports the acquisition of new knowledge; (2) a Knowledge Base that contains the Knowledge Sources and Fact Files needed to solve problems; and (3) interfaces through which user queries, problem data, and expert knowledge can be communicated to the system.

3. A KBS CASE STUDY

This section--an examination of an actual knowledge-based system, MYCIN--is presented with two purposes in mind. The first is to provide a more detailed view of KBS technology by examining a specific system as it relates to our definition. The second is to encompass other related aspects of KBS technology that are covered in more general terms in the later sections. MYCIN, a medical consultation system, was chosen for several reasons. Most improtantly, there exists sufficient documentation about the system to perform such a study--namely, a book and related papers [SHORTLIFFE76, 75a, 75b, 73; DAVIS77, 7 MYCIN, in our opinion, is also the best representative system of the present state of KBS technology. Last, but not least, it appears to have exerted significant influence on other computer scientists involved in KBS technology development and applications. It should be noted that this study of MYCIN is based on the content of the available literature; we have not observed the system in use or examined the program itself. Since this section only summarizes the system for our specific purposes here, we encourage those interested in more detail to read Shortliffe's book [SHORTLIFFE76].

3.1 THE PROBLEM DOMAIN AND THE USERS

Before examining MYCIN's elements, we must first understand something about the problem domain and the intended user community. Simply stated, MYCIN is a knowledge-based interactive computer system intended to provide advice to physicians on prescribing antimicrobial therapy for bacterial infections. The present version is limited to providing advice about bacterial infections of the blood (bacteremia); the intent is to gradually broaden the system into other infectious-disease topics. As one would imagine, the problem of therapy selection and recommendation for an infectious disease is difficult and complex. Even when restricted to bacteremia, the problem of therapy selection poses many problems. The first is to determine whether or not the infection is serious enough to warrant treatment. It is to no one's benefit, economically or physically, to prescribe unnecessary drugs or other treatment, though it has been observed to happen far too frequently. If it is determined that treatment is warranted, there is no panacea for infectious diseases. Therefore, one should know what organism is causing the infection, but that itself is not a simple problem. One must obtain a specimen of the infection for culturing, analysis, and identification by a laboratory. This is a time consuming process. It takes from 12 to 24 hours to determine whether there is an organism and make a preliminary identification of its general characteristics. Another 24 to 48 hours are required to obtain specific identification, and possibly even more time to determine which specific antimicrobial drug is most effective in either counteracting the organism or arresting its growth. In many cases, the infection is serious enough that treatment must be begun before all of the analysis can be completed. Therefore, any recommended therapy must be based on incomplete information. To further complicate matters, the most effective drug against the suspected or identified organism may be totally inappropriate for the specific patient because of age or medical conditions and problems. Thus, any system or consulting physician must be aware of all of these complexities if proper advice is to be rendered in each specific case. MYCIN has been designed to cope with just such complexities and interrelationships among the many variables and to provide a physician with advice that is proper for each individual patient.

Though the problem is quite complex, the domain is well bounded. MYCIN need not have knowledge about medicine in general, or any of the many medical specialities that have no bearing on infectious diseases. It does require specific knowledge that relates to local experience with various infectious organisms in terms of resistance of known strains to specific drugs, which varies from locale to locale. It does not need general knowledge about the theory of infectious disease, but it must have knowledge of symptoms related to specific infections. We will explore below in more detail the specific kinds of knowledge incorporated in MYCIN.

MYCIN is intended to be used by physicians. The dialogue that it carries on with the user is in the jargon of medicine and specifically that of infectious diseases, laboratory procedures, infectious organisms, drugs, etc. Thus, a user is expected to be a competent medical practitioner.

3.2 MYCIN'S KNOWLEDGE BASE

MYCIN's knowledge base (KB) contains several knowledge sources--decision rules (or production rules), clinical parameters, special functions, and procedures for therapy selection. We will briefly describe the content and purpose of each and indicate the method of representation for each. Detailed descriptions of these and other representational techniques are contained in Section 4.

The primary knowledge source in MYCIN is the collection of decision, production, or situation-action (SA) rules. Most of the other knowledge in the system relates to the use or evaluation of the rules. Each rule consists of a Premise, which may be a condition or a conjunction of conditions, an Action to be taken, and sometimes an Else clause. For the action to be taken, each of the conditions in the Premise must hold. If the truth of the Premise cannot be ascertained or the Premise is false, the action in the Else clause is taken if one exists; otherwise, the rule is ignored. In addition, the strength of each rule's inference is specified by a certainity factor (CF) in the range -1 to +1. (CF's will be discussed below under the topic of the cognitive engine.) Each rule in MYCIN falls into one and only one of the following categories:

(1) rules that may be applied to any culture,

(2) rules that may only be applied to current cultures,

(3) rules that may be applied to current organisms,

(4) rules that may be applied to any antimicrobial agent administered to combat a specific organism,

(5) rules that may be applied to operative procedures,

(6) rules that are used to order the list of possible therapeutic recommendations,

(7) rules that may be applied to any organism,

(8) rules that may be applied to the patient,

(9) rules that may be applied to drugs given to combat prior organisms,

(10) rules that may be applied only to prior cultures,

(11) rules that may be applied only to organisms isolated in prior culture

(12) rules that store information regarding drugs of choice.

One can readily infer from these categories both the scope of MYCIN's knowledge embodied in rules and the intent of that knowledge. Each one of these categories is in turn related to one (or at most two) of the ten "contexts" with which MYCIN must deal in its reasoning processes. The ten contexts and the creation of the system's context tree will be discussed below under the cognitive engine.

The system also contains a collection of clinical parameters, represented as <attribute, object, value> triples. These clinical parameters specify the characteristics of the various contexts that appear in the context tree. The parameters are of three fundamentally different kinds: single-valued, multi-valued, and binary (a special case of single-valued with only two possible values, yes or no). These clinical parameters fall into six categories: (1) attributes of cultures, (2) attributes of administered drugs, (3) attributes of operative procedures, (4) attributes of organisms, (5) attributes of the patient, and (6) attributes of therapies being considered for recommendation. Each of the parameters has a certainty factor reflecting the system's "belief" that the value is correct and an associated set of properties that is used during consideration of the parameter for a given context. These properties specify such things as the range of expected values a property may have, the sentence to transmit to the user when requesting data from him, the list of rules whose Premise references the parameter, the list of rules whose Action or Else clause permits a conclusion to be made regarding the parameter, etc. Only those properties that are relevant to each parameter are associated with it. However, properly specifying how the parameter is to be represented in English is mandatory for all.

Additional information is contained in simple lists that simplify references to variables and optimize knowledge storage by avoiding unnecessary duplication. These lists contain such things as the names of organisms known to the system and the names of normally sterile and non-sterile sites from which organisms are isolated.

In conjunction with a set of four special functions, MYCIN uses knowledge tables to permit a single rule to accomplish a task that would otherwise require several rules. The knowledge tables contain a record of certain clinical parameters and the values they may take on under various circumstances. One such table contains the gramstain, morphology, and aerobicity for every bacterial genus known to the system.

This constitutes the majority of MYCIN's knowledge base, which permits the system to comprehend the nature of an infection without complete information about the organism involved and provide the physician with proper advice regarding treatment under the circumstances. This organization and structure, along with the way the knowledge is used, facilitates the system's ability to explain its actions and advice.

There is one knowledge source in MYCIN that is not represented by any of the above, but is implemented as a set of functions. This is the knowledge required for choosing the apparent first-choice drug to be recommended. Because of the manner in which this knowledge is incorporated in the system, its ability to explain how the selection was made is inadequate. Ways for representing this knowledge as decision rules are being studied at this time.

3.3 MYCIN'S COGNITIVE ENGINE

The following description of MYCIN's cognitive engine is somewhat simplified, but it retains the essential flavor. MYCIN's cognitive engine is domain independent in the sense that none of the knowledge required to provide advice about bacteremia is embedded in it. Thus, additional rules concerning infectious disease may readily be added, or a new knowledge base could be substituted to provide therapeutic advice about a different domain of infections. It is possible that this CE could be applied to domains completely outside medicine, and it is said that this has been done. But it does not follow that MYCIN's CE is universal enough to be usable in any knowledge-based system.

The task that MYCIN performs, under the control of its CE, can be characterized as a four-stage decision process:

(1) decide which organisms, if any, are causing significant disease;

(2) determine the likely identity of the significant organisms;

(3) decide which drugs are potentially useful,

(4) select the best drug or drugs.

A consultation session between a physician and MYCIN results from a simple two-step procedure:

(1) Create the patient "context" as the top node in the context tree.

(2) Attempt to apply the "goal-rule" to the newly created patient context.

The "goal-rule" is one of the rules from the category of those that may be applied to the patient (as described above), and states:

If there is an organism that requires therapy and
   consideration has been given to the possible existence of additional
   organisms requiring therapy, even though they have not been recovered
   from any current cultures,

then do the following:

Compile a list of possible therapies which, based upon sensitivity data, may be effective against the organisms requiring treatment and
determine the best therapy recommended from the complied list;

otherwise, indicate that the patient does not require therapy.

This rule obviously embodies the tasks of the four-stage decision process given above.

The two components or programs that constitute MYCIN's CE are called MONITOR and FINDOUT. MONITOR's function is to determine whether the conditions stated in the Premise of a rule are true. To do so, it considers each condition of the Premise at hand, first determining whether it has all of the necessary information to make the determination. If it requires information, it calls FINDOUT to obtain what is needed. FINDOUT first determines whether the needed information is laboratory data. If it is, it asks the physician for it. If the physician cannot provide it, FINDOUT retrieves the list of rules that may aid in deducing the information and calls MONITOR to evaluate the rules. When the process completes, control is returned to MONITOR. If the information needed is not laboratory data, FINDOUT retrieves the list of rules that may aid in deducing the needed information and calls MONITOR to evaluate the rules. If the deductive process of applying the rules (backward from a goal to the data or information needed) cannot provide the needed information, the physician is asked to provide it. In either case, control is returned to MONITOR. Given the information that is provided by FINDOUT or that was already available, MONITOR determines whether the entire Premise is true. If it is not, and there is no Else clause, the rule is rejected. If the Premise is true or the Else clause is invoked, the conclusion stated in the Action of the rule or in the Else clause is added to the ongoing record of the consultation, and the process completes. Note that there is a recursive relationship between MONITOR and FINDOUT, such that so long as any information is needed to evaluate a Premise, or rules are required to develop the needed information, the two components are in a recursively dependent and oscillating relationship until the very first rule invoked, the "goal-rule", is satisfied. In the process of evaluating the rules, a great deal of related and necessary information and data are developed and retained in various tables and structures in the workspace. They serve two purposes: (1) they prevent wasted effort that would be required to redevelop information that has already been obtained, and to prevent the system from endlessly chasing its tail; and (2) they provide the necessary history required for the explanations that may be requested by the user.

In addition to having certainty factors (CFs) for the rules and the clinical parameters in the knowledge base, the physician, when asked for either laboratory data or other information that the system itself cannot deduce, may attach a CF to his input. The default, if the physician does not provide a CF, is assumed to be +1. The certainty factors are the key to permitting MYCIN to perform inexact reasoning. The rationale, mathematics, and application are thoroughly treated in [SHORTLIFFE76] and we will provide only the barest sketch here.

A certainty factor (CF) is a number between -1 and +1 that reflects the degree of belief in a hypothesis. Positive CFs indicate that there is evidence that the hypothesis is valid; the larger the CF, the greater the degree of belief. A CF=1 indicates that the hypothesis is known to be correct. A negative CF indicates that the hypothesis is invalid; CF=-1 means that the hypothesis has been effectively disproven. A CF=0 means either that there is no evidence regarding the hypothesis or that the evidence is equally balanced. The hypothe- ses in the system are statements regarding values of clinical parameters for the nodes in the context tree. To properly perform, MYCIN must deal with competing hypotheses regarding the value of its clinical parameters. To do so, it stores the list of competing values and their CFs for each node in the context tree. Positive and negative CFs are accumulated separately as measures of belief (MB) and measures of disbelief (MD) and added to form a resultant CF for a clinical parameter. The CF of a conclusion is the product of the CF of the rule that generated the conclusion and the tally of the CFs of the clinical parameters that were used in substantiating the conclusion. When a second rule supports the same conclusion, the CFs are combined by $z=x+y(1-x)$, where x is the CF of the first supporting rule, y is the CF of the succeeding rule and z is the resultant CF for the conclusion. The CFs permit the system to report findings to the physician with varying degrees of certainty such as, "There is strongly suggestive evidence that...," "There is suggestive evidence that...," "There is weakly suggestive evidence that...," etc.

The context tree has been mentioned several times above. A brief explanation of it is in order here. The topmost node in the tree is always the patient. Branches are added successively to the existing nodes as FINDOUT discovers a need for them in attempting to obtain requested information for MONITOR. Thus, given only the patient, when MONITOR requests information from FINDOUT about organisms in order to evaluate the first condition in the Premise of the goal-rule, FINDOUT discovers that it cannot get organism information without having information about cultures. Thus, context(s) concerning culture(s) is spawned from the patient node, from which eventually are spawned contexts for the organisms identified by the cultures. For those organisms deemed significant, links attach to context nodes about the relevant drugs for treating these organisms. Thus, the context tree is tailored for each patient as the system progresses through its reasoning process.

MYCIN's cognitive engine is relatively simply yet quite powerful in that it performs both efficiently and quite effectively in conjunction with the knowledge base in providing advice on bacteremia as judged by an independent panel of physicians (among whom, it was noted, there was some disagreement on what the proper therapy should be in each of the cases discussed).

3.4 MYCIN'S EXPLANATIONS

One of the primary design consideration taken in MYCIN was the requirement that the system be able to explain its decisions if physicians were going to accept it. Selecting rules as the representation of the system's knowledge greatly facilitated the implementation of this capability. The physician using the system enters the explanation subsystem automatically when the consultation phase is completed, or he may enter it upon demand during the consultation session at any point at which the system requests input from him. In the latte case, he can input "WHY" to request a detailed answer about the question just asked of him or he can input "QA" to enter the general question-answering expla nation subsystem to explore the decisions and other aspects of the consultation up to the point of divergence.

The explanation provides several options to the physician. Since the system automatically (having rendered its advice) enters this mode at the end of the consultation, the physician may simply input "STOP", which terminates the system. He may input "HELP", which provides him with the list of explanation options, which include:

Input	Option
EQ	Explain a specific question asked of the physician during the consultation--each has a sequence number, which must accompany the EQ request.
PR	Requests a particular rule be printed and must be followed by the rule number.
IQ	Is a prefix for a question about information aquired by the system during the consultation. The question is phrased in the limited English that MYCIN can handle.
no prefix	A general question is assumed being asked about the content of MYCIN's rules.

Thus, the physician can ask to have Question 48 explained by inputting "EQ48". To which the system would respond: 48 QUESTION 48 WAS ASKED IN ORDER TO FIND OUT THE PATIENT'S DEGREE OF SICKNESS (ON A SCALE OF 4) IN AN EFFORT TO EXECUTE RULE068. He may then optionally input "PR68" or "WHAT IS RULE068" to see what exactly was being sought and why.

One shortcoming of the explanation system is the requirement of prefixing questions related to information acquired by the system by "IQ" to distinguish them from the general questions about the rules. Both are dealt with by MYCIN's simple language processor (chosen as a compromise between the need for efficient computation to minimize response time and expressive power in posing questions). It is unclear, particularly to the novice user, when the prefix is needed. The designers are exploring ways of dispensing with the requirement.

On balance, the present explanation system (enhancements are being planned) strikes a proper balance between the needs of the users and the ability to meet those needs without unduly complicating the system or overburdening the available computing resources.

3.5 MYCIN'S INTERFACES

The present system incorporates two interfaces. One is for the using physician through which he may answer questions posed by the system and ask questions of it; the other is a knowledge-acquisition interface accessible only to experts recognized as such by the system.

All of the questions asked of the user have been carefully designed not to require the language-understanding component. Thus, instead of asking, "What is the infectious disease diagnosis for the patient?" it asks, "Is there evidence that the patient has a meningitis?" To which only a simple "yes" or "no" (with the possible addition of a CF) is required.

The knowledge-acquisition interface, on the other hand, permits the expert to input a new rule in stylized English, with prompting to obtain the rule in the proper sequence: Premise first, condition by condition, followed by the Action and then an Else clause if one is required. The system then translates the rule into internal form, reordering the conditions of the Premise if necessary, according to a set of criteria developed to improve the rule-evaluation process It then retranslates the rule into English and requests that the expert decide whether the rewritten version was the one intended. If not, the expert may modify selected parts and is not required to restate the entire rule unless there has been a gross misunderstanding.

The same mechanism is used when an expert wants to correct or modify an existing rule. In all cases, when a new or corrected rule has been approved by the expert, the system checks to see whether the rule is consistent with the existing rule set. These consistency checks are not as complete as they might be. If the new or modified rule subsumes or is subsumed by an existing rule, it is not readily discoverable, and no test is made for this condition. If a rule is discovered to be in conflict with an existing rule, it is rejected. (The designers believe that it may be possible to accommodate these conflicts by storing conflicting rules separately and asking the user--if the situation arises--which of the two rules was about to be invoked, that is, which expert's opinion is favored. Once a rule is accepted, all of the tables and properties that need to refer to it are updated, since, in converting the rule to internal form, the system determined which category it belonged in and which context it related to.

The user and expert interfaces appear to have been well thought through and provide a useful and civil interface to the appropriate user within the limitation imposed by the present state of the art. The designers realize that more can be done as technology develops and are actively pursuing those ends.

3.6 DESIGN CONSIDERATIONS FOR MYCIN

Before the actual design of the specifics of the system were undertaken, several conditions were satisfied. It was first established that there was a need for such a system (and an inference that, if such a system came into beinç it would be accepted by the intended users). The need was verified by observation of the present state of medicine in its application of antimicrobials to infectious diseases. It was being overdone--far too many drugs were being prescribed too frequently. It was not being done well, too many broad-spectrum drugs were prescribed when more specific less toxic drugs were available, and a inappropriate drug was being prescribed far too frequently.

Next, it was established that the chosen domain was well bounded and that there were motivated experts who would cooperate in the design process and provide the expert knowledge required by such a system. Given this starting point, it was determined that the system must possess the following six characteristics:

(1) <u>The system should be useful.</u> There must be a need for the assistance provided by the system. The advice given should be reliable; the system must be human engineered for usability by its intended user population.

(2) <u>The system should be educational when appropriate.</u> The system should not overburden the user with information he may not want, but it should be instructive when responding to a user's informational requests. It should provide sufficient information so that, over time, the physician may need to consult the system only in exceptional cases.

(3) <u>The system should be able to explain its advice.</u> It is observed that physician acceptance will, to a great degree, be dependent upon whether or not he is satisfied, not only with the specific advice rendered, but by the system's justification for that advice. The physcian will not accept a dogmatic replacement for his own decisions.

(4) The system should be able to understand questions. If the system is to explain its advice, then it must do so in response to questions. Hardly any physician will bother himself to learn a formal or arcane language by which he could extract explanations. Therefore, a natural-language facility, albeit limited, is mandatory.

(5) The system should be able to acquire new knowledge. Not only is it hardly conceivable that one could incorporate all of the knowledge the system would ever need at its inception, but in a continuously changing world, new knowledge and insights are constantly developing. Thus, if the system is to remain current with the state of knowledge and grow in reliability and performance, it must be able to incorporate new knowledge. Further, some errors are bound to occur, and the erroneous knowledge must be changed or replaced.

(6) The system's knowledge must be easily accessed and modified. This requirement not only sets criteria for providing the user with the content of the system's knowledge, it sets criteria on how the knowledge is to be represented, in terms of how the representation matches the knowledge that the expert uses and how he conceives it. Thus, there should be a good match between the amount of knowledge that can be represented in one unit and the way that it is expressed.

From those design consideration were developed a more detailed set of specifications and requirements that eventually led to construction of the present version of the system. In hindsight, some of the decisions and tradeoffs were less than optimal, but that is usually the case in most new ventures. The project that developed MYCIN continues and is reviewing the original considerations and design decisions, with the goal of refining the system to come as close to the ideal design as possible.

3.7 SUMMARY

It is reasonable to conclude that MYCIN was well conceived and met the majority of its initial requirements. It has yet to be used by the ultimate users for whom it was designed, the doctors in the hospital wards. That will eventually provide validation or rejection of the various assumptions put forth related to specific functional capabilities and the impact of such a system on the practice of medicine.

The two most obvious shortcomings (improvements are presently being sought for both) are embedding the therapy-selection process in functions that severely inhibit explanation of their results and the requirement that the user label certain questions with an identifying prefix. In this sense, MYCIN is not an ideal system, but in all other respects it is one of the best existing examples of a well-done knowledge-based system.

4. TECHNIQUES USED TO CONSTRUCT KBS

This section may be skipped by the reader who does not wish to read a detaile account of the technologies used to construct a KBS. However, Section 3, A Case Study, is highly recommended reading in order to obtain the flavor of a KBS by examining one particular system, its capabilities, usage, and engineering, in depth.

The purpose of this section is to introduce the reader to the techniques and methodologies used to construct problem solving knowledge-based systems. (KBS Because of the many and striking similarities between these systems and Computer Assisted Instruction (CAI) KBS in particular and Artificial Intelligence (AI) systems in general, techniques used in the latter two groups are also covered herein; these techniques are components of the "parts kit" from which the next generation of KBS will be constructed. In addition to AI, several other computer science areas have developed techniques that are used in the construction of KBS. A partial list of the major contributions are summarized in Table 4.1. The list of contributors and techniques is necessarily long, because the complexity and diversity of tasks performed by a KBS require the utilization of many different methodologies.

The following subsections discuss KBS technologies grouped according to function. Section 4.1 describes the methods used to represent the knowledge contained in the Knowledge Sources (KS). Section 4.2 describes the methods used to model and represent the work-space--the dynamic state of a system during its problem-solving activity. Section 4.3 describes techniques that are used to construct Cognitive Engines (CE). Section 4.4 describes the techniques used to build the interface between the KBS and the user.

There is some overlap in the material covered in sections 4.1-4.4 because the choice of a particular technique in one area strongly affects and limits the available choices in the other areas. This effect is shown in Figure 4.1. On the left are shown limiting influences from the domain in which the KBS is to

TABLE 4.1 ORIGINS OF KBS TECHNIQUES

ARTIFICIAL INTELLIGENCE (AI)

- hueristic search
- inference and deduction
- pattern matching
- knowledge representation and acquisition
- system organization

LANGUAGE PROCESSING

- parsing and understanding
- question and response generation
- knowledge representation and acquisition

THEORY OF PROGRAMMING LANGUAGES

- formal theory of computational power
- control structures
- data structures
- system organization
- parsing

MODELING AND SIMULATION

- representation of knowledge
- control structures
- calculation of approximations

DATA BASE MANAGEMENT

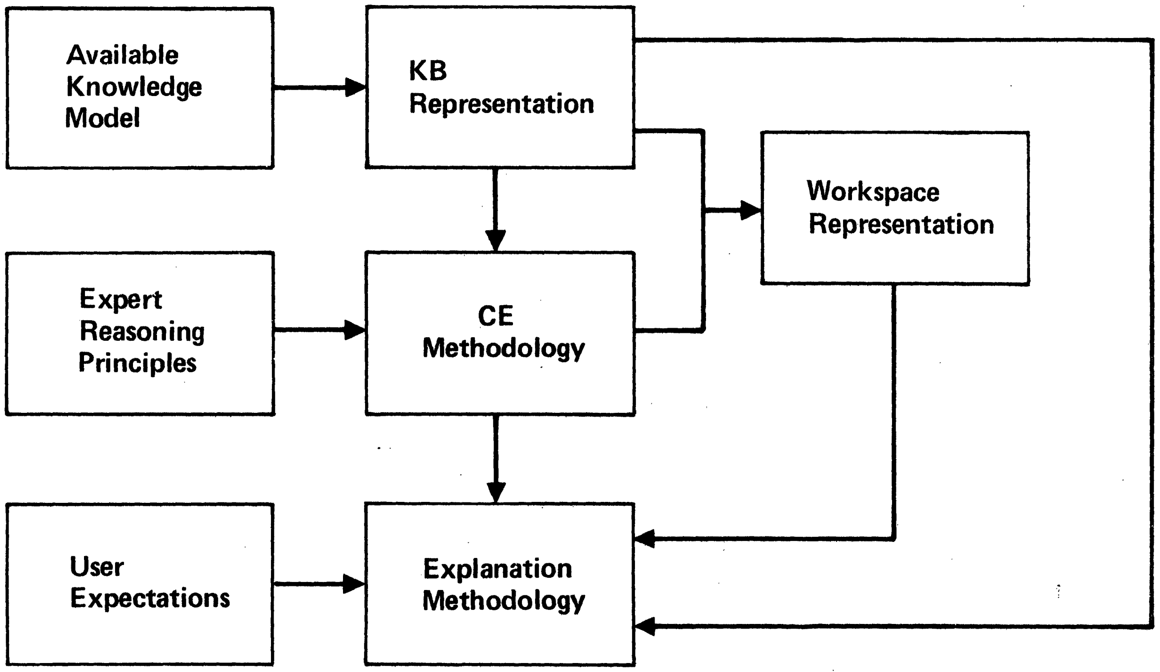
- information retrieval
- updating
- file organization

SOFTWARE ENGINEERING

- system organization
- documentation
- iterative system development

APPLICATION AREAS

- domain-specific algorithms
- human engineering



Note: Read $\alpha \longrightarrow \beta$ as "Choice of α Restricts Options in Choice of β"

Figure 4.1 Restrictions on Choices of KBS Methodologies

perform--namely, the expert's available knowledge model, the expert's reasoning principles and methods, and the users' expectations for the system. These three domain-specific items constrain the selection of the techniques and the methods to be used in the KBS for representing the KB, the CE, and the explanation-generation mechanism. The diagram represents our perception of the relative importance of choice in a KBS. The most important influences are domain considerations followed by choice of a KB representation. Everything else is of less importance. The ordering of importance is reflected in the KBS literature and sets that literature apart from the corresponding literature for Artificial Intelligence. In the latter field, the most important considerations are CE methodology and workspace representations, followed by KB representations. Domain considerations are of relatively minor importance.

It would be extremely valuable to provide here a comparison of techniques and methods. However, such a comparison is difficult to provide for a number of reasons. The most important is that there does not exist a reasonable taxonomy (nor have we been able to invent one) on which to base it. Another difficulty arises because of the contraints discussed above--namely, choices are limited by domain-specific considerations as well as technical incompatibilities. In a sense, a techniqal option is good or bad as it is natural to the domain. Thus, relative merit is as much a domain as a computer-science based measure. Where possible, the following sections attempt to make comparisons based upon abstract features of the various techniques. However, the ultimate comparisons can be made only in the context of a particular domain and problem.

4.1 KNOWLEDGE REPRESENTATION

The knowledge base in a KBS consists of one or more knowledge sources and may, in addition, contain fact files. Fact files are collections of hard data such as values and attributes, e.g., the contents of an engineering handbook or its equivalent. A knowledge source contains an expert's knowledge about the application area--knowledge such as definitions, cause-and-effect correlations, descriptions of plans and procedures, abstractions, problem-solving strategies, and meta rules governing the use of the contents of other knowledge sources (and fact files) in the system and plausible reasoning in the domain. The purpose of this section is to characterize and describe knowledge sources and the techniques and methodologies used to represent them in a computer. In section 4.1.3, an attempt is made to compare the various techniques.

The following references should be of interest to anyone desiring a deeper introduction to the general topic of knowledge and its representation. (Citations are provided throughout the rest of section 4.1 for the techniques now in wide use.) We would be remiss if the only literature we mentioned was from the computer-science community; therefore, we take this opportunity to list a few writings outside the field that are of major historical importance.

[BLOOM56] attempts to describe and provide a taxonomy of intellectual functions It is summarized in Appendix A.

The Greek philosopher, Plato, put forward his theory of forms to establish an epistemology of definitional knowledge that generally resembles a basis for semantic networks. See, in particular, "The Phaedrus", "Parmenides", "The Republic", and "Theaetetus".

Another ancient Greek philosopher, Aristotle, developed and organized the concepts of a predicate calculus and its proper methods of application in discourses about philosophy, ethics, and law. The particular works of interest are "Posterior Analytics" and "Metaphysics", Book 4.

[FREUD 60 and 55] describe a theory of cognitive economy and propose it as an explanation of many intellectual functions.

In [BARTLETT 32], a theory is put forth of necessary ingredients in any explanation of human recall and reasoning processes. This work has been cited by many as the psychological basis for frames.

Some general overview and opinions about knowledge representation from the computer science literature are to be found in [BOBROW 75b and 75c], [BROWN 75b], [CHARNIAK 75], [COLLINS 76], [HAWKINSON 75], [MOOREJ 73], [SIROVICH 72], and [WEISS 61].

4.1.1 Characteristics and Terminology of Knowledge Sources

4.1.1.1 Knowledge Representation Forms

A knowledge source may assume several different forms of representation through a KBS. The domain expert imparts new knowledge to the knowledge acquisition mechanism in the external form. The acquisition mechanism transforms or compiles the external representation into the physical form and merges the new knowledge into the appropriate KS. The physical form is a data structure such as a matrix, list, or n-tuple, or a procedural representation, or some combination of these forms. When another component of the system (such as the CE) accesses the KS, the logical form is used at the interface. The logical form is generally functional and in terms of symbolic keys or indices; that is, it defines the set of questions that can be answered immediately by the KS. The power available at the logical interface is determined by the external form of the knowledge and the amount of compilation performed by the acquisition mechanism. Finally, knowledge is transformed back into the external form when the system provides explanations to the user. Normally, the input form and explanation form of the knowledge are the same or similar except when the input form is highly abbreviated or nontextual. Figure 4.2 summarizes the transformations of knowledge representations throughout a KBS.

4.1.1.2 Knowledge Chunks

Both the external and logical knowledge representation format are partially characterized by the term chunk size. A knowledge chunk is a primitive unit in the representation--that is, in a KS that contains the definitions of several interrelated terms, the definition of a single term is a "chunk". Unless the knowledge-acquisition mechanism compiles incoming chunks by combining them, the chunk size of the external and physical representations will be approximately the same. In the case of combination by the acquisition mechanism, the chunk size of the logical representation will be greater.

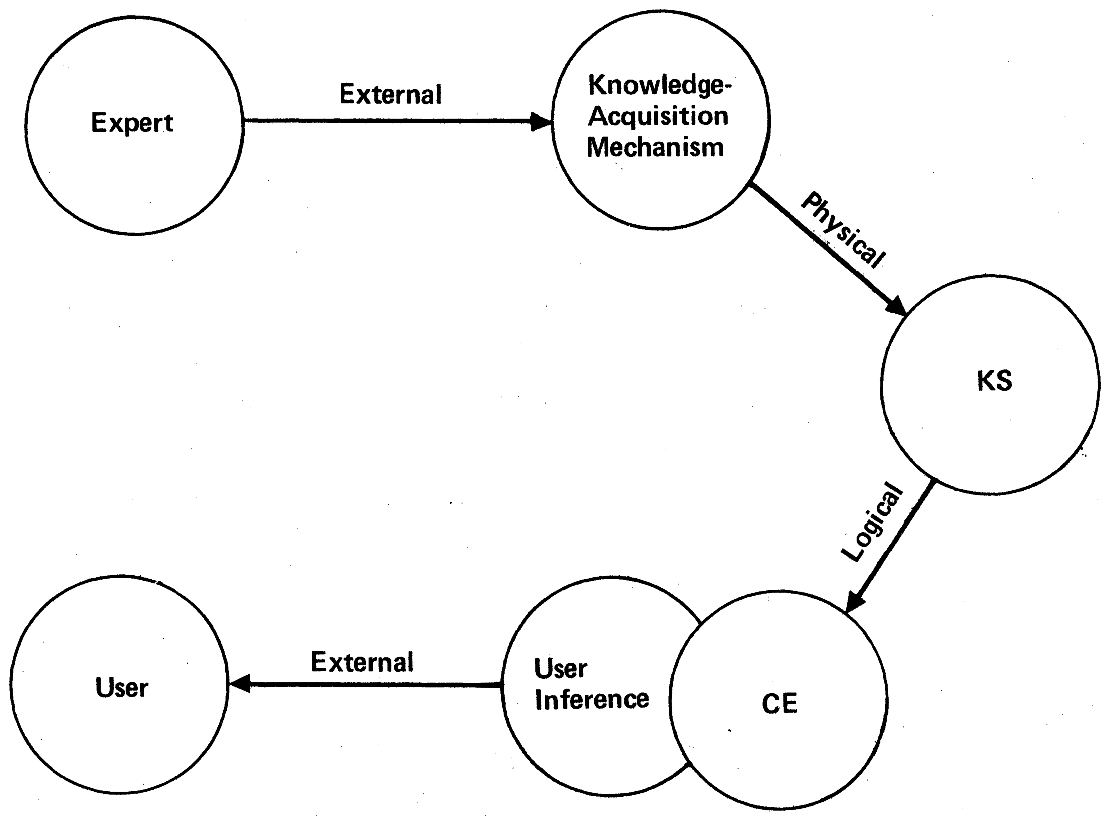


Figure 4.2 Knowledge Representation Forms

The concept of a knowledge chunk is important in describing a KBS because it determines the basic unit (or grain) of behavior. The knowledge chunk is the unit by which the expert augments (or modifies) the KS. The simplest action that can be taken by the CE is to apply or use a single chunk. Therefore, the most primitive explanation of system behavior is a presentation of the chunk form which the behavior resulted.

Chunk size is an inexact and, at best, a relative measure. For certain types of knowledge, the chunk size could be defined as the information-theoretic entropy. (See [SHANNON 49].) However, for the kinds of knowledge required to be in a KBS, computation of entropy is not a practical possibility. (From a theoretical standpoint, it is not clear even what is meant by entropy for many types of knowledge found in a KBS, e.g., definitions and rules of plausible inference.) However, in spite of inexactness, chunk size of knowledge is an important consideration to KBS technology for three reasons:

1. It determines the level at which the expert can instruct the system. If the chunk size is either too large or too small, the expert is forced into an unnatural mode of expressing his knowledge.
2. It in part determines the acceptability of the system's explanation mechanism. Since the knowledge chunks used to derive and support the system's conclusions form the essential part of explanations, acceptability is enhanced when the chunks are the same size or level of detail used by one worker in the application field describing results to another worker in the same field.
3. It determines the kinds and efficiency of reasoning techniques to be used in the KBS. Larger chunk sizes generally permit shorter lines of reasoning. For that reason, they are more likely to lead to a correct conclusion when inexact but plausible inference techniques are used.

These three influences of chunk size all suggest the advantages of a coarse-grained knowledge source. In fact, the most successful KBS and AI systems tend to be characterized by large chunk sizes. The representation techniques used ir most of the early AI systems were predicate calculi and semantic nets; in today's systems, these representations are gradually being replaced by production rules and frames. The continuance of the trend towards large chunks, along with the ability of a KBS to use inexact but plausible reasoning techniques, will result in systems that are capable of intelligent perfromance as measured by even the strictest standards.

4.1.1.3 Credibility Factors

All of the knowledge in a KS need not be true in an axiomatic sense; in fact, it is unclear that a KBS would be necessary or appropriate for use in a domain in which axiomatic knowledge is available. Much of the content of a KS may be "rules of thumb" and working hypotheses. This raises the issue of how a system is to use knowledge of this sort to product acceptable results. The CE, as the reasoning component in the system, has the major responsibility in this area. In many KB systems, the chunks in the KS are rated as to their credibility by the experts who entered them into the system. This rating is then available to the CE as a guide in the reasoning process.

Besides credibility factors for individual knowledge chunks in a KS, credibility factors can occur in other contexts in a KBS--for example, the input problem parameters may not be known with certainty. (Another case occurs when knowledge chunks are combined with each other and with the problem-specific parameters: given the credibility factors of the parameter values and of the knowledge chunks that have been used, what is the certainty of the conclusion?)

There are at least four possible meanings or interpretations of credibility factors:

1. A probability: the fraction of the time that the chunk is true.
2. Strength of belief: how certain is the expert that the chunk is always true?

3. Relevance: what is the probability that use of this chunk will ultimately lead to a completed chain of reasoning that solves the problem at hand?

4. Acceptability: is this a preferred method (a matter of taste) or fact to workers in the field?

It is essential that the kind of credibility factor that is to be used be stated and agreed upon by the expert who instructs the system and by the programmer who builds the CE, because the mathematics for combining and evaluating each of the four kinds is different.

A good discussion of credibility factors, including some mathematical derivations and justifications of the technique used in MYCIN can be found in [SHORTLIFFE 76]. An approach, called "fuzzy logic" is described in [ZADEH 75, 74, and 65] and [GOGUEN 68]. A theory of "confirmation" is described in [CARNAP 50], [HEMPEL 45], and [HARRE 70]. A theory of choice is described in [TVERSKY 72] and [LUCE 65]. Also see [TÖRNEBOHM 66] for a description of criteria that should be met by a choice function.

4.1.1.4 Declarative versus Procedural Representations

There are three different but often confused dichotomies for representing knowledge in computer-based systems: (1) data versus program, (2) active versus passive, and (3) declarative versus procedural. The first data versus program, is at best intuitive and depends upon the evaluator's viewpoint; for example, consider an interpreter-based program-language system: a <u>program</u> written in that language is <u>data</u> from the standpoint of the interpreter.

The second dichotomy, active versus passive, is really not a knowledge-representation issue. Rather, it is a question of control regime and what system component(s) is (are) responsible for instantiation and activation. An <u>active</u> component is always instantiated and may instantiate and control the activation of other components. In other words, an active component is a logistics manager for the available set of program counters. A passive component is one that may operate only at the behest of a more active component. Therefore, active and passive are the endpoints on a (partially) ordered scale of activity. The importance of the active-passive distinction to KB system technology is that, in general, the CE is the only active component in the system, and each KS is strictly passive with respect to all non-KB components in the system. This is true even when the chunks in a KS are programs, since they are operated only by the CE. Even when a KS provides heuristic directions to the CE about what to do next, the CE (through some sort of agenda mechanism) still makes the flow-of-control decisions and is ultimately responsible for resolving potential conflicts from the advice. That this is the case follows from the logical separation of the CE and KB.

The third dichotomy, declarative versus procedural,* is really the computer scientist's version of the epistemologist's know what and know how distinction. It may be argued successfully that (1) there is no strictly formal difference in the power of the two--they are both "universal"--and that (2) both are necessary. However, the real issue is the attitude towards the management of complexity of the interrelationships among knowledge chunks. A proponent of procedural respresentation argues that a major part of intelligent behavior is the ability to apply specialized rules to exploit situation-dependent relationships among knowledge chunks. Hence, he believes that many of the ad hoc interrelationships should be made explicit and that procedures are the best way to do this. On the other hand, an advocate of declarative representations believes that parsimony is the most desirable goal for knowledge representation, and that this is best accomplished using reasonably modular and independent knowledge chunks that are combined by a general-purpose reasoning mechanism to produce the desired results through inference and deduction.

*The remainder of section closely follows [WINOGRAD 75].

An example may help to clarify some of the issues involved. A declarative representation of the statement, "All Chicago lawyers are clever" could be

$$\forall(x)\ [\text{CHICAGOAN}(x)\ \&\ \text{LAWYER}(x) \rightarrow \text{CLEVER}(x)]$$

A general reasoning mechanism could use this single fact for many purposes. For instance, to answer the question, "Is Dan clever?", it would check to see whether Dan is from Chicago and is a lawyer. The same fact could also be used to infer that Richard is not from Chicago, given the information that he is a stupid lawyer. The property of being able to use the same chunk for many purposes, as in this example, is called reversibility. In a strictly procedural representation, the fact would need to be represented differently for each of the many possible usages. Each would demand a specific form, such as "If you find out that someone is a lawyer, check to see whether he is from Chicago, and if so, assert that he is clever". It is not possible to show a simple example demonstrating a clear advantage of a procedural representation, because the value of a procedural representation lies in the complex cases in which interaction of many pieces of knowledge are involved. (However, see Section 4.1.2.2.)

Intelligent systems can be constructed to operate in complex domains only if they incorporate substantial bodies of both know-what and know-how knowledge. Hence, both declarative and procedural knowledge must be present. One way of accomplishing this is called procedural attachment; it is used in the emerging theory of frames, as well as in some production systems (see section 4.1.2.2). The basic concept underlying procedural attachment is that most knowledge should be expressed declaratively (as a data structure) and should permit the optional association of programs with the knowledge chunks and/or the data items within the chunks. The CE executes these programs whenever the knowledge associated with them is referenced. The programs can perform local inference, detect inconsistencies, and give the CE advice on what to do next.

The major topic of this section has been viewpoints on dealing with complexity in knowledge-based systems. Simon (see [SIMON69]) addresses many of the same issues through what he calls "nearly decomposable systems": "...the short-run behavior of each of the component subsystems is approximately independent of the short-run behavior of the other components....In the long run, the behavior of any one of the components depends in only an aggregate way on the behavior of the other components." Winograd (op. cit.) goes on to comment at some length on this remark:

> One of the most powerful ideas of modern science is that many complex systems can be viewed as nearly decomposable systems, and that the components can be studied separately without constant attention to the interactions. If this were not true, the complexity of real-world systems would be far too great for meaningful understanding, and it is possible (as Simon argues) that it would be too great for them to have resulted from a process of evolution.
>
> In viewing systems this way, we must keep an eye on both sides of the duality--we must worry about finding the right decomposition, in order to reduce the apparent complexity, but we must also remember that "the interactions among subsystems are weak but not negligible". In representational terms, this forces us to have representations which facilitate the "weak interactions".
>
> If we look at our debate between opposing epistemologies, we see two metaphors at opposite poles of the modularity/interaction spectrum. Modern symbolic mathematics makes strong use of modularity at both a global and a local level. Globally, one of the most powerful ideas of logic is the clear distinction between axioms and rules of inference. A mathematical object can be completely characterized by giving a set of axioms specific to it, without reference to procedures for using those axioms. Dually, a proof method can be

described and understood completely in the absence of any specific set of axioms on which it is to operate. Locally, axioms represent the ultimate in decomposition of knowledge. Each axiom is taken as true, without regard to how it will interact with the others in the system. In fact, great care is taken to ensure the logical independence of the axioms. Thus a new axiom can be added with the guarantee that as long as it does not make the system inconsistent, anything which could be proved before is still valid. In some sense all changes are additive--we can only "know different" by knowing more.

Programming, on the other hand, is a metaphor in which interaction is primary. The programmer is in direct control of just what will be used when, and the internal functioning of any piece (subroutine) may have side effects which cause strong interactions with the functioning of other pieces. Globally there is no separation into "facts" and "process"--they are interwoven in the sequence of operations. Locally, interactions are strong. It is often futile to try to understand the meaning of a particular subroutine without taking into account just when it will be called, in what environment, and how its results will be used. Knowledge in a program is not changed by adding new subroutines, but by a debugging process in which existing structures are modified, and the resulting changes in interaction must be explicitly accounted for.

If we look back to the advantages offered by the use of the two types of representation, we see that they are primarily advantages offered by different views toward modularity. The flexibility and economy of declarative knowledge come from the ability to decompose knowledge into "what" and "how". The learnability and understandability come from the strong independence of the individual axioms or facts. On the other hand, procedures give an immediate way of formulating the interactions between the static knowledge and the reasoning process, and allow a much richer and more powerful interaction between the "chunks" into which knowledge is divided. In trying to achieve a synthesis, we must ask not "how can we combine programs and facts?", but "How can our formalism take advantage of decomposability without sacrificing the possibilities for interaction?"

If the declarative and procedural formalisms represents endpoints on a spectrum of modularity/interaction, we should be able to see in each of them trends away from the extreme. Indeed, much current work in computing and AI can be seen in this light.

This section closes with a quote from [MINSKY75] about the declarative versus procedural issue from another point of view.

I draw no boundary between a theory of human thinking and a scheme for making an intelligent machine; no purpose would be served by separating these today since neither domain has theories good enough to explain--or to produce--enough mental capacity. There is, however, a difference in professional attitudes. Workers from psychology inherit stronger desires to minimize the variety of assumed mechanisms. I believe this leads to attempts to extract more performance from fewer "basic mechanisms" than is reasonable. Such theories especially neglect mechanisms of procedure control and explicit representations of processes. On the other side, workers in Artificial Intelligence have perhaps focussed too sharply on just such questions. Neither have given enough attention to the structure of knowledge, especially procedural knowledge.

It is understandable why psychologists are uncomfortable with complex proposals not based on well established mechanisms. But I believe that parsimony is still inappropriate at this stage, valuable as it

> may be in later phases of every science. There is room in the anatomy and genetics of the brain for much more mechanism than anyone today is prepared to propose, and we should concentrate for a while more on sufficiency and efficiency rather than on necessity.

The above quotations represent two viewpoints on the major problem facing AI--namely, the management of complexity. Of course, this is also an issue for research in knowledge-based systems. However, the successes to date with knowledge-based systems have been attained by carefully controlling this complexity by selecting and working in domains that are sufficiently constrained while still possessing an interesting and rich problem space. Without these constraints, the KBS developer would have to face all of the issues that confront the psychological modeler and AI researcher. With these constraints, he has been able to construct practical systems with heretofore unachievable capabilities.

4.1.2 Methods of Representing KS

This section describes six techniques used to implement a knowledge source and, hence, represent knowledge in a KBS. These six were selected for discussion because they were the six most discussed in the literature. (Many other, lesser-known techniques have been successfully used to construct systems and should not be ignored simply because they are not included here.) Two of the techniques described below, finite-state machines and programs, are normally used to represent procedural knowledge. Three of the techniques, predicate calculus, production rules, and semantic networks, are normally used to represent declarative knowledge. The sixth technique, frames, is an effective method of combining both procedural and declarative knowledge in a single representation.

From a theoretical viewpoint, all of these techniques have identical representational power because, combined with an appropriate and simple CE, each can represent a universal Turing machine; therefore, the decision to use one method instead of another is based more upon pragmatic considerations, such as naturalness and efficiency for the intended application. Section 4.1.3 makes a comparison of these six techniques based upon some of their inherent properties.

4.1.2.1 Finite-State Machines

A finite-state machine (FSM) is a representation technique for procedural knowledge. The FSM is a finite collection of states. Each state specifies a computation and a decision rule to determine what state should next be entered. Two states are special: the start state is the first state entered, and the calculation terminates whenever the end state is entered. There are two major uses of an FSM: the first is to represent a grammar; the second is to represent protocols or plans of action. The use of an FSM to represent grammars is described in Section 4.4.1.1. (Also, see [WOODS73].) Figure 4.3 graphically shows an FSM representation of a plan of action for making and

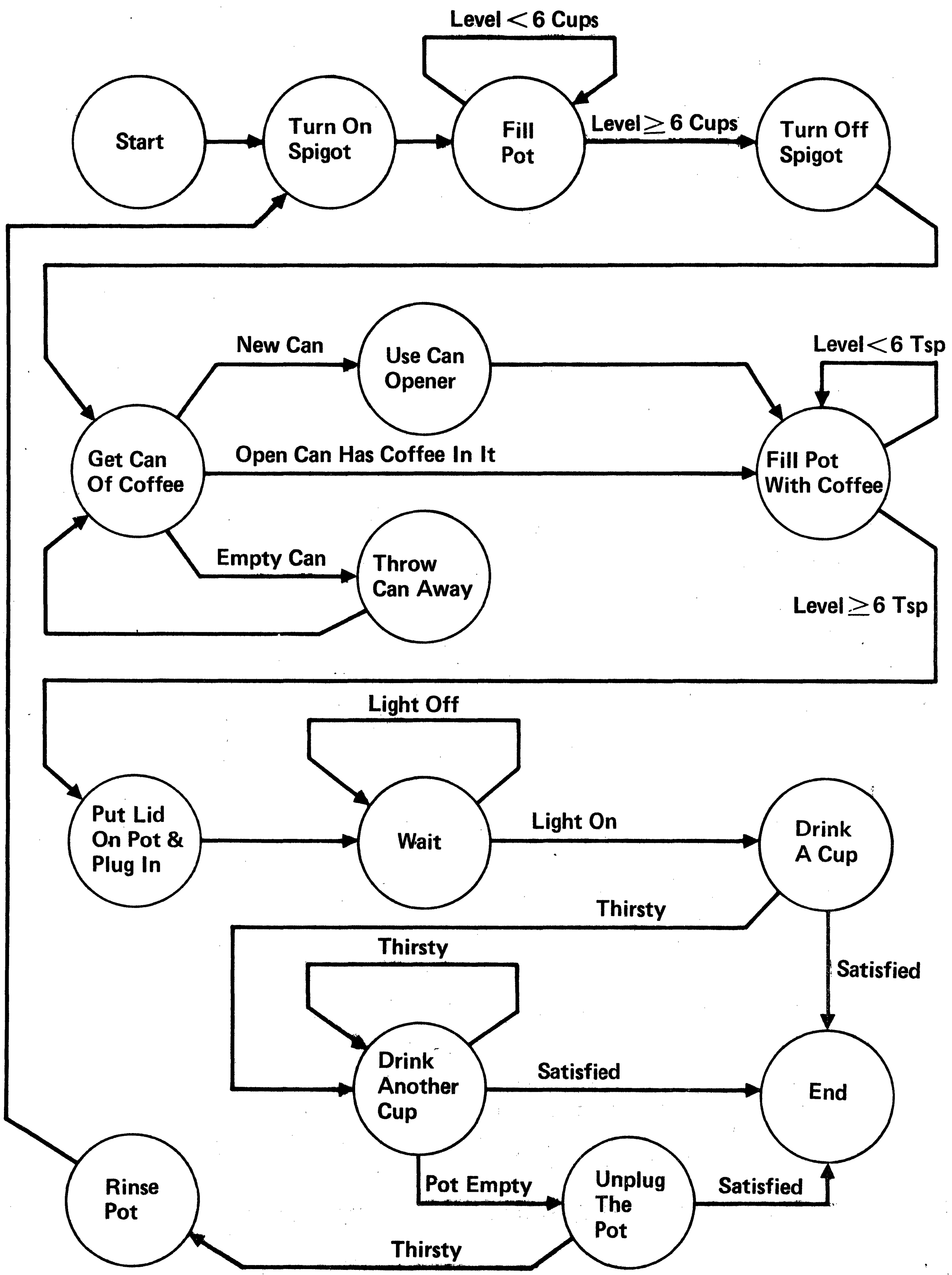


Figure 4.3 Finite-State Machine Representation of a Plan to Make and Drink a Pot of Coffee

drinking a pot of coffee. The circles are the states, and each describes an action to be taken. The decision rule for each state is represented by labels on the set of arcs that leave the state. An arc label is a predicate that must be true for control to pass along it. For example, the state marked "fill pot" has two arcs leaving it. One arc is labeled "level < 6 cups." This arc keeps the FSM in the pot-filling state until the water level reaches the 6-cup line. When it is reached, the arc labeled "level ≥ 6 cups" takes the FSM to the state at which the spigot is turned off.

Four options are available that can affect the power,* size, and reversibility of an FSM: (1) the set of allowable computations in a state, (2) the set of allowable predicates on the arcs, (3) parameterization, and (4) the control mechanism. The kinds of choices available for (1) and (2) are categorized by specification of a set of primitive actions or computations, specification of the rules of combination of actions (e.g., functional composition, sequencing, etc.), and specification of the memory space that can be referenced by the primitive actions. It is also possible to make an FSM with parameters. For example, in the FSM of Figure 4.3, the number of cups of coffee to be brewed could be passed as an argument, and the number, six, replaced by the parameter name on the four arcs on which it appears.

There are two primitive types of control structure** for FSM: <u>deterministic</u> and <u>nondeterministic</u>. In a deterministic FSM, at most one arc is followed out of the present state. This is accomplished by either requiring that at most one arc predicate be true, or by having a rule that selects one arc out of the set that qualifies. In Figure 4.3, the state "drink another cup" has arcs leaving it labeled "pot empty," "satisfied," and "thirsty." It was assumed

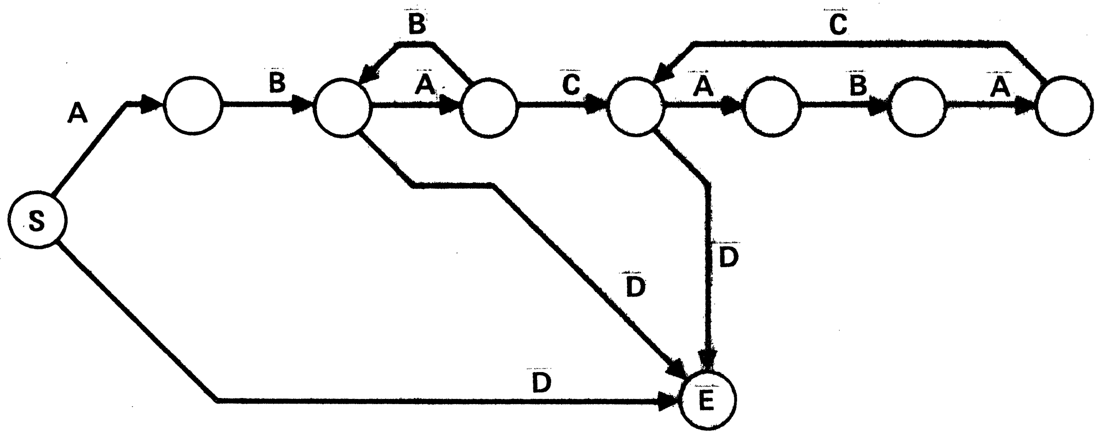
*As used herein, the term finite-state machine describes a representation methodology, not a specific restriction on computation power. For example, if the states of the FSM are permitted access to a read-write tape of indefinite length, full Turing power will result.

**See [FISHER 70] for a more complete taxonomy of control structures.

that there was a selection rule that gave priority to the "pot empty" arc so as to not burn the urn. If the FSM cannot leave the state it currently is in (excepting the end state) because no arc predicate is satisfied, the operation of the machine is said to be blocked. Blocks can occur for two reasons: firs an error in FSM specification--a legally occurring situation is not handled; and second, the plan of action represented by the FSM is unsatisfiable given the current context. The user of the FSM assumes the block has arisen for the second reason and uses the negative result as a cue to try another method or procedure. Sometimes the reason for the block can be determined by inspection and user to guide the new attempt.

In a nondeterministic FSM, it is possible for several different arcs leaving the same state to be satisfied simultaneously. The assumptions are that each path will be followed and that, if any path finally reaches the end state, the FSM has terminated normally. Paths through the nondeterministic FSM may be dropped when they block. Figure 4.4 shows both a deterministic and non-deterministic FSM that recognizes symbol strings that start with zero or more "AB", followed by zero or more "ABAC", and are terminated by a D. (Such an FSM is called a recognizer or an acceptor.) The states (circles) perform no computation. The start states are labeled "S", and the end states are labeled "E". The predicates on the arcs test the next character in the input sequence for equality. The arc predicates are abbreviated by the name of the next necessary character. In the nondeterministic FSM, the state labeled "X" has two arcs leaving it that are both labeled with "A." State "X" is handling two cases: (1) an "A" in one of the initial "AB" groups and the second "A" in the first "ABAC" group. In the example, the deterministic FSM has one more state than the nondeterministic FSM. Classes of FSM are known such that the number of states in a deterministic FSM must be at least an exponential function of the number of states in a nondeterministic FSM that performs the same calculation. Therefore, there is in some cases an obvious advantage to using a nondeterministic FSM even though the interpreter (CE) is more complex.

Deterministic FSM



Nondeterministic FSM

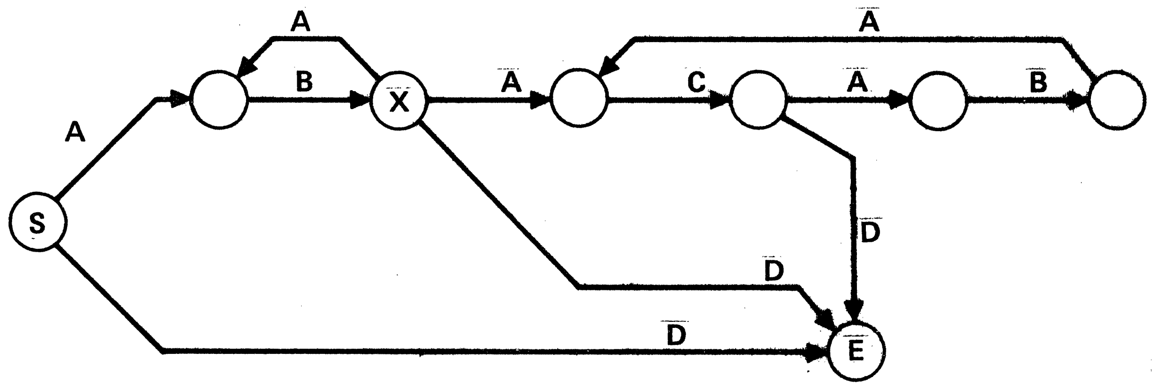


Figure 4.4. Finite State Recognizers for
(AB)* (ABAC)* D

There are certain categories of restrictions, upon allowable computations in the states and upon the arc predicates, for which deterministic and nondeterministic control structures yield differences in computational power. For the simple class of FSM exemplified by Figure 4.4, there is no difference. However, for the class of FSM in which the states can place and remove a character on and from a pushdown stack, and in which the arc predicates can test for equality of the next input character and/or for equality of the top character on the pushdown stack, a power difference exists. A task that shows the difference is the recognition of symbol strings that are symmetric around their midpoint. A good discussion of power differentials in various classes of FSM can be found in [MINSKY 67].

Since FSMs so closely resemble flowchart representations of procedures written in a programming language, it is worthwhile to list some of their desirable and undesirable characteristics in light of that comparison. The desirable characteristics are:

(1) The ability to easily implement nondeterministic control.

(2) The ability to represent and model plans of action for which "procedural" execution inside a computer is meaningless.

(3) Reversibility--that is, an FSM may be examined to answer such questions as what needs to occur to allow it to end up in a particular state.

(4) New plans of action may be constructed dynamically because an FSM representation is easily manipulated.

(5) Many disciplines, both scientific and nonscientific, represent part of their published expert knowledge in a form similar to that of an FSM.

The undesirable characteristics of FSMs are:

(1) The loss of efficiency compared to compiled procedures.

(2) The enforcement of low-level uniformity in the representation, which can make the FSM hard to understand (in a sense, FSMs are better at representing strategies than tactics).

(3) The external format of an FSM representation can lose clarity unless there is a graphic medium available for computer input and display.

4.1.2.2 Using Programs to Represent Knowledge

By definition, every computer system contains some knowledge represented by programs, albeit trivial. The purpose of this section is to describe the techniques used to represent non-trivial expert knowledge in programs. To be specific, a program is code written in an effective formal language. By effective is meant that at each step of execution (equivalently, at each step of the interpretation), the next step can be unambiguously determined by an agreed-upon set of rules. This means, of course, that the rules themselves must constitute an effective program, for which there must exist an agreed-upon set of rules, and so on, ad infinitum. One should not, however, become preoccupied by this "infinite regression" in the definition of effectivity; that problem, like other problems with attempting to formalize intuitive concepts, belongs to the logicians and philosophers. However, the issues involved cannot be taken lightly because a similar problem of definition is encountered when one tries to ascribe meaning to the contents of a KS. One widely held viewpoint is that the contents of a KS have no meaning per se and can come to have meaning only when it is understood how the knowledge is used and/or what effects follow from its use. From this viewpoint, there is an analogy between the CE in a KBS and the effective rules of application for a program.

Programs are usually, but not always used to represent procedural knowledge; in some instances, the majority of chunks in a declarative KS exhibit a regularity that can be exploited by generating those chunks algorithmically. While it is true that such an algorithm contains "how-to" knowledge--namely, how to generate specific knowledge chunks--the user of the KS has available only the declarative knowledge that results.

Figure 4.5 depicts a program representation of knowledge necessary to turn on a water spigot. The example program has two arguments: a human agent, who will perform the task, and the desired temperature of the running water. Much world knowledge is imbedded in this program. For example,

- Water taps are in sinks.
- You need to be close to the sink to control the water taps.
- Cold water comes from the right tap, hot water from the left tap, and temperatures in between by mixing the two.
- Water taps are turned on by twisting clockwise and off by twisting counterclockwise.
- Before adjusting a mixture of water from the two taps to the desired temperature, the hot water should run until it is at full temperature.
- Relative values of temperature such as cold, lukewarm, hot, etc., are used and compared.

Besides this world knowledge, the program contains knowledge about itself--for example:

- The program will not recur indefinitely (when the cold water is turned on to mix with the hot water).
- The program will not get stuck in an infinite loop while trying to adjust the temperature, because only an approximate equality ($\approx$) is necessary to terminate.

```
PROCEDURE turn_on_the_water(agent human, temp temperature)
          locate_at(agent, "sink");
          IF temp="cold"
             THEN tap←"right_tap"
             ELSE tap←"left_tap";
          twist_tap(agent, tap, "right", "full_turn");
          IF tap="left_tap"
             THEN BEGIN
                    WHILE water_temperature≠"hot" idle;
                    IF temp≠"hot"
                       THEN turn_on_the_water(agent, "cold");
                    x←"half_turn";
                    UNTIL water_temperature≈temp
                          LOOP[IF water_temperature>temp
                                  THEN twist_tap(agent, "right_tap", "right", x)
                                  ELSE twist-tap(agent, "right_tap", "left", x);
                               x←x/2];
                    END BLOCK;
          END turn_on_the_water;
```

Figure 4.5. Procedural Knowledge Example

- Program "locate-at" will effectively move the agent to the desired location.
- Program "twist-tap" expects the agent to be in proximity of the tap.

It is interesting to speculate about casting these types of knowledge into declarative rather than procedural form. Of the program's "self" knowledge, only the last example, a case of "What can other knowledge expect of me?", would need to be explicit. The other kinds of self knowledge shown (control structure and what other programs do) are got from a generalized reasoning process normally used with declarative knowledge, and hence need not be explicated in the KS. On the other hand, all the world knowledge listed above, including necessary temporal ordering of the steps, would need to be present in the system. Because that knowledge is almost all ad hoc; it is not easy to see how it could be inferred by or from general principles of reasoning. The advantage of the program representation is that all of this knowledge is brought together in a natural manner. The disadvantages become apparent if one tries to extend this example to a problem domain with multiple kinds of water spigots. Much of the present knowledge applies to only a few cases (e.g., there are two spigots in the sink--there could be one-handled spigots), while some of the knowledge is more universal (e.g., let the water heat up before adjusting the temperature). The problem is simply how to preserve the knowledge that applies to multiple cases--this is the virtue of declarative representations.

Discussed below are two of the many options available when using programs to represent knowledge: invocation methods and control regimes for state retention. The four major methods of program invocation are: direct, procedural attachment, demon, and pattern directed.

Direct invocation occurs when the user (using program) knows precisely which program is to be used and includes a lexical reference to that program through a mechanism such as a subroutine call. Procedural attachment was mentioned in Section 4.1.1.4. The idea is that programs can be associated with data fields in a KS or (dynamically) with parts of the evolving workspace representation. Then, any accessor of a data field that has an associated program is required to invoke that program. The invoker of a program may be unaware of both what program is invoked and what functions the invoked program is to perform. Usually, only the program that makes the attachment has that knowledge.

Programs invoked by the third method are called demons. A demon is introduced to the run-time monitor by a statement such as

DEMON(P,C)

which means "if condition C (a predicate or situation description) is ever encountered during future execution of the system, call program P." A demon is like an interrupt handler in an operating system because it sits on the sidelines, rather like a sentinel that protects the system; they perform no action until (and unless) a specific situation is encountered. They allow knowledge that pertains to highly specialized or unusual situations to be left out of the main stream, making programs more readable and easier to organize. The run-time monitor has the task of watching for an enabling condition for any of the introduced demons. This can be an expensive operation and represents the chief drawback of demons. The alternative to this kind of invocation scheme is to make explicit in-line tests for unusual situations with a resultant lack of clarity.

The fourth method of program invocation is variously called pattern-directed or goal-directed invocation. In a system using this method, each program is named by a pattern that describes the kind of tasks it performs. This pattern is used in lieu of the program's name. Thus, the invoking program specifies a goal that needs to be achieved, and the run-time monitor searches for one or more programs whose patterns match the current goal. One of the programs so found is selected and invoked. If that program succeeds in fulfilling the goal, execution proceeds. If not, another program found by the pattern-matching search can be tried. There are issues concerning in what order to try the programs and what to do if no program can achieve the goal; these issues are discussed further below as control regimes. An example of a pattern for the "locate_at" goal (used in Figure 4.5) might be

(locate_at human object)

This states that the program can plan the sequence of actions necessary to move a human into proximity to an object. Another program in the same system could have a pattern such as

(locate_at object_1 object_2)

This program plans the sequence of actions necessary to move object_1 into proximity to object_2. The second program performs a different task from that of the first program, because the entity that is moved may require an external agent to effect transfer. An interesting case arises if a human is defined to be a kind of object; namely, any goal that matches the first pattern would also match the second pattern.

Since programs operating in a system that allows pattern-directed goal invocation can themselves pose subgoals, this method of invocation allows a natural way of performing means-ends or problem-reduction analysis. Another natural use is for theorem proving where domain-specific heuristics are imbedded with the axiomatic knowledge. The proof proceeds through decomposing the original problem to a set of successively smaller subproblems, the solution of which implies the validity of the original theorem. The heuristics control the search by specifying the order in which subproblems are generated. The chief disadvantage of the pattern-directed invocation technique is that, in practice, each program must have in it a tremendous amount of information about the other programs it may invoke. If it does not, glaring inefficiencies and bugs in the form of infinite generation of subproblems or no satisfaction of certain goals will surely arise. Systems using pattern-directed invocation techniques, therefore, become increasingly difficult to modify and extend over time.

Besides the options available in invocation techniques, there are options in the control regimes that may be used for a system. Though there are many facets of this topic, we are interested only in the problem of state retention when the system faces several alternatives as to what to do next. There are three basic choices for a control regime: sequential, parallel, and nondeterministic. In a sequential regime, the program itself explicitly makes the choice of what to do next and how to reestablish enough state in case the first attempts are failures. One program invokes another and expects the latter to return, subroutine style, at the completion of its activity. In a parallel regime, many subprograms can operate simultaneously or, at least, in some interleaved fashion. The programs themselves are responsible for explicit synchronization activities to avoid the many problems that can arise when various resources (such as variable bindings) are shared by the active set of programs. A nondeterministic control regime is like a parallel control regime except that each program, when operating, is guaranteed to have the same environment it would have if it were the only program of the active set that had ever operated. A nondeterministic control regime is often called automatic backtracking. The idea is that one of the many possible alternative branches is followed. During execution of a branch, changes made to the computation state are remembered. In case a failure is encountered, the remembered state changes are undone, and control returns to a decision point at which an alternate branch remains. Then this branch is followed, and so on.

The different invocation and state-retention options discussed above differ in their generality. The more general the mechanism used, the more complicatec must be the run-time monitor. The penalty for complexity is loss of efficienc) On the positive side, systems using the more general mechanisms tend to be better organized and, hence, easier to modify and extend because a large amount of bookkeeping is buried in the run-time monitor.

A good overview of recent developments discussed in this section can be found in [BOBROW75d]. Another paper that discussed many issues of general interest is [HEWITT73]. In [BOBROW73], a general method of providing state retention is described--commonly called a "spaghetti stack." [FISHER70] describes a toxonomic theory of control structures. A few interesting systems that use programs to represent knowledge are described in [HEWITT72], [R. MOORE75], [SUSSMAN75], and [WINOGRAD72].

4.1.2.3 Predicate Calculus Representation of Knowledge*

The predicate calculus is a formal symbolic notation system (formal language) for expressing logical relationships and making assertions about a domain or model. There are three parts to its definition: (1) syntax specification--th grammar that defines legal expressions in the language, (2) semantic specification--the rules that relate the symbols in the language to objects in the domain, and (3) legal operations--rules of inference that create legal expressions from other legal expressions. The syntactically legal expressions in the predicate calculus are called Well-Formed Formulae (WFF). Through the semantic specification rules, a WFF makes an assertion about the domain. The WFFs are said to have the value T or F, depending on whether the assertions are true or false of the domain. The legal operators are constrained in such

*Many definitions and examples in this section are taken from [NILSSON71] a book that should be read by anyone seriously interested in the topic.

a way that the value (T or F) of a WFF output by a transformation can be directly determined from the values of the WFFs input to the transformation.

The syntax specification of the _first-order_ predicate calculus has two parts: the specification of an alphabet of symbols and the method by which legal expressions are constructed from these symbols. The alphabet consists of the following set of symbols:

1. Punctuation marks: , ( )
2. Logical symbols: $\sim \Rightarrow \vee \wedge$ (The symbols are read, respectively as _not_, _implies_, _or_, and _and_.)
3. Quantifier symbols: $\forall$ $\exists$ (The symbol $\forall$, is called the _universal quantifier_ and is read _for all_; the symbol $\exists$ is called the _existential quantifier_ and is read as _there exists_.)
4. n-adic function letters: f_i^n $(i \geq 1, n \geq 0)$ (The f_i^0 are called constant letters. For simplicity, it is conventional to use lowercase letters near the beginning of the alphabet (i.e., a, b, c), or lowercase words (e.g., line, dog) as abbreviations for the f_i^0. Similarly, the lowercase letters f, g, h and lowercase function names, such as cos, are used without subscripts in place of the other f_i^n.)
5. n-adic predicate letters: p_i^n $(i \geq 1, n \geq 0)$ (The p_i^0 are called proposition letters. For simplicity, capital letters near the middle of the alphabet (i.e., P, Q, R) and capitalized predicate names (e.g., GREATER-THAN, MALE) are used, without subscripts, as abbreviations of the p_i^n.)
6. Variables: x_i (The x_i are frequently abbreviated by letters near the end of the alphabet without subscripts, i.e., x, y, z.)

From these symbols, the definition of a WFF can be recursively expressed:

1. Terms

   a. Each constant letter is a term.

   b. Each variable letter is a term.

   c. If f_i^n is a function letter and $t_1\ t_2 \ldots t_n$ $(n \geq 1)$ are terms then $f_i^n(t_1, t_2 \ldots t_n)$ is a term.

   d. No other expressions are terms.

2. Atomic formulae (Domain-specific Boolean-valued expressions)

   a. The propositional letters are atomic formulae.

   b. If $t_1\ t_2 \ldots t_n$ $(n \geq 1)$ are terms and p_i^n is a predicate letter, the expression $p_i^n(t_1, t_2 \ldots t_n)$ is an atomic formula.

   c. No other expression is an atomic formula.

3. WFFs

   a. An atomic formula is a WFF.

   b. If A and B are WFFs, then so are

      i $(\sim A)$ (Read as <u>not A</u>)

      ii $(A \Rightarrow B)$ (Read as A <u>implies B</u>)

      iii $(A \vee B)$ (Read as <u>A or B</u> (or both))

      iv $(A \wedge B)$ (Read as <u>A and B</u>)

   c. If A is a WFF and x is a variable, then the following are WFFs:

      i $(\forall x)A$ (Read as <u>for all x, A</u>)

      ii $(\exists x)A$ (Read as, <u>there exists x such that A</u>)

   d. No other expressions are WFF.

The parentheses shown in 3b and 3c are usually omitted in cases where no confusion will result. Some examples of WFFs, using abbreviated notation, are:

$\sim P(a,g(a,b,a))$

$P(a,b) \Rightarrow (\exists y)\ (\exists x)\ (Q(a,y) \vee S(x,y,a))$

$(LESS(a,b) \wedge LESS(b,c)) \Rightarrow LESS(a,c)$

Some examples of expressions that are not WFFs are:

$\sim f(a)$

$h(P(a))$

$Q(f(a),(P(b) \Rightarrow Q(c)))$

The semantic specification rules for the predicate calculus give a "meaning" to the WFFs by making a correspondence between symbols in the calculus and objects in the domain. The domain, D, is a nonempty set* of objects. The necessary correspondences are:

1. Associated with every constant symbol in the WFF is some particular element of D.
2. Associated with every function letter in the WFF is an n-adic function over (and into) D.
3. Associated with every predicate letter in the WFF is some particular n-place relation among the elements of D. (A relation may be considered as a function whose only values are T and F.)

The specification of a domain and these associations constitute an <u>interpretation</u> or a <u>model</u> of the WFFs.

*The domain set may be finite or infinite and, in fact, of any cardinality. This raises issues about our notation, f_i^n, p_i^n, and x_i, which restricts us to countable symbols. The issues are not addressed herein.

Given a WFF and an interpretation, we can assign a value, T or F, to each atomic formula in the WFF. These values can be used in turn to assign a value, T or F, to the entire WFF. The process by which a value is assigned to an atomic formula is straightforward: If the terms of the predicate letter correspond to elements of D that satisfy the associated relation, the value of the atomic formula is T; otherwise, the value is F. For example, consider the atomic formula:

P(a, f(b, c))

and the interpretation

D is the set of integers

a is the integer 2

b is the integer 4

c is the integer 6

f is the (two-argument) addition function

P is the relation greater-than

With this interpretation, the above atomic formula asserts that "2 is greater than the sum of 4 and 6". In this case, the assertion is false and P(a,f(b,c)) has the value F. If the interpretation is changed so that a is the integer 11, then the value is T.

The method of assigning a value to an atomic formula containing variables is not so simple. For example, the atomic formula:

$(\forall x)P(f(x,a),x)$

with the interpretation

D is the set of integers

a is the integer 1

f is the (two-argument) addition function

P is the relation greater-than

makes the assertion, "for all x in D (x any integer), x plus one is greater than x". Hence, the atomic formula has a value only under the "influence" of the quantifier. When more than one quantifier is used, then the operation of each may depend upon those further to the left. Let the interpretation be

D is the set of integers

P is the relation greater-than

Then, the WFF,

$$(\forall x)(\exists y)P(y,x)$$

asserts that for all x (integer) there exists a y (integer) which may depend upon the chosen x--such that y is greater than x. The value of this WFF is T. However, the WFF

$$(\exists y)(\forall x)P(y,x)$$

asserts that there exists a y (integer) such that y is greater than any (integer) x. The value of this WFF is F.

The values of WFFs composed using logical symbols are derived by a set of rules that are independent of the interpretation. If X is any WFF, then (~X) has the value T when X has the value F, and (~X) has the value F when X has the value T. Table 4.2 shows how the values of WFFs composed by the other logical connectives are determined from the values of the WFFs X_1 and X_2.

Given these definitions of the logical and quantifier symbols, it is easy to show that the symbols $\wedge$, $\vee$, and $\exists$ are redundant because they can be expressed in terms of the symbols ~, $\Rightarrow$ and .

$$X_1 \wedge X_2 \equiv \sim(X_1 \Rightarrow \sim X_2)$$

$$X_1 \vee X_2 \equiv (\sim X_1) \Rightarrow X_2$$

TABLE 4.2. DEFINITION OF THE LOGICAL CONNECTIVES

X_1	X_2	$X_1 \vee X_2$	$X_1 \wedge X_2$	$X_1 \Rightarrow X_2$
T	T	T	T	T
F	T	T	F	T
T	F	T	F	F
F	F	F	F	T

Several terms are used to describe properties of WFFs and the calculus itself: A WFF that has the value T for all interpretations is called valid. It can be shown, by consulting the truth table (Table 4.2) that the WFF ($P(a) \Rightarrow P(a)$) has the value T regardless of the interpretation and is therefore valid. A calculu is called decidable if there exists a general method of determining, for any WFF in that calculus, whether it is valid. Otherwise, the calculus is said to be undecidable. If the same interpretation makes each WFF in a set of WFFs have the value T, then this interpretation is said to satisfy the set of WFFs. If no interpretation exists such that each WFF simultaneously has the value T, then the set of WFFs is said to be unsatisfiable. A WFF W logically follows from a set of WFFs, S, if every interpretation satisfying S makes the value of W T. To prove W given S means to show that W logically follows from S.

The calculus described above is called the first-order predicate calculate and is known to be undecidable. That is, there does not exist a procedure for dete mining whether any arbitrary WFF is valid. If the use of quantifiers and variables is prohibited, the result is called the propositional calculus, a decidable subset of the first-order predicate calculus. A second-order predicate calculus comes about by allowing quantification of propositional letters (p_i^n

and, if desired, the f_i^n over i) in addition to the quantifications allowed in the first-order theory. This permits WFFs of the following sort:

$$(\forall P)(T(P) \Longrightarrow (\forall x)(\forall y)(\forall z)((P(x,y) \wedge P(y,z)) \Longrightarrow P(x,z)))$$

This defines transitivity of a predicate. That is, if T(P), then P is transitive. If this is to be approximated in the first-order theory, then for <u>each</u> of the possibly many transitive predicates, say P, there must be an individual axiom of the form:

$$(\forall x)(\forall y)(\forall z)((P(x,y) \wedge P(y,z)) \Longrightarrow P(x,z))$$

Obviously, the second-order form is more general and expressive than the first-order form. It is easy to see how this process of generating higher-order calculi could be continued indefinitely by allowing quantification of the higher-order predicate letters, such as T in the above example. (Such a calculus is called "omega ordered".) However, the higher-order calculi have not been used in KB or AI systems to date because no one has a clear notion of how to implement procedures for using them.

The predicate calculus provides a natural way of expressing delcarative knowledge. A KS is a collection of WFFs and the semantic rules that relate them to the domain of application. The included WFFs all have the value T and are called <u>axioms</u>. The semantic rules are usually straightforward and implicit; i.e., the abbreviated names used for the f_i^n and p_i^n are chosen in such a way that the correspondence to the domain is intuitive. Recall, for instance, the example WFF used in Section 4.1.1.4:

$$(\forall x)(\text{CHICAGOAN}(x) \wedge \text{LAWYER}(x) \Longrightarrow \text{CLEVER}(x))$$

which asserts that all Chicago lawyers are clever. New knowledge is derived and problems are solved by automatic proof procedures. The results have the status of theorems and may be used to derive further results.

Another example (taken from [P. KLAHR77]) is shown in Figure 4.6. There are four axioms: (1) Jack is the husband of Jill, (2) Jill lives in Boston, (3) if x_1 is the husband of x_2, then x_1 and x_2 are married, and (4) a married couple lives in the same place. (The illustration ignores the fact that the marriage relation is symmetric, i.e., $\mathrm{MARRIED}(x_1,x_2) \Rightarrow \mathrm{MARRIED}(x_2,x_1)$). The assertion derived is "Jack lives in Boston." The proof is shown schematically with the reasoning chain depicted by the single arrows. Thus, the proof consists of the above axioms as steps (1) through (4) followed by:

(5) Jack is married to Jill--because of (1) and (3).

(6) Jack lives in Boston--because of (2), (4), and (5).

When passing along the arrows, an association is established between the variables and/or the terms on each side of the arrow. For example, along the arrow labeled u_1, x_1 and x_2 are respectively associated with Jack and Jill, and along the arrow labeled u_2, x_1 and x_2 are respectively associated with x_3 and x_4. Each such association is called a unification. The set of all such unifications are summarized, under the heading "Variable chains", at the bottom of the figure. There are three chains in the example: (Jack x_1 x_3), (Jill x_2 x_4), and (Boston x_5). The chains are formed as equivalence classes of terms and variables so that each variable is in one and only one chain, no variable in one chain unifies with a variable in another chain, if the chain contains more than one element then each element unifies with at least one other element in the chain, and the number of chains is maximal.

In order for a proof to be proper, there are three consistency criteria: (1) at most one term can occur in an equivalence class--all variables in the class then have this value; (2) if no terms occur in a class, then there must exist an object in the domain such that all variables in the chain may legally assume that value; and (3) either condition (1) or (2) must apply simultaneously to every chain. (The definition of variable chains and other consistence criteria are more complicated than stated herein if terms are present that are built up from f_i^n, $n>0$.)

AXIOMS:
(1) HUSBAND (Jack, Jill)
(2) LIVES.IN (Jill, Boston)
(3) $(\forall x_1)\ (\forall x_2)$ (HUSBAND $(x_1,x_2) \Longrightarrow$ MARRIED (x_1,x_2))
(4) $(\forall x_3)\ (\quad x_4)\ (\forall x_5)$ ((MARRIED $(x_3,x_4) \wedge$ LIVES.IN $(x_4,x_5)) \Longrightarrow$ LIVES.IN (x_3,x_5))

$$\begin{tikzcd}
\text{HUSBAND(Jack,Jill)} \arrow[d, "u_1"] & & \text{LIVES.IN(Jill,Boston)} \arrow[ddl, "u_3"] \\
\text{HUSBAND}(x_1,x_2) \Longrightarrow \text{MARRIED}(x_1,x_2) \arrow[dr, "u_2"] & & \\
& \text{MARRIED}(x_3,x_4) \wedge \text{LIVES.IN}(x_4,x_5) \Longrightarrow \text{LIVES.IN}(x_3,x_5) \arrow[dr, "u_4"] & \\
& & \text{LIVES.IN(Jack,Boston)}
\end{tikzcd}$$

Variable chains:

$$\begin{tikzcd}
\text{Jack} \arrow[r, "u_1"] & x_1 \arrow[r, "u_3"] & x_3 \arrow[r, "u_4"] & \text{Jack}
\end{tikzcd}$$

$$\begin{tikzcd}
\text{Jill} \arrow[r, "u_1"] & x_2 \arrow[r, "u_2"] & x_4 \\
\text{Jill} \arrow[rru, "u_3"] & &
\end{tikzcd}$$

$$\begin{tikzcd}
\text{Boston} \arrow[r, "u_3"] & x_5 \arrow[r, "u_4"] & \text{Boston}
\end{tikzcd}$$

Theorem: LIVES.IN(Jack,Boston)

Figure 4.6. Proof that Jack Lives in Boston

The example shows a method of determining a value (in this case T) of the assertion, "Jack lives in Boston." This raises the natural question of how to deal with the problem, "Where does Jack live?" The method described in Nilsson (op.cit.) for solving this kind of problem is based on the resolution technique for generating proofs in the first-order predicate calculus. The method consists of two parts: (1) use resolution to generate a proof for a related problem--for our example, $(\exists x)$LIVES.AT(Jack,x); and (2) use the generated proof to find an appropriate answer to the problem--in this case, x=Boston.

The next example, known as the monkey and bananas problem, shows one method of solving a planning problem--the method is called state-space and operators. In the monkey and bananas problem, the monkey is initially at position a, there is a box at position b, and a bunch of bananas hanging above position c. The monkey cannot get the bananas unless he is standing on the box at position c. A state in this problem consists of four facts: (1) the monkey's position, (2) the box's position, (3) whether the monkey is on the box, and (4) whether the monkey has the bananas. Five operators are available to transform one state into another state: (1) the monkey can walk to any position, (2) he can push the box around the room, (3) he can climb onto the box, (4) he can climb off of the box, and (5) he can grasp the bananas. Thus, the problem can be restated as, find a sequence of operators that transform the above stated initial state (monkey at a, box at b, etc.) into the desired goal state (the monkey has grasped the bananas).

Figure 4.7 formalizes the monkey and bananas problem in predicate claculus. The major predicate letter, P, is a relation that determines valid (legal) states. $P(x_1,x_2,x_3,x_4,x_5)$ is the representation of the assertion that the monkey is at position x_1, the box is at position x_2, the monkey is on the box if x_3 is T and off the box is x_3 is F, and the monkey has the bananas if x_4 is T and does not if x_4 is F. The last term of P, x_5, is a contrivance that makes it possible to find the sequence of operators that solve the problem. Basically, x_5 is a representation of how we have come to be in this state. This will be clarified below.

Axioms:

(1) $P(a, b, f, f, i)$

(2) $(\forall s_1)(\forall x_1)(\forall x_2)(\forall x_3)(\forall v_1)P(x_1, x_2, f, v_1, s_1) \Longrightarrow P(x_3, x_2, f, v_1, \text{walk}(x_1, x_3, s_1))$

(3) $(\forall s_2)(\forall x_4)(\forall x_5)(\forall v_2)P(x_4, x_4, f, v_2, s_2) \Longrightarrow P(x_5, x_5, f, v_2, \text{pushbox}(x_4, x_5, s_2))$

(4) $(\forall s_3)(\forall x_6)(\forall v_3)P(x_6, x_6, f, v_3, s_3) \Longrightarrow P(x_6, x_6, t, v_3, \text{climbbox}(s_3))$

(5) $(\forall s_4)(\forall x_7)(\forall v_4)P(x_7, x_7, t, v_4, s_4) \Longrightarrow P(x_7, x_7, f, v_4, \text{jumpoff}(s_4))$

(6) $(\forall s_5)P(c, c, t, f, s_5) \Longrightarrow P(c, c, t, t, \text{grasp}(s_5))$

Model:

a is the original location of the monkey
b is the original location of the box
c is the location over which the bananas hang
t is "true"
f is "false"
walk, pushbox, climbbox, jumpoff, and grasp are the sequence of operators achieved by applying the operator to its arguments
i is the null sequence of operators
$P(x_1, x_2, x_3, x_4, x_5)$ is the relation - can the monkey be at location x_1, the box at location x_2, the monkey on the box (as x_3 is t or f), the monkey has the bananas (as x_4 is t or f), after the application of the operator sequence x_5?

Proof:

$$\begin{tikzcd}
(1)\ P(a, b, f, f, i) \arrow[r, "(2)"] & P(b, b, f, f, \text{walk}(a, b, i)) \arrow[dl, "(3)"'] \\
P(c, c, f, f, \text{pushbox}(b, c, \text{walk}(a, b, i))) \arrow[d, "(4)"] & \\
P(c, c, t, f, \text{climbbox}(\text{pushbox}(b, c, \text{walk}(a, b, i)))) \arrow[d, "(6)"] & \\
P(c, c, t, t, \text{grasp}(\text{climbbox}(\text{pushbox}(b, c, \text{walk}(a, b, i))))) &
\end{tikzcd}$$

Figure 4.7. The Monkey and Bananas Problem

The initial state is defined as axiom 1 in Figure 4.7 as P(a,b,f,f,i)--that is, the monkey is at a, the box is at b, the monkey is not on the box, he does not have the bananas, and this state has been achieved by applying the null sequenc of operators, i. The five operators are defined by axioms 2-6.

Axiom 2 defines the walk operator as: The monkey can walk from where he is at (x_1) to any place (x_3) as long as he is not on the box. Note, this axiom explicitly states that the box stays in the same place and possession of the bananas does not change because of walking.

Axiom 3 defines the pushbox operator as: If the monkey and the box are at the same place (x_1) and he is not on the box, then he can push the box anywhere he wishes (x_5). Further, after pushing the box, the monkey and box will be at the same location, and possession of the bananas does not change.

Axiom 4 defines the climbbox operator as: If the monkey and the box are at the same place (x_6) and he is not on the box, then he can climb onto the box. Positions and possession remain the same.

Axiom 5 defines the jumpoff operator in a similar manner.

Axiom 6 defines the grasp operator as: If the monkey and the box are both at position c (where the bananas are located), he is standing on the box, and he does not have the bananas, then he may grasp them. Further, positions do not change and he is still on the box.

The problem is solved in a manner similar to that used for the last problem. First, a related theorem is proved; namely:

$$(\exists s)(\exists x_1)(\exists x_2)(\exists x_3)P(x_1,x_2,x_3,t,s)$$

From this proof, the values of s, x_1, x_2, and x_3 are found, by mechanical means, to establish the more interesting theorem,

$$P(c,c,t,t,grasp(climbbox(pushbox(b,c,walk(a,b,i)))))$$

P's fifth argument then gives the monkey a verified plan to get the bananas: walk from location a to b, push the box from b to c, climb on the box, and, finally, grasp the bananas.

The proof of the final theorem is illustrated at the bottom of Figure 4.7. The arrows connecting the WFFs are labeled with the number of the axiom used to draw the conclusion at the end of the arrow. It should be noted that (1) the derived result is not unique--for instance, the monkey could walk all over the room, jump on and off the box, and generally engage in monkeyshines, until he gets hungry enough to get down to business; and (2) it is unlikely that any automated problem solver would find just the operator applications shown in the figure without trying several false paths. For example, after walking to the box, walk, climbbox, and pushbox are all allowable operations. Some sort of trial-and-error or heuristic method would be necessary to determine what step should be next tried. These comments apply also to the previous example.

This section has been relatively lengthy because predicate calculi are the best theoretically understood and among the oldest used techniques for representing knowledge in a computer. We conclude by summarizing some of the characteristics and merits of this formalism.

The predicate calculus is clearly a declarative form of knowledge representation. At such, it is modular and reversible. The chunk size (i.e., a WWF axiom) is variable. In practice, however, one would never expect to encounter an axiom that comprised more than a half dozen or so clauses because it is formulated by a human expert. If it were any larger, its intelligibility (to a human) would be greatly diminished. On the other hand, representing procedural knowledge in the predicate calculus is at best difficult. (Consider encoding, as WFFs, the knowledge embedded in the example shown in Figure 4.5; further consider the difficulty of a human domain expert's doing the encoding.)

Another disadvantage of the predicate calculus is that the entire set of axioms must be consistent. That is, if a WFF, W, logically follows from the axioms, then the WFF ~W does not. If an axiom set is not consistent, it is easy to demonstrate that every WFF has the value T. The onus of maintaining consistency in a KS containing WFFs is a major problem. For one thing, it makes it impossible to include heuristic and possibly contradictory rules of thumb and other sorts of expert knowledge, and thorough formalization may not be within the state of the art of the application area.

One advantage of using the predicate calculus is that automatic procedures are known such that if N follows from the axioms, then it can eventually be proved. However, no theoretical upper bound exists on the amount of time it will take to find a proof. One can attempt to evade this problem by trying to prove W and W in parallel and quitting when either proof succeeds. However, since the first (and higher) order predicate calculi are undecidable, it may therefore be the case that neither proof process terminates. Hence, one must impose some sort of resource limitation on the effort expended to derive or prove something from the axioms.

Another characteristic of predicate-calculus representations is demonstrated by the example of Figure 4.6--namely, there are two broad categories of axioms. First, there are specific facts such as "Jack is Jill's husband" or "Jill lives in Boston." Second, there are general assertions such as "married couples live at the same place." In any actual application domain, the number of facts will be overwhelming when compared to the size of the axiomatic base for a branch of mathematics or logic. The result is impractically slow proof procedures or the use of different methods, in the CE, to handle facts and general knowledge. For a good discussion of this problem and a proposed approach to using predicate calculus in practical areas, see P. Klahr (op. cit.).

Many publications exist in the area of using the predicate calculus for a knowledge representation and theorem-proving techniques for solving problems. The best introduction to the area is Nilsson (op. cit.). A sampling of other work is to be found in [BLEDSOE73], [FIKES71], [GERLERNTER63], [GREEN68], [P. KLAHR77 and 75], [POPLE73], [ROBINSON70 and 65], [SANDEWALL70], and [YATES70].

4.1.2.4 Production Rules

Production rules have been used as the prinicpal method of representing knowledge in many (if not most) of the highly successful KB systems--for example, MYICIN and DENDRAL. Therefore, their importance is immediate. This section describes production rules using several examples and introduces terminology and issues.

A _production rule_ is a specification of a conditional action. It consists of a _left hand side_ (LHS), also called the condition or the antecedent, which describes a situation, and a _right hand side_ (RHS), also called the action or consequence, which describes something that may legally be done in a situation described by the RHS. For example, in "If you are outdoors and it is raining, then open an umbrella." The conditions are (1) being outdoors and (2) rain. The action is to open an umbrella.

Production system is an ambiguous term. Its original meaning was a KS comprised of production rules. In current usage, the meaning of the term production system refers to a three-component entity: (1) a collection of production rules, (2) a workspace, and (3) a control mechanism. The production rules are represented by some agreed-upon syntax, by means of which the LHS and RHS are built up from a set of primitives and symbols that correspond to objects, functions, and predicates in a domain. (See the previous section for a description of similar correspondence rules when using predicate calculus formulae.) The workspace,* also called the _data base_, contains the total description of the system's current state or situation. The LHS of a rule describes, or is matched against, the contents of the workspace. If a production is _applied_, i.e., its LHS matches and its RHS is executed, then the RHS actions modify the workspace.

*In systems used to do psychological modeling (e.g., PSG [NEWELL73]), the workspace is limited by a fixed maximum number of entries, usually 7 to 12, called the short-term memory.

The control mechanism generally has the form shown in Figure 4.8. The first part, represented by the FOR loop, builds the conflict set--the set of all production rules whose LHSs are satisfied. If the conflict set is empty, then processing is terminated, and the result is the contents of the workspace. However, if the conflict set is not empty, then the conflict-resolution strategy selects one member of the conflict set and the RHS of the selected production rule is executed. The entire cycle is then repeated until the termination condition is reached.

```
loop: conflict_set←empty;
      FOR p IN set_of_production_rules
          DO IF left_hand_side(p) matches work_space
                THEN add p to conflict_set;
      IF null conflict_set THEN terminate;
      q←resolution_of(conflict_set);
      execute right_hand_side(q);
      GO TO loop;
```

Figure 4.8. Control Mechanism for Production Systems

Several conflict-resolution strategies have been used or proposed. Among them are:

rule order--There is a complete ordering of all production rules. The rule in the conflict set that is highest in the ordering is chosen.

rule precedence--A precedence network (which may contain cycles) determines an ordering.

generality order--The most specific rule is chosen.

data order--Elements of the workspace are ordered. The rule chosen is the one whose LHS references the highest-ranking workspace element(s).

regency order--Execute the rule in the conflict set that was most (least) recently executed, or the rule in the conflict set whose LHS references the most (least) recently referenced element(s).

non-deterministic--Execute every rule in the conflict set as if it were the only member. Computation stops when any path terminates. This is equivalent to backup in program schemes--see Section 4.1.2.1.

The original use of a production system formalism to perform or simulate computational tasks can be traced to [Post 36]. The example shown in Figure 4.9 closely approximates the original notation. There are three rules--named I, M, and F for convenience (the names are not part of the formalism). The rules are applied to a workspace that is an ordered sequence of symbolic characters made up from the two-character alphabet consisting of 1 and #. The form of each rule is

$$\text{LHS} \Longrightarrow \text{RHS}$$

Both the LHS and RHS have the same form, namely, an alternating sequence of $\$_i$ and strings of characters (including empty strings) from the alphabet. A character string matches any identical string in the workspace, and a $\$_i$ matches any string (including the empty string). The LHS is said to match if the sequence of $\$_i$ and character strings match the entire workspace in their lexical order. When a production is applied, the entire workspace is replaced by a string defined by the RHS. The RHS defines the character sequence that is shown, with substitutions for each $\$_i$ by the characters matched by the LHS. For example, rule I matches the work space

#111#1111#

with $\$_1 \equiv 111$ and $\$_2 \equiv 1111$. Thus, the RHS, $\#\$_1\#\$_2\#\#$, generates or defines the new workspace as #111#1111##. In the figure, the process of applying the rules is shown until a final result is produced. Arrows connecting consecutive workspace

PRODUCTION RULES

(I) $\#\$_1\#\$_2 \Longrightarrow \#\$_1\#\$_2\#\#$

(M) $\#1\$_1\#\$_2\#\$_3 \Longrightarrow \#\$_1\#\$_2\#\$_2\#\$_3\#$

(F) $\#\#\$_1\#\$_2 \Longrightarrow \#\$_2\#$

EXAMPLE USE

$$\begin{tikzcd}
\#111\#1111\# \arrow[d, "I"] \\
\#111\#1111\#\# \arrow[d, "M"] \\
\#11\#1111\#1111\# \arrow[d, "M"] \\
\#1\#1111\#11111111\# \arrow[d, "M"] \\
\#\#1111\#111111111111\# \arrow[d, "F"] \\
\#111111111111\#
\end{tikzcd}$$

Figure 4.9. Use of Production Rules to Multiply

representations are labeled with the name of the applicable rule. The conflict-resolution strategy is to use either rule M or rule F instead of rule I whenever there is a conflict. If the original input is a sequence of n 1's and a sequence of m 1's delimited by #, i.e.,

$$\#1_1 1_2 \cdots 1_n \# 1_1 1_2 \cdots 1_m \#$$

then the result is a sequence of nm 1's bracketed by a pair of #'s (e.g., unary multiplication).

The next example (see Figure 4.10) shows a production system that translates arithmetic infix expressions, composed from variable names A, B, and C and operators + and *, into the equivalent Polish prefix expression. A Polish prefix expression is either a variable name or a list whose first element is a function or operator and whose subsequent elements are arguments. Thus, $(+\ \text{exp}_1\ \text{exp}_2)$ is the Polish equivalent of the infix expression $\text{exp}_1+\text{exp}_2$. This example is based upon standard techniques for constructing compiler-compilers; see, for example, [SCHORRE64].

In the example, each production rule has the form

$$\text{pat}_1 \cdots \text{pat}_n \rightarrow \text{name}[\text{part}_1 \cdots \text{part}_m]$$

The LHS of the rule is the sequence $\text{pat}_1 \cdots \text{pat}_n$, and the RHS is $\text{name}[\text{part}_1 \cdots \text{part}_m]$. Each pat_i is either one of the input character set (i.e., A, B, C, +, or *) or a name appearing in some rule's RHS. The LHS is satisfied whenever each pat_i exactly matches an entity in the workspace and the entities in the workspace are contiguous and in the same order as the appearance of the pat_i. Note that it is not necessary to match the entire workspace for the LHS to be satisfied. The RHS specifies that the matched portion of the workspace is to be replaced by name. Associated with name is a value, which is $\text{part}_1 \cdots \text{part}_m$. (The value is not used in determining subsequent LHS matches.) Each part_i is either a character constant (from the input

PRODUCTION RULES

(v1)	A→var[A]
(v2)	B→var[B]
(v3)	C→var[C]
(t1)	var→term[1]
(t2)	term*term→term[(*1 3)]
(a1)	term→exp[1]
(a2)	exp+exp→exp[(+ 1 3)]

EXAMPLE USE

$$\begin{tikzcd}
\text{A+B*C} \arrow[d, "\text{v1-3}"] \\
\text{var[A]+var[B]*var[C]} \arrow[d, "\text{t1}"] \\
\text{term[A]+term[B]*term[C]} \arrow[d, "\text{t2}"] \\
\text{term[A]+term[(* B C)]} \arrow[d, "\text{e1}"] \\
\text{exp[A]+exp[(* B C)]} \arrow[d, "\text{e2}"] \\
\text{exp[(* A (* B C))]}
\end{tikzcd}$$

Figure 4.10. Use of Production Rules to Translate

set or, in this case, one of the additional characters "(" or ")"), or an integer. If $part_i$ is an integer, say k, then it specifies the value associated with pat_k. The control mechanism uses the first rule that applies in the given order--v1 through a2.

The bottom portion of the figure traces the chain of modifications to a workspace whose initial content is A+B*C. When activity terminates, the workspace is the single part, exp, and the value associated with this part is (+ A (* B C)), the Polish-prefix equivalent of the initial infix expression. Consecutive workspace configurations are connected by labeled arrows; the label is the name of the production rule that caused the modification. (When several modifications are made by the same (or similar) rules, e.g., v1, v2 and v3, then all transformations are shown simultaneously.) When the workspace contents are

term[A]+term[B]*term[C]

rule t2 is applied. The table below identifies the components of the rule and the matched parts and values for this rule application:

pat_1	term	term	[B]
pat_2	*	*	
pat_3	term	term	[C]
name	term		
$part_1$	(		
$part_2$	*		
$part_3$	1		B
$part_4$	3		C
$part_5$	)		

Thus, this rule replaces term[B]*term[C] with term[(* B C)]. Therefore, the entire workspace becomes

term[A]*term[(* B C)]

The rest of the example can be analyzed in a similar manner.

The last example is a production system that assists the service manager and mechanics in an automobile repair agency (see Section 2.1). The scenario for using this system is the arrival of a customer at the agency. He reports the symptoms and problems to the service representative, who then enters this infor mation into the system. The system has at its disposal a data base of past problems, repairs, and services performed on the vehicle, and a KS of production rules that describe cause-and-effect relationships among the performance characteristics and measurable attributes of an automobile. Using the reporte information, the past-history data base, and the KS, a diagnostic and repair pl is formulated and implemented.

Figure 4.11 gives a few of the production rules that might be present in such a system. Each rule is named; however, the rule names are used only for convenience. The format of the rules is

IF lhs_1 C_1 $lhs_2 \cdots C_{n-1}$ lhs_n

THEN $rhs_1[p_1]$ K_1 $rhs_2[p_2] \ldots K_{m-1}$ $rhs_m[p_m]$;

where the C_1 and K_1 are the connectives AND and OR. The LHS is everything between the keywords IF and THEN, and the RHS is everything following the THEN. Each lhs_i is an observable or measurable condition predicate, e.g., that the tension of the fan belt is low or the engine is overheating. Each $rhs_i[p_i]$ is a condition, rhs_i, that will follow with certainty or probability, p_i. Thus,

```
R1  IF   fan belt tension is low
    THEN alternator output will be low [.5] AND engine will overheat [.2];

R2  IF   alternator output is low THEN battery charge will be low [.7];

R3  IF   battery charge is low THEN car will be difficult to start [.5];

R4  If   automatic choke malfunctions OR automatic choke needs adjustment
    THEN car will be difficult to start [.8];

R5  IF   battery is out of warranty THEN battery charge may be low [.9];

R6  IF   coolant is lost OR coolant system pressure cannot be maintained
    THEN engine will overheat [.7];

R7  IF   there is a high resistance short AND fuse is not blown
    THEN battery charge will be low [.8];

R8  IF   voltage regulator output is high
    THEN battery will boil off fluid [.3];

R9  IF   battery fluid is low THEN battery charge will be low [.4];
```

Figure 4.11. Production Rules for Automotive System KS

rule R1 says that, if the tension of the fan belt is low, then there are two possible consequences:

(1) That about one-half of the time the output of the alternator will be low, and

(2) About one-fifth of the time the engine will overheat.

The other production rules, R2-R9, are interpreted in a similar manner.

A fact file in the system is shown in Figure 4.12. The information included for each observation or measure is the agent from whom to gather data and the relative difficulty (or cost) of gathering the data. There are four possible agents for data gathering: (1) the customer (Cust), (2) the historical data base,, (3) inspection by the service manager (SrvM), and (4) measurement by the mechanic (Mech). The difficulty information will be combined with the confidence factors in the production rules to formulate the most cost-effective and timely plan for the needed diagnostics and repairs.

Assume that a customer arrives at the agency with the vague complaint that his car is hard to start. The service manager enters this information, including appropriate customer and vehicle identification. The system then grows a structure similar to that shown in Figure 4.13. The boxes are labeled with observable or measurable symptoms and are connected by arrows labeled with the names of the production rule they represent. To the far right of each of the unknown values (e.g., the box labels, such as battery-fluid level), the associated agent and relative difficulty are listed. At this point, the system would check the data base for information about the battery's warranty. If nothing decisive was found, then the customer would be asked whether the car was running hot, and the service manager would continue to make on-the-spot observations. Diagnostic procedures for causes not ruled out by the procedure to date would then be placed on an ordered schedule for a mechanic. The ordering would be based upon (1) cost effectiveness--a function of test difficulty, estimated

OBSERVATIONS	AGENT	DIFFICULTY
Alternator output level	Mech	4
Battery charge level	Mech	3
Battery fluid level	SrvR	2
Choke adjustment	Mech	5
Choke function	Mech	5
Coolant level	SrvR	2
Coolant system pressure	Mech	5
Difficulty to start	Cust	1
Engine temperature	Cust	1
Fan belt tension	Mech	3
Fuse condition	SrvR	2
Short in electric system	Mech	8
Voltage regulator level	Mech	4
Warranties	Data Base	0

Figure 4.12. Data Gathering Procedure Fact File

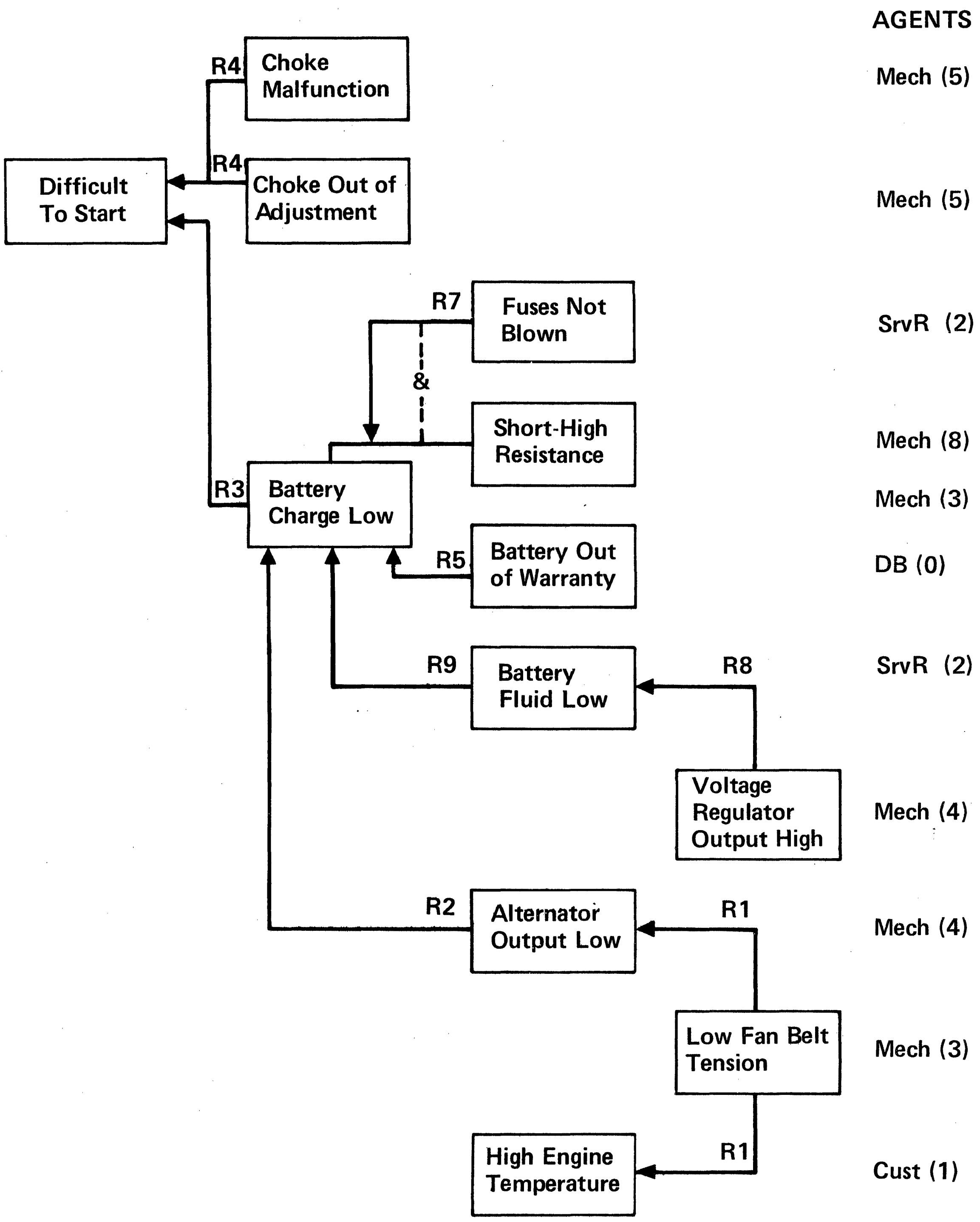


Figure 4.13. Example Flow in Auto Diagnostic System

probability of being necessary, and ability to eliminate other tests; and (2) availability of resources--specialty mechanics and test equipment.

The structure shown in Figure 4.13 was grown by an algorithm called back chaining. A condition--in this case, "difficult to start"--is taken as a given, and the goal of the system is to find the cause(s). Note that the production rules state causes, then effects. Thus, the rules are used as if the knowledge possessed a kind of symmetry. The back-chaining algorithm is

(1) Find all rules that have the initial or derived condition as their consequence--in this instance, rules R3 and R4.

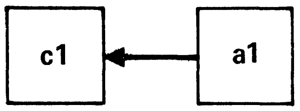
(2) Call the antecedents of these rules' derived conditions.

(3) Repeat steps (1) and (2), and terminate when no more can be done.

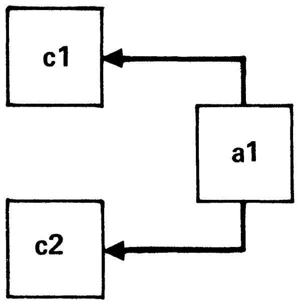
Figure 4.14 graphically shows the kind of structure grown for each kind of rule format. In each example in the figure, c1 is the initial or a derived condition.

Rule E1 is the simplest; a1 is added to the set of derived conditions. Rule E2 states that if a1 is the case, then both c1 and c2 ought to follow. Thus, a1 is a derived condition, and c2 may or may not be considered a derived condition, depending upon the particular strategy used by the system. Rule E3 is really equivalent to two independent rules "IF a1 THEN c1" and "IF a1 THEN c2." Therefore, a1 is added to the set of derived conditions, and the c2 part is ignored. Rule E4 states that both a1 and a2 must occur to support the conclusion, c1. Therefore, both are derived conditions. If either a1 or a2 is found to not hold, then the search for support for the other can be discontinued. Rule E5 is equivalent to the two separate rules "IF a1 THEN c1" and "IF a2 THEN c2." Thus, both a1 and a2 are added to the set of derived conditions.

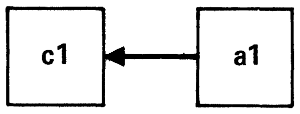
E1 **IF a1 THEN c1**



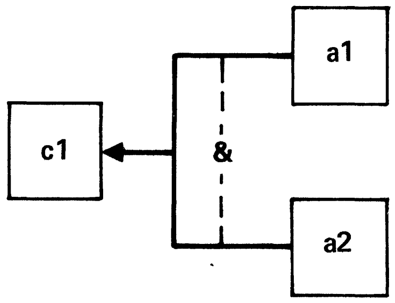
E2 **IF a1 THEN c1 AND c2**



E3 **IF a1 THEN c1 OR c2**



E4 **IF a1 AND a2 THEN c1**



E5 **IF a1 OR a2 THEN c1**

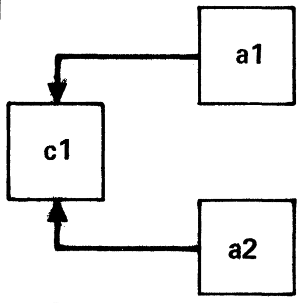


Figure 4.14. Back Chaining

In this example and discussion, we have omitted several problems that can arise. For example, suppose that rule R8 (in Figure 4-11) had been written more accurately as the two rules

R8′ If voltage regulator output is high
THEN the battery will overcharge

R8′′ IF battery is overcharged
THEN battery will boil off fluid

With these new rules, a fragment of structure shown in Figure 4.13 would be replaced by that shown in Figure 4.15. Now, the interesting conclusion is that a high battery charge implies a low battery charge. This is an apparent contradiction, since both conditions cannot hold at the same time. This kind of situation can often arise in unpredicted ways if the system contains many rules--more than a few dozen. In this instance, the contradiction is more apparent than real--i.e., the charge of the battery will oscillate between high and low as the battery fluid is replaced and boils off, respectively. So, in a sense, there is a missing rule of the form that adding fluid to a battery whose charge and fluid levels are low will probably allow the battery to return to normal conditions. However, to handle this kind of situation in general, it is necessary that the control mechanism or CE have some knowledge about how to proceed when faced with apparent conflicts and contradictions. One virtue of production systems is that ad hoc knowledge may be relatively easily incorporated in the system to handle this.

Another issue not yet raised is that the structure shown in the above figures may actually be a graph rather than a simple tree. This may arise from several causes. For example, assume that high engine temperature caused battery fluid to boil off (call this Rule R10). Then Figure 4.16 would show a fragment of the resulting graph. Another cause of graph structure is loops within the rules; the simplest cause is two conditions, either of which can cause the other to occur. For example, high engine temperature causes coolant to boil off, and low coolant level will cause the engine to overheat. A major problem

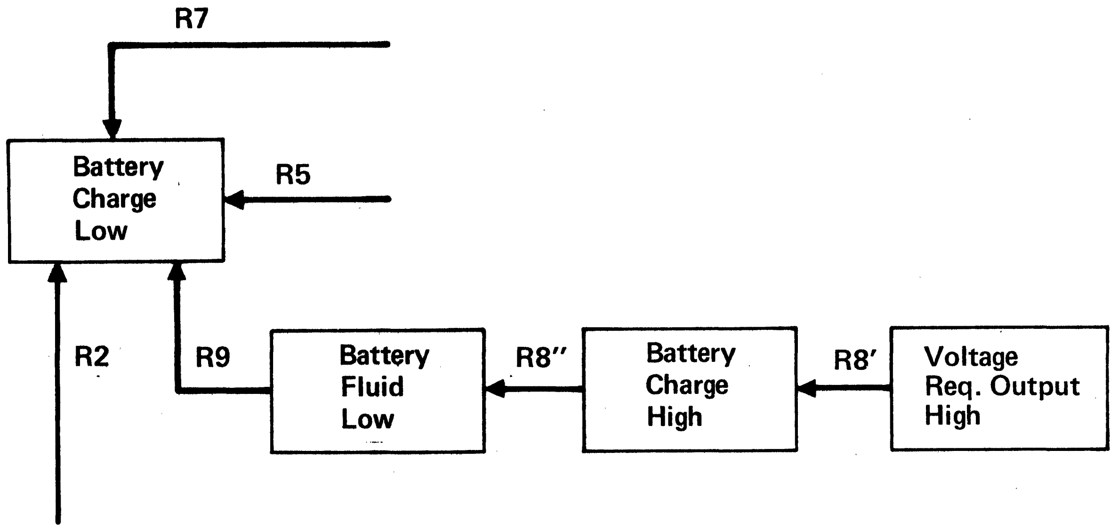


Figure 4.15. Fragment of Graph Structure

with the graph structure that occurs is development of appropriate mathematical techniques to handle the generation of the confidence (or probability) factors used to guide the system.

The remainder of this section discusses some of the key features and characteristics of the use of production rules to make a KS. Figure 4.17 summarizes the following discussion in showing some of the interactions among the characteristics.* An arrow labeled with a "+" means that the source characteristic enhances the destination characteristic; the opposite is true for arrows labeled with "-".

Rules as primitive actions. In a production system, the production rule is the knowledge chunk. The smallest grain of behavior in which the system can engage is the application of a single rule.

Indirect limited interaction channel. Rules are constrained to see and modify only the workspace. They cannot "call" each other as subroutines. To achieve

*The discussion and figure are based, with some modifications, on [DAVIS75].

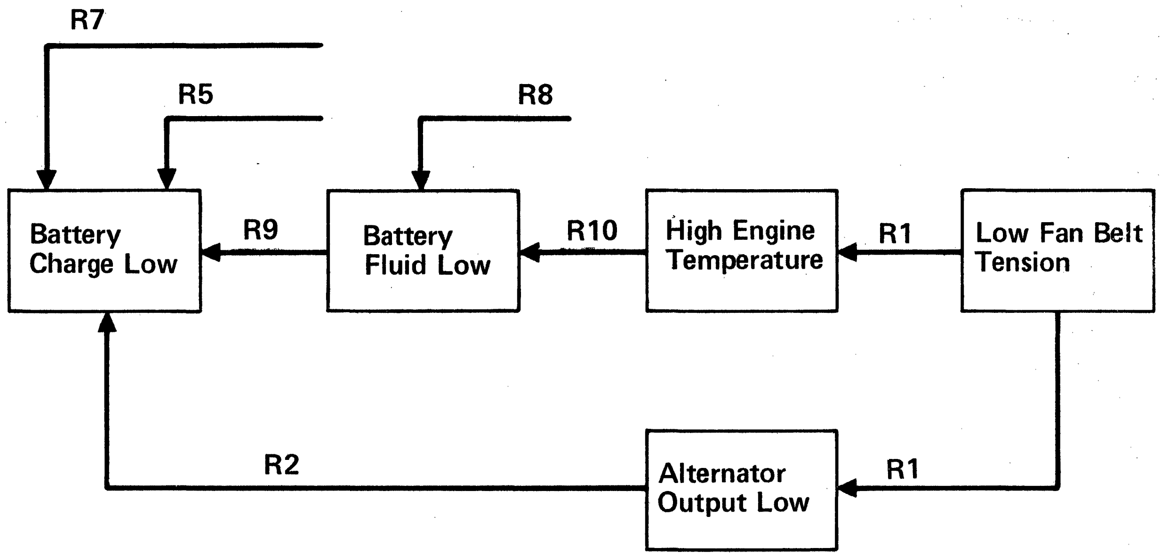


Figure 4.16. Fragment of Graph Structure

the effect of a call, one rule must leave a unique message in the workspace that is recognized only by the invoked rule. This becomes more difficult to do as the number of rules increases and is a method that quickly destroys the major benefits of using production systems, such as independence of the knowledge chunks.

Constrained format. The LHS and RHS of the rules are normally built from a simple set of primitives through a straightforward syntax. Even though some systems allow programmer-supplied predicates and procedures to be invoked by the rule's LHS and RHS, some restrictions are obeyed: (1) the operation of the LHS will not modify the workspace, and (2) operation of the RHS will perform only conceptually simple actions. These restrictions, like those mentioned in the previous paragraph, are accepted so that the major advantages of the production (such as ability to explain results) will not be compromised.

Machine readability. Because of the constrained format of production rules, machine readability is enhanced. Also, compilation by the knowledge acquisition mechanism (such as computing links between one rule's consequence and another rule's antecedent) is simplified.

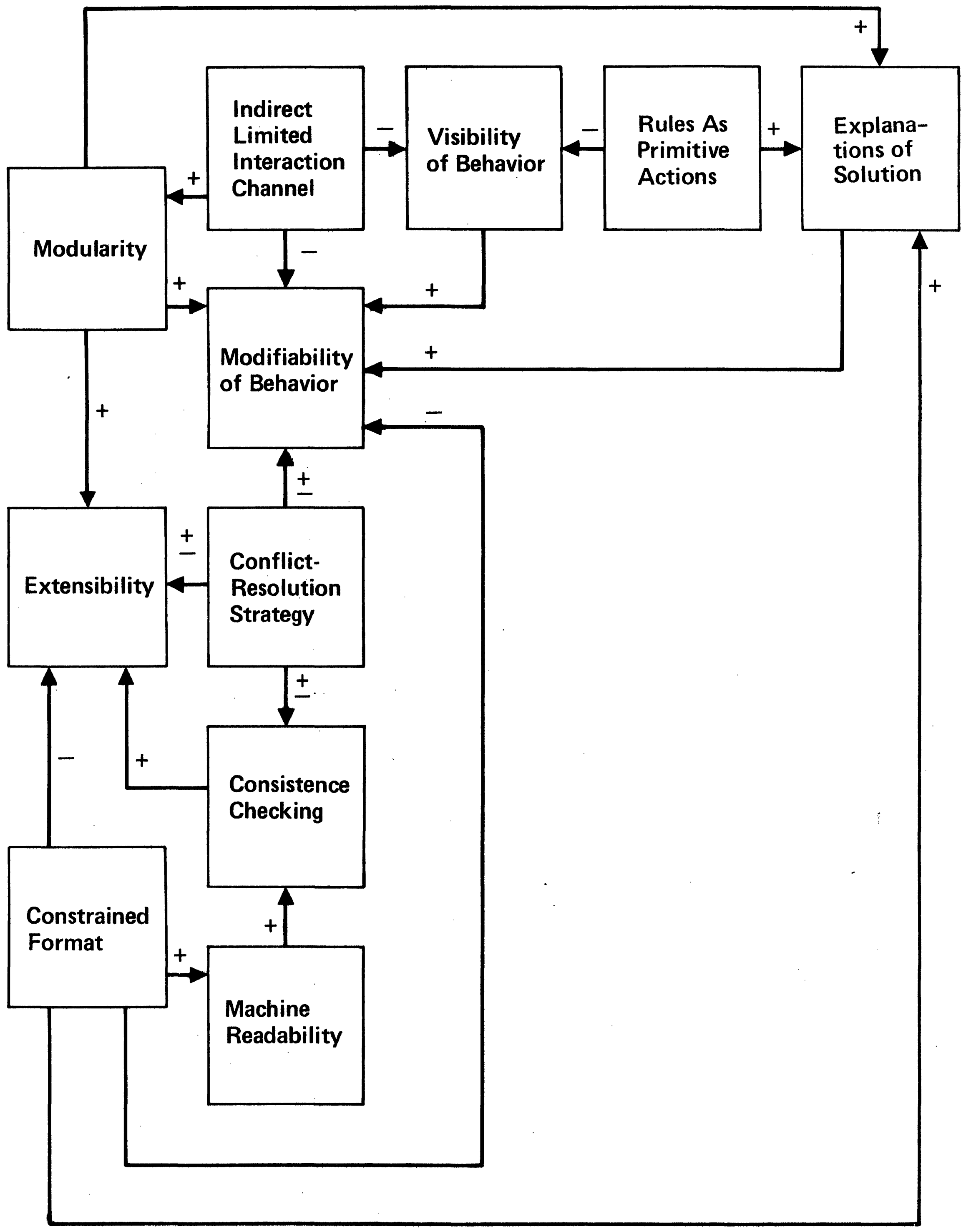


Figure 4.17. Facets of Production Systems

Modularity. Since direct interaction among rules is constrained, it is possible to modify rules, delete rules, and add new rules as necessary because other rules are not directly dependent upon the rules that are changed or added.

Extensibility. Extensibility is a corollary of modularity. The ability to augment the system to perform in an expanded domain is obviously enhanced by the modularity and low interaction among the original rule set. On the other hand, extensibility may be hampered because of format constraints if the expanded domain necessitates the use of a more robust set of primitives.

Visibility of behavior flow. The issue here is not the external manifestations of the system's performance; rather, it is the ability to understand how the system proceeds to a solution by a step-by-step analysis of its internal working. In tracing a production system with a large rule set, one may be surprised at how often it goes off on nonproductive tangents before exhibiting reasonable goal-directed activity. Several things account for this. One is that application of a single rule is the system's step size, and all rules get an opportunity to examine intermediate results. Therefore, even when one rule "knows" the rule most likely to continue on a path to a solution, the limit on direct rule-to-rule communication inhibits the system from focusing attention. One method of increasing goal-directed behavior in a production system is the use of higher-level, strategic and tactical rules to guide the conflict-resolution strategy. For an interesting discussion of this approach, see [ENGLEMORE77] and [DAVIS76].

Modifiability of behavior. This is a problem closely akin to extensibility. However, the issue here is the ability to modify the rules so that the system focuses attention better or more quickly. This is obviously aided by modularity of the rule set and hindered by the problems that arise when explicit control and sequencing are desired in a production system.

Explanation of solution. A production system can (and usually does) explain and validate its solutions to problems by displaying the rules it used to derive the solutions. Because the rules are of a situation/conclusion form and are a reasonable chunk size, the explanation method is appealing. However, if the behavior is too erratic (see the paragraph above on visibility of behavior flow), the system may provide an excellent explanation and defense of a seemingly silly activity. Modularity of the rules also contributes to the acceptability of the explanation because each rule is reasonably well self-contained.

Conflict-resolution strategy. Conflict-resolution strategy has an effect on the ability to extend the system and/or modify its behavior. For example, if the rules are ordered, it may take a great deal of work to insert a new rule or modify an old one, because the ordering enforces an implicit dependency among the members of the rule set.

Consistency checking. Some control mechanisms will not work properly if the rule set can generate inconsistent results. (See the example shown in Figure 4.15.) For such systems, it is desirable that the knowledge-acquisition mechanism be able to determine whether such conflicts can arise. This endeavor is aided by the simplicity of format and ease of machine processing, but can be difficult (if not impossible) with some conflict-resolution strategies because the strategy determines whether the conflict can ever arise and, is so, how it will be resolved.

Some works that describe the philosophy and theory of production systems are: [CHOMSKY63], [DAVIS76 and 75], [GALLER70], [HEDRICK76 and 74], [J. McDERMOTT76a and 76b], [MINSKY67], [J. MOORE73], [NEWELL76a], [POST43 and 36], and [VERE77]. Some work on the use of production systems for psychological modeling are [MORAN73a], [NEWELL72b], and [WATERMAN75, 74 and 70].

Works using production system models in the field of medical applications are: [DAVIS77 and 76], and [SHORTLIFFE76, 75a, 75b, and 73]. Some works relating to chemistry, molecular structure, and genetics are [BUCHANAN76a, 76b, 72, and 69], [ENGLEMORE77], [FEIGENBAUM71], [LEDERBERG68], [MARTIN75], and [MICHIE73]. Some other works about usage of production rules are: [ANDERSON76a and 76b], [BARNETT76b], [COLLINS76], [EVANS64], [FLOYD61], [FORGY76], and [RYCHENER76 and 75].

4.1.2.5 Semantic Networks

A semantic network is a method of representing declarative knowledge about the relations among entities. The major application has been to embody non-syntactic knowledge (e.g., semantics and pragmatics) in natural-language-understanding systems, but this has not been the only use. Because of their inherent generality and naturalness, semantic networks have been used to represent highly interrelated information that cannot be properly processed by standard data (base) management techniques.

A semantic network is a KS. It is built up from knowledge chunks that are instances of a relation. The format of a chunk is

$$\text{rel}(a_1 \cdots a_n)$$

where rel is a relation name and the ordered tuple, $(a_1 \ldots a_n)$, is in the relation rel. For example,

ISA(DOG,MAMMAL)

means (DOG, MAMMAL) is a member of the relation ISA. ISA is conventionally taken to be the relation, more-specific-example-of. Thus, the above is the representation of the fact that a DOG is a specific kind of MAMMAL. For the example,

BETWEEN(2,1,5)

the interpretation is the obvious one; namely, 2 is between 1 and 5.

Figure 4.18 shows a semantic network (or "net"). The top of the figure lists the instances of relations using the relation names TEMP, LOC, COLOR, SIZE, ISA, and BETWEEN. (The latter two are defined as above.) TEMP(a,b) means a is the temperature of b; LOC(a,b) means a is located at b; COLOR(a,b) means that a is the color of b; and SIZE(a,b) means a is the size of b. The knowledge in a semantic net is given meaning, as demonstrated here, by defining the relation names and other symbols used in the instances of relations, in terms of external entities. Fortunately, the correspondences of names to external entities can be made highly mnemonic by careful choice of the names.

The graph in the middle of Figure 4.18 shows exactly the same knowledge that is in the set of instances at the top of the figure. The entity names are connected by arrows labeled with appropriate relation names. For example, the instance,

ISA(DOG,MAMMAL)

produces the graph fragment

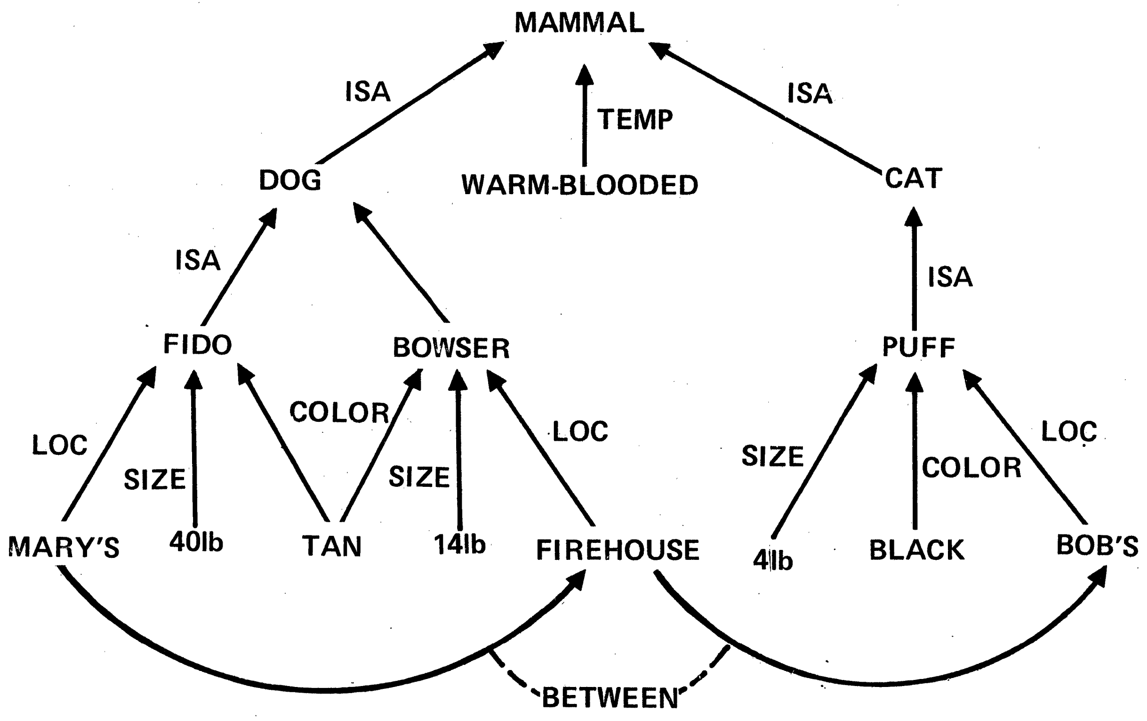
$$\text{DOG} \xrightarrow{\text{ISA}} \text{MAMMAL}$$

Production of graph fragments for other than binary relations is more difficult but still straightforward--see the example of BETWEEN in the figure.

RELATIONS

TEMP (WARM-BLOODED, MAMMAL)
ISA (DOG, MAMMAL) ISA (CAT, MAMMAL)
ISA (FIDO, DOG) ISA (BOWSER, DOG) ISA (PUFF, CAT)
LOC (MARY'S, FIDO) LOC (FIREHOUSE, BOWSER) LOC (BOB'S, PUFF)
COLOR (TAN, FIDO) COLOR (TAN, BOWSER) COLOR (BLACK, PUFF)
SIZE (40lb, FIDO) SIZE (14lb, BOWSER) SIZE (4lb, PUFF)
BETWEEN (MARY'S, FIREHOUSE, BOB'S)

SEMANTIC NETWORK



RULES OF INFERENCE

$\mathrm{ISA}(x,y) \wedge \mathrm{ISA}(y,z) \Longrightarrow \mathrm{ISA}(x,z)$
$\mathrm{SIZE}(x,y) \wedge \mathrm{SIZE}(u,v) \wedge x < u \Longrightarrow \mathrm{SMALLER}(y,v)$
$\mathrm{ISA}(x,y) \wedge r(u,y) \Longrightarrow r(u,x)$

Figure 4.18. Example Semantic Network

The external format of knowledge in a semantic network is usually very similar to the one used herein with the addition of a capability to factor common parts--for example, something like:

ISA({DOG CAT3}, MAMMAL) or

SIZE{(40 lb, FIDO)(14 lb, BOWSER)(4 lb, PUFF)}

However, the internal storage of the semantic network closely corresponds to the graphical presentation--that is, a network structure built using pointers and list structures. The explicit connections among the entities enhances efficiency of programs that search through the semantic network.

The bottom of Figure 4.18 gives some examples of <u>inference rules</u> for the semantic network. The format of the rules is well-formed formulae from the predicate calculus. It is also possible to represent the inference rules as a production system. This has the advantage of allowing procedural knowledge to be used to test for complex enabling conditions that might be difficult to express as WFFS. Variables, written as small letters, are assumed to be universally quantified. The first rule says that (for all x, y, and z) if x is a y and y is a z, then x is also a z. An example of this is: FIDO is a DOG and a DOG is a MAMMAL; therefore, FIDO is a MAMMAL. The second rule says that if y and v are two entities that "have" SIZE, and the size of y is less than the size of v, then y is SMALLER than v. Thus, the instance of the relation,

SMALLER(PUFF,BOWSER)

This inference rule defines instances of relations whose names do not appear explicitly in the semantic network. Contrast this to the first rule above, which states that ISA is transitive.

The last example inference rule says that, if x is a y, and y has a property conferred by the binary relation, r, then x has the same property conferred by r, i.e., properties are inherited. Thus, FIDO is a MAMMAL (by the transitivity of ISA--first rule), and a MAMMAL has the property, WARM-BLOODED (conferred by the relation TEMP), therefore, FIDO is WARM-BLOODED. Formally,

$$\mathrm{ISA(FIDO,DOG)} \wedge \mathrm{ISA(DOG,MAMMAL)} \Longrightarrow \mathrm{ISA(FIDO,MAMMAL)}$$

$$\mathrm{ISA(FIDO,MAMMAL)} \wedge \mathrm{TEMP(WARM\text{-}BLOODED,MAMMAL)} \Longrightarrow \mathrm{TEMP(FIDO,WARM\text{-}BLOODED)}$$

However, the indiscriminate use of the third rule can cause derivation of incorrect results. For example,

$$\mathrm{ISA(DOG,MAMMAL)} \wedge \mathrm{ISA(CAT,MAMMAL)} \Longrightarrow \mathrm{ISA(DOG,CAT)}$$

In order to avoid this kind of problem, it is necessary to have some (non-syntactic) knowledge about the relations to which inference rules are applied. One possible solution is to have a rule, such as the third example rule, for each relation that is inheritable. (The variable, r, is replaced by the relation name in the rule.) Another solution is to embed the interface rules in the CE along with the necessary ad hoc knowledge to avoid the problems. Both approaches cause problems, however, if the number of relations occurring in the semantic network is large or if the relation set can be modified or expanded.

A more general appraoch to the problem treats relation names and entity names more uniformly. For example, temperature is defined as an inheritable property by an instance like

INHERITABLE(TEMP)

The third inference rule is then rewritten as

$$\mathrm{ISA}(x,y) \wedge r(u,y) \wedge \mathrm{INHERITABLE}(r) \Longrightarrow r(u,x)$$

With this approach, relations can be arguments to relations, and hence have the same properties as other entities. This is similar to higher-order rules in the predicate calculus (see section 4.1.2.3). There are several advantages to this. For one thing, instances such as

ISA(TAN,COLOR)

are allowed and provide a natural method of delineating legal values in a relation and, therefore, of enhancing error detection and consistency checking. Another advantage is improved flexibility and expandability. The major drawback is a loss in run-time efficiency.

Another choice and tradeoff about a semantic network is the decision about which relations and which instances in the relations ought to be stored explicitly and which should be computed via the inference rules. Explicit storage costs space, and inference rules cost computation time. For all but very small semantic networks, some inference rules are necessary because the number of instances of relations can grow in a highly nonlinear way; for the example in Figure 4.18, the number of instances of the relation, SMALLER, grows as a quadratic function of the number of DOGs and CATs.

A technique often used with semantic networks is to make a (somewhat arbitrary) distinction between general knowledge and specific knowledge and to store the two in a different manner. Specific knowledge has the general characteristic of being "low" in the tree--as shown in the middle of the figure. This means (1) there are few if any chains below it; (2) therefore, properties have simple values; (3) most entities in the same general classification have all and only a known set of properties; and (4) there are a large number of entities in a general class. For our example, the specific knowledge can be displayed tabularly as

ENTITY	ISA	SIZE	COLOR	LOC
FIDO	DOG	40lb	TAN	MARY's
BOWSER	DOG	14lb	TAN	FIREHOUSE
PUFF	CAT	4lb	BLACK	BOB's

The above conditions make it likely that (1) the specific knowledge can be gathered into a tabular form (perhaps a different form for different classes of knowledge) by simple mechanical means, and (2) the specific knowledge (which is usually most of the semantic net) can be kept in relatively inexpensive secondary storage and even accessed through an efficient, existing data management system. The general knowledge (everything else) is kept in primary memory. Fortunately, most processing by the inference rules occurs on other than the "bottom" of the network, so that efficiency is maintained.

Semantic networks are by far the best available technology for representing definitional and relational knowledge that is too complex for ordinary data management techniques. This is the case because (1) the structure allows for the inclusion of ad hoc and pathological information, and (2) the utilization of inference rules permits straightforward enhancement of the inherent representational power and completeness.

On the other hand, there are some disadvantages to the use of semantic networks to represent knowledge in a KBS. The principal one is that the chunk size is fairly small. This leads to two problems: (1) instances of relations do not lend themselves to use in explanations of chains of reasoning developed by the inference rules--chains can be quite lengthy and tedious, i.e., below the threshold of interest; and (2) processing a semantic net can consume large amounts of computer time. Another disadvantage is that many kinds of knowledge cannot be expressed (as instances of relations) in a natural manner. Examples are most procedural knowledge, relative and subordinate knowledge, and quantified knowledge. (See [WOODS75] for a thorough discussion of these and other issues associated with the use of semantic networks.)

Some papers and articles about semantic networks and their utilization are: [DUDA77], [GRIGNETTI75], [MYLOPAULOS75a and 75b], [NORMAN75], [QUILLIAN68], [SCHANK75a], [SIMMONS73], [TRIGOBOFF76], [WOODS71], and [YAKIMOVSKY76].

4.1.2.6 Frames

A research topic of great current interest in computer representation of knowledge is frame theory. No one has succeeded in defining frames to all researchers' satisfaction, but there is a commonality in motivation and in some of the proposals to date. The common motivating issues are (1) accommodation of both declarative and procedural knowledge in the same representational formalism, (2) accommodation of mundane, ad hoc, and idiosyncratic knowledge along with that which is more uniform and repetitive in nature, (3) accommodation of partial and somewhat contradictory or inconsistent knowledge, and (4) ability to plausibly reason from a KS with features like the above. Two major issues not yet dealt with in the emerging theory are explanation of system behavior and naturalness of the knowledge-acquisition interface. (These issues are related, and both stem from an unwieldy external format of a frame. See the example below.)

Some of the common features in proposals about frames are: (1) A frame is a knowledge chunk that (2) has a collection of definitional and procedural knowledge about an object, action, or concept. (3) A frame is a complex data structure that (4) has named slots corresponding to definitional characteristics and (5) the ability to attach procedural knowledge to the slots and/or the frame itself. Further, there can be associated with the slots: (6) restrictions on the contents (like data-typing information), (7) default values that can be static or computed in terms of the values in other slots, and (8) monitors--procedures that test for and deal with unusual conditions. There can be associated with the frame itself: (9) expectations--assumptions or predictions based upon the existence of an entity described by the frame, (10) methods of logging and correcting complaints that arise when expectations are not met, and (11) specialized procedural knowledge for manipulation of the entity.

It is not possible to give a simple example that demonstrates all the above features of a frame. Therefore, the example in Figure 4.19 makes no such claim--it is offered to show some of the intended flavor of a frame system. The top of the figure gives definitional information about a dog. The first line states that a dog is a "mammal". The next line means that there is a slot, named "kind" (of dog), that may be filled with a value of (type) "breed". ("Breed" in this example is itself a frame.) The color of a dog is limited to one or a mixture of the stated colors by the SUBSET.OF operator. Default values are indicated by underlining, and the FROM operator is used to pick out values from other frames. Thus, the combined effect of the phrase <u>FROM color OF kind</u> is to make the default value for the color of a dog the default for his breed. Going on, the dog frame has a slot for the number of legs that is restricted to be no more than four with a default of four, and a weight that is a positive number with a default weight that is determined by the typical size of members of the same breed. The state of a dog is either "adult", the default, or "puppy" if the age is known to be less than one year. The age of a dog is restricted to a positive number and its default value can be <u>calculated procedurally</u> by the trivial expression, "now-birthday". The "dog" frame is finished by declaring birthday to be a date and name to be a "string".

The bottom of the figure shows a frame for "boxer" and declares that boxer is a breed--but only a breed of dog. The color of a boxer is restricted to one of the colors "tan", "brown", and "brindle", with a default of "tan". (Note, it is legal for this to conflict with the dog frame; i.e., brindle is not mentioned

```
  dog   FRAME ISA mammal

        kind   breed

        color  SUBSET.OF {tan brown black white rust}

               FROM color OF kind

        leggedness   0...4

        weight       >0, FROM size OF kind

        state        adult OR puppy if age <1

        age          >0, now-birthday

        birthday     date

        name         string

        END          dog


boxer   FRAME  ISA breed OF dog

        color          ONE.OF {tan brown brindle}

        size           40...60

        tail           bobbed OR long

        ears           bobbed OR floppy

        temperment     playful

        COMPLAINTS     IF weight >100 THEN ASSUME(great dane)

        END            boxer
```

Figure 4.19. Example Frame Definitions

there.) If this breed did not have a characteristic color restriction, then this slot would be omitted; this would have the effect of not giving a default assignment for color in the above dog frame.) The next slot says that the size of a boxer is between 40 and 60 pounds. No default is specified. Thus, the default value for weight of a boxer in the "dog" frame is just this range (rather than an exact value) when applied to a boxer. The tail and ears slots are defined with default value "bobbed" and the respective alternatives of "long" and "floppy". Temperament is shown to always be playful. The last line shows an example of a complaint and ad hoc knowledge used to make a recommendation, namely, if you see a giant boxer, then assume that it might be a Great Dane instead.

Figure 4.20 shows an example use of frames in a recognition task. The top of the figure shows some feature values (e.g., color is tan, ears are bobbed) that have been detected for an object, here identified as number 456. The CE has matched the known feature values with the available frames and has manufactured the working hypothesis shown at the bottom of the figure--namely, a boxer dog that is assumed to have a bobbed tail and to be an adult. It is noted that this particular boxer (object number 456) is mean and that this is exceptional. Also, the size of the boxer was only approximately known, but the approximation has been used in lieu of a more accurate value.

It is hard to see from this example how the CE goes about this kind of recognition task. However, a possible scenario would follow these lines. A general matching procedure would attempt to instantiate all frames in the system until a reasonable fit was found; in the example, "boxer" is a reasonable match. Slots that are yet unfilled would be used to hypothesize other values not yet detected. For the boxer frame, a bobbed tail would be predicted and put on an agenda of things to look for. Assuming there was a frame for tails, it might possibly contain heuristic knowledge about how to more carefully scan the raw data to confirm or deny the existence of a particular kind of tail. Other

```
LOW-LEVEL INFORMATION

  OBJECT 456

       color = tan

       ears = bobbed

       leggedness = 4

       size = 40-45

       temperment = mean


TRIAL IDENTIFICATION

  [OBJECT 456 ISA dog

   kind        boxer WITH [color tan

                           size  40-45

                           tail  ASSUMED bobbed

                           ears  bobbed

                           temperment EXCEPTIONAL mean]

   color       tan

   leggedness  4

   weight      40-45

   state       ASSUMED adult]
```

Figure 4.20. Inexact Match by a Frame System

activity that could emanate from the boxer frame is the activation of a complaint. Thus, if the weight of the boxer was too large, the complaint mechanism could (tentatively) change the identification of the instantiation of the boxer frame into one for a Great Dane. There are two advantages to this: (1) rather than returning to a very general pattern-matching activity, a candidate that is highly likely to be right is selected next, and (2) the slot values for this frame can be transferred to the new frame with little additional work.

Besides the prediction and correction activity resulting from the partial match to a frame, a third process can be tried. Namely, if the match is good enough, then the frame can become more informative. For our example, the transformation is from a boxer to a boxer dog where more information is absorbed, e.g., leggedness.

The above steps (prediction, correction, and inclusion of more information) continue until all of the low-level information is consumed and the correction activity reaches quiescence. The belief is that this style of recognition will be more goal directed--and hence more accurate and efficient--than general techniques that depend upon regularity and uniformity of structure.

Some interesting work using frame representations are [DBOBROW77a, 77b, and 75c], [DAVIS76], [DUNLAVEY75], (GOLDSTEIN76], [KUIPERS75], [MALHOTRA75b], [MINSKY75], [RUBIN75b], and [WINOGRAD75].

4.1.3 Comparison of Knowledge Representation Techniques and Issues

This section attempts, in an informal way, to compare the six techniques described in Section 4.1.2 for representing knowledge in a KS. It must be stated that those six are not the only ones available. They were selected for analysis because they are the most widely used and best documented, which suggests that they are the techniques that have been most successful for a variety of applications. On the other hand, there are many successful systems and research activities that have used one-of-a-kind techniques and methods or are attempting to discover new, general methods to enhance the above six. An example of such a unique endeavor can be found in [KAHN 75]. This work is cited in particular because it addresses an important issue for problem-solving systems--namely, techniques for representing knowledge of time dependencies and temporal history. Other examples of specialized representation techniques are available in the literature on speech understanding systems and game-playing programs.

4.1.3.1 Comparison of Techniques

Figure 4.21 compares the six described techniques by a variety of criteria. A three-valued scale is used: Good, Mediocre, and Bad.[*] For each criterion, the full scale range has been used even though it may be the case that no technique deserves one of the extremal ratings. For example, none of the techniques has a form that is really "natural to the expert." That is, none are technical English with mathematical expressions. However, given only a three-valued scale, it is desirable to use all of it in order to provide reasonable discriminations. On the other hand, the relative ratings are merel the opinion of the authors, and any more than a three-valued scale would imply unrealistic precision. (The meanings of most of the comparison criteria can be found in Section 4.1.2.4.)

[*] Perhaps, to assuage the feelings of the proponents of each technique, the scale values should be excellent, great, and good.

	FSM	PGM	P.CALC	PROD	S.NET	FRAME
Represent declarative KS	M	B	G	G	G	G
Represent procedural KS	G	G	B	M	B	G
Represent credibility factors	M	M	B	G	M	G
Represent meta-knowledge	B	G	M	G	M	M
Represent ad hoc knowledge	M	G	B	M	M	G
Chunk size	big	big	sml	med	sml	big
Aid to explanation	B	B	M	G	M	B
Natural to expert	M	B	M	G	M	B
Modularity	M	B	G	G	G	M
Easy to extend system	M	B	G	G	G	M
Easy to modify behavior	M	M	B	M	B	G
Reusable components	M	B	G	M	M	B
Run-time efficiency	M	G	B	M	B	M
Tolerate inconsistency	M	M	B	G	B	G
Available theory	G	M	G	M	G	B

Figure 4.21. Comparison of Knowledge Representation Techniques

In some sense, it is difficult to compare the techniques because, for most applications, one of them will clearly be more natural than the others. However, two general comparisons are worthy of some discussion: Predicate calculus versus production systems, and semantic networks versus frames.

Predicate-calculus and production-system representations invite comparison because of a striking similarity in their surface appearances. (Consider well-formed formulae written as implications and production rules written in an IF-THEN formalism.) In fact, the predicate-calculus representation can be modeled in a reasonably straightforward way as a production system. However, the converse is not true, because: (1) production rules may use domain-specific procedures, (2) production rules have access to a data base in which to maintain state in a domain-specific manner, (3) production rules may include confidence factors, (4) production systems can tolerate inconsistent knowledge, and (5) production rules can describe state-changing operations better. (See the example of the monkey and bananas problem in Section 4.1.2.3. The axiom defining each operator must produce a total state description so that at each stage, consistency can be maintained.)

Obviously, production systems offer a more natural and more efficient technology for many applications. The value of the predicate calculus is that the theory of use is highly developed, and, if the system can rely upon the consistency of the knowledge, then algorithms are available to solve any problem that has a solution. Another comparison between predicate calculus and production systems is chunk size. Because of the five above points, production rules can express more knowledge (much of it implicit) for a given (lexical) size. Also, it appears that the chunk size is more appropriate to the experts using the system and providing knowledge at the acquisition interface.

The comparison between semantic networks and frames is at best tenuous, since there is no agreed-upon definition for frames. However, an attempt at comparison is worthwhile, because some of the differences point out directions in which research is going in the area of knowledge representation. The first thing to note is that frames, as we have shown them in Section 4.1.2.6, appear to "cover" semantic networks. This is evident when one makes a correspondence between entity names and frame names and slot values, and between relation names and slot types. With these correspondences in mind, the rest of the comparison is easier to make.

First, compare the chunk size. In a semantic net, a single relation is a chunk. For frames, a chunk is the total set of relations in which the entity is envolved. The second comparison is simply that the frame may have attached procedures to assist the general reasoning mechanism in various ad hoc ways, and the semantic networks do not. On the other side, the semantic network is more amenable to use with "universal" reasoning principles, e.g., the rules of inference. Also, uniformity in structure allows the more specific knowledge in semantic networks to be handled in less costly ways (e.g., as an external data base); frame systems are not as likely to profit from uniformity. In summary then, it appears that frame systems are a better and more efficient representation technique when the domain has a large amount of ad hoc or partial knowledge and uses many idiosyncratic principles of reasoning. Semantic networks appear better when the domain has a high degree of regularity and uniformity. It is our guess that the former characteristics better describe the domains in which non-toy problem-solving KBS ultimately will perform, and, hence, that frames will ultimately replace semantic networks and the other representation techniques for most applications. But we do not expect this to take place before another five or ten years.

4.1.3.2 Knowledge Representation Issues

This section briefly enumerates some of the important problems and hence research topics in knowledge representation.

Epistomology--A general definition of knowledge and understanding, though not completely necessary, would provide a common ground for competing theories.

An important issue in this area is the development of a taxonomy of knowledge that would permit various representation and reasoning techniques to be more adequately compared.

Higher-level knowledge--Included in this are control knowledge (what to do next), meta-knowledge (knowledge about knowledge), and self-knowledge (understanding limitations and capabilities). The issues are how to represent higher-level knowledge, where to put it (i.e., in the CE or the KB), and how to use it effectively.

Confidence factors--The problems are where to get them, where to use them, how to combine them, and how to interpret them--hence, how they should effect the reasoning process.

Explanations--How is a system to incorporate supporting knowledge chunks into explanations and into defenses of its proposed solutions? The problem arises because the existing representation techniques are not flexible in format or chunk size. Hence, the user (and the knowledge-providing expert) must sometimes conform to the system, rather than the (desirable) reverse. The problem is often made worse when the input knowledge is compiled into a physical form from which the original chunks cannot be recovered. Also, it is unknown how higher-level knowledge and confidence factors should be included in explanations.

Extendability--There are several issues here: Who--the domain expert or the system--is responsible for maintaining consistency of the KB? If it is the system, how can such checks be automated? How can partial knowledge be handled? What knowledge compilation techniques are available that do not preclude incremental modification and additions? How can the knowledge-acquisition bottleneck be widened by allowing more natural modes of expression? This and the last issue, explanations, raise the question of how the system interface and knowledge-representation methodologies should be related.

<u>KS cooperation</u>--In many KB systems, the KB consists of many fact files and knowledge sources. The issue is how these fact files and knowledge sources should cooperate, particularly if they are represented by different techniques. Related topics are: How do different knowledge sources refer to the same entity? How are discrepancies resolved when different knowledge sources provide conflicting advice or conclusions? How does the acquisition mechanism know where to put new chunks? How does the CE know which knowledge source to use for which problem? How are "infinite loops" avoided when two or more knowledge sources start passing the buck?

This list of issues about knowledge representation is in no way complete; it omits open problems concerning a particular representation methodology (which are, however, briefly covered in the sections about the individual methods), and we hope it gives some flavor of the kinds of research areas that are currently being pursued.

4.2 WORKSPACE REPRESENTATION

The topic discussed in this section is methods for representing the workspace. The workspace is the encapsulation of the system's current state in a problem-solving activity. It includes: (1) global variables--computed values, goals, and input problem parameters; (2) an agenda--a list of activities that can be done next; and (3) a history--what has been done (and why) to bring the system to its current state.

The simplest example of a workspace representation is the push-down stack in a LISP-like system. The stack contains the bindings of global variables, return addresses (a history snapshot), and the values of temporaries. There is no agenda in a simple system other than the program counter. A more complicated example is the data base in a production system. It contains the entire state of the system, including an implicit agenda (the conflict set of rules that can apply).

In most computer programs, the workspace can be represented in an ad hoc way using whatever techniques are provided by the containing program-language system. However, this is not always adequate in KB systems because (1) the capability to provide explanations is based in part on an ability to find the trace of events (history) that produced the solution, and (2) a major part of efficient plausible reasoning behavior is the procedure for selecting the next thing to do--hence, the necessity for an explicit agenda and visibility of enough state (global variables) to make informed decisions. Further, if a KBS has many knowledge sources, the workspace representation may be used for "impedance matching" and to provide a communication channel among them.

The remainder of this section will briefly describe a few techniques used to represent workspaces. No attempt is made to compare these techniques, because there is rarely a choice; given a knowledge representation technique and a procedure for reasoning in a domain, the choice of workspace representations is strictly limited.

4.2.1 The CMU Blackboard

The Carnegie-Mellon University (CMU) speech-understanding system, Hearsay II, employed a novel and interesting workspace representation called a blackboard--see [ERMAN 75]. The same technique has been adopted for use in a KBS that determines the structure of a protein from X-ray crystallographic data--see [ENGLEMORE77].

The blackboard is a data structure that serves as an intermediary among multiple knowledge sources and the system's CE. The blackboard is two dimensional; one dimension is levels of representation, and the other dimension is time. In Hearsay II, the levels of representation are conceptual, phrasal, lexical (words), syllablic, surface-phonemic, phonetic, segmental, and acoustic-parametric. The concept of level is that an entity at one level (e.g., a word) is made up of a sequence (in time) of entities at a lower level (e.g., syllables). (In the protein crystallography system, the levels form a partially ordered hierarchy instead of a well-ordered set.)

In these systems, a KS examines entities at one level and hypothesizes or confirms (the existence of) entities at another (usually adjacent) level. Figure 4.22 shows the levels and knowledge sources in the Hearsay II system. Arrows, labeled with KS names, show input and output levels, and some are bi-directional. An entity includes several pieces of information: a level name, time restrictions or boundaries, a name, confidence level, and support. Support for an entity is the collection of other entities that cause this entity to exist.

Figure 4.23 shows a fragment of a blackboard. As depicted, the support is ambiguous. For example, the word BAD at the lexical level could be supported by the existence of the phonemes B, AH, and D at the phonetic level. However, the opposite could be true, namely, the word BAD could have been predicted from higher-level considerations and then caused the phoneme predictions. Therefore, part of the support representation must include directionality information. This simple example does not expose another important issue--competition. For instance, assume the phoneme D was ambiguously recognized

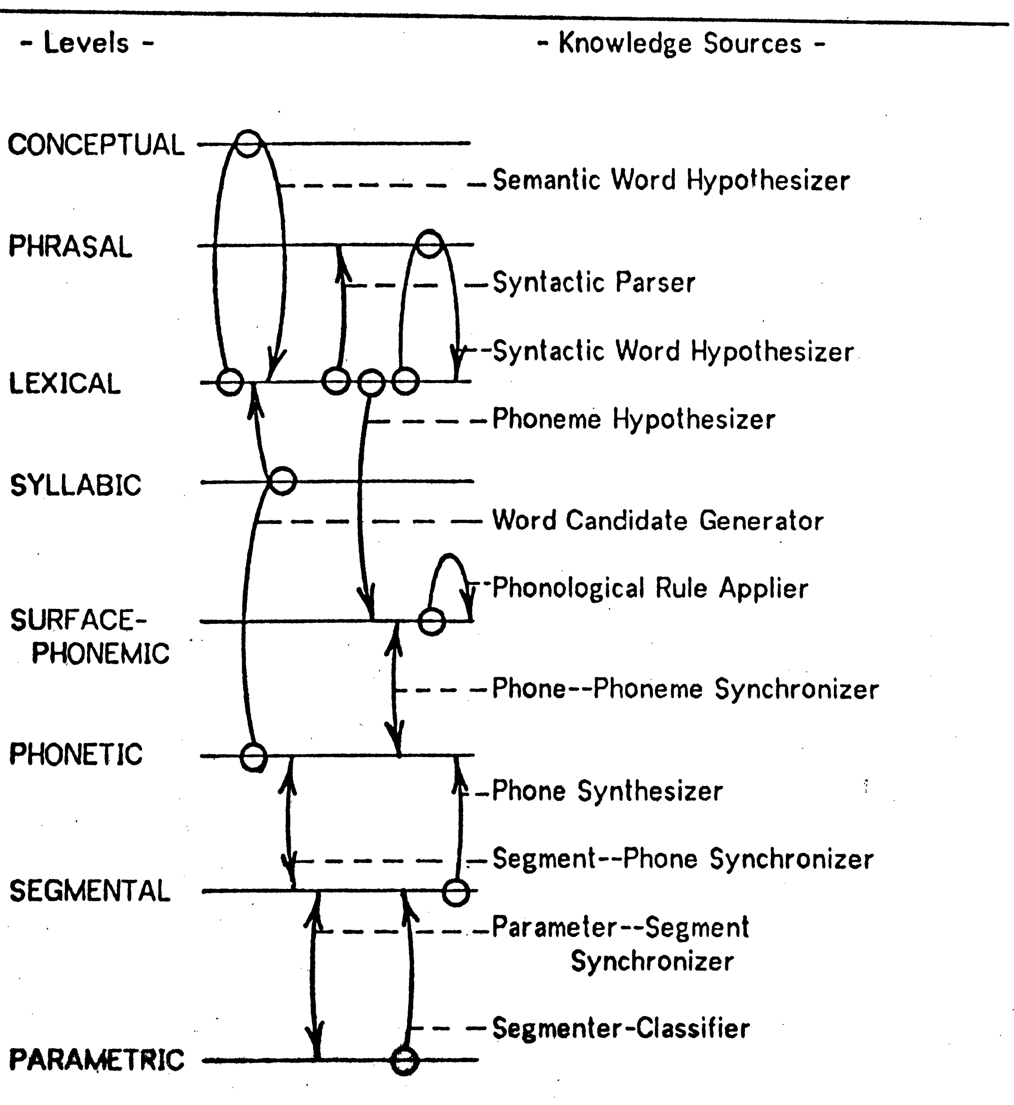


Figure 4.22. Hearsay II Levels of Representation and Knowledge Sources (from [ERMAN 75])

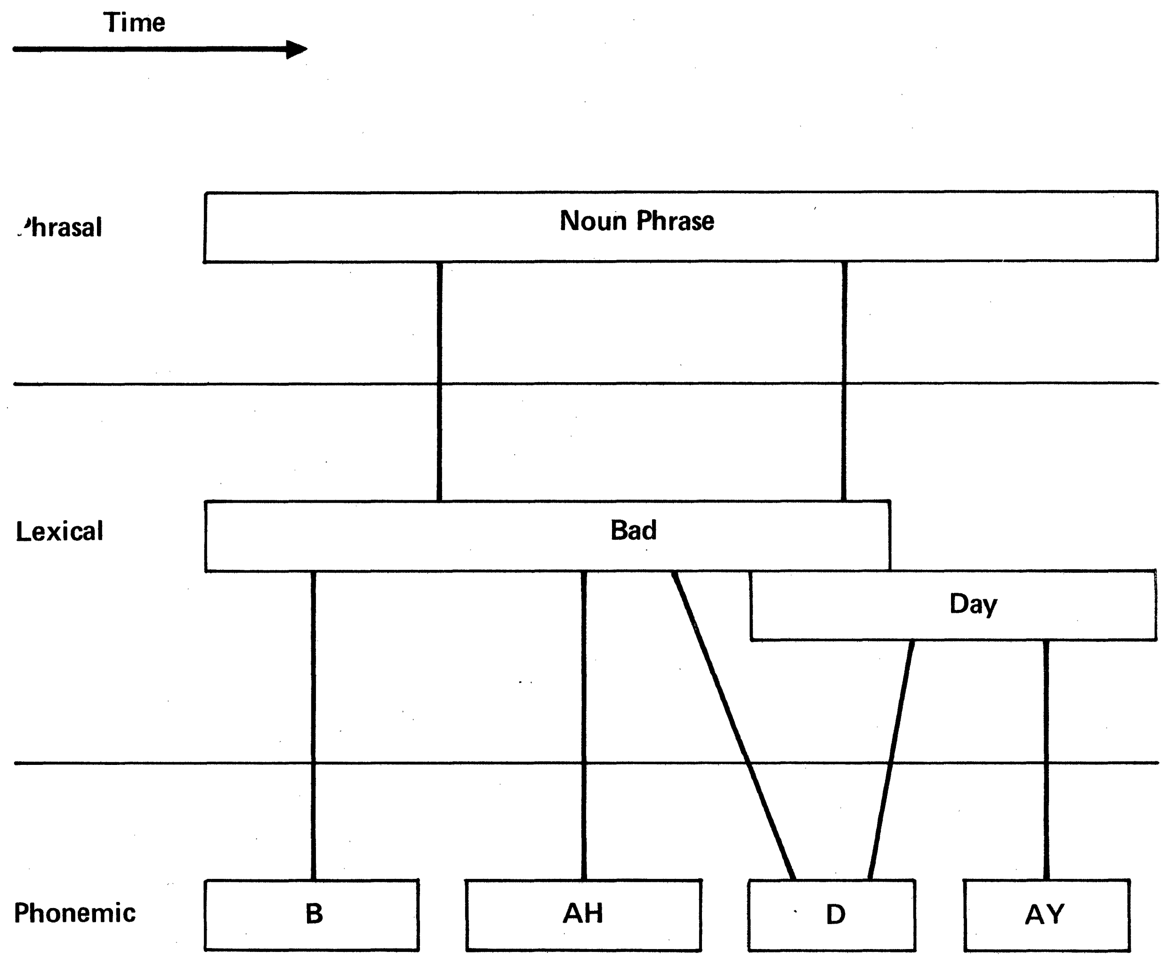


Figure 4.23. Blackboard Example

as either a D or a T, then the word BATTY (assuming it was in the system's lexicon) could be in competition with the shown sequence BAD DAY. In fact, the blackbcard is an ideal structure for representing this type of competition. All that is done is to use a "third dimension" to allow competing entities to pile up.

To summarize, a blackboard fills the three roles of a workspace representation by (1) global variables--the blackboard is the globally visible data structure in the system; (2) an agenda--when an entity is placed in the blackboard, it is to be presented to the knowledge sources that have the entity's level as their input level, and the set of all such presentations that have not yet been performed are the agenda; and (3) a history--the support represented explicitly in the blackboard is a trace of the evolution of the system's state.

4.2.2 Move Graphs

Move graphs are one of the better-known methods of representing a workspace. A good introduction to the topic can be found in [NILSSON 71]. Move graphs are usually used in problems requiring search. Problems of this kind have, as given, a start state and a goal state. Both states and all generated intermediate states are represented in a single formalism chosen for the system. A KS provides operators--methods of transforming one state description into another. The transformations allowed by the operators correspond to the legal moves in a game. The problem is then to find a sequence of operators (moves) that transform the initial state into the goal state. (For an example of this kind of problem formulation, see the description of the monkey and bananas problem in Section 4.1.2.3.)

The nodes in a move graph are the representations of a state; the edges connecting the nodes are directed and labeled with the operator that produced the transformation. Figure 4.24 shows a move graph for the game of tic-tac-toe, three in a row. The initial state is the empty board and the desired final state (not yet reached) is three X's in a line. The edges are labeled with the name of the operator, X or O, who has made the move. (Only positions not equivalent by symmetry are generated.) The graph is made by first placing the start node on an agenda. The system then loops using the following technique: Select and remove a node from the agenda and use the KS to generate all legal successors to the selected node. Link the new nodes to the original and place them on the agenda. If none of the new nodes is a goal node, then repeat the loop. An obvious place for such a system to be smart is when it picks a node from the agenda and generates all of its successors, a process called expanding a node.

If the system expands nodes that are on short paths to the goal node, the performance will be good. In some systems, a KS is used to provide knowledge as to what node should be expanded next. An issue begged by the simple

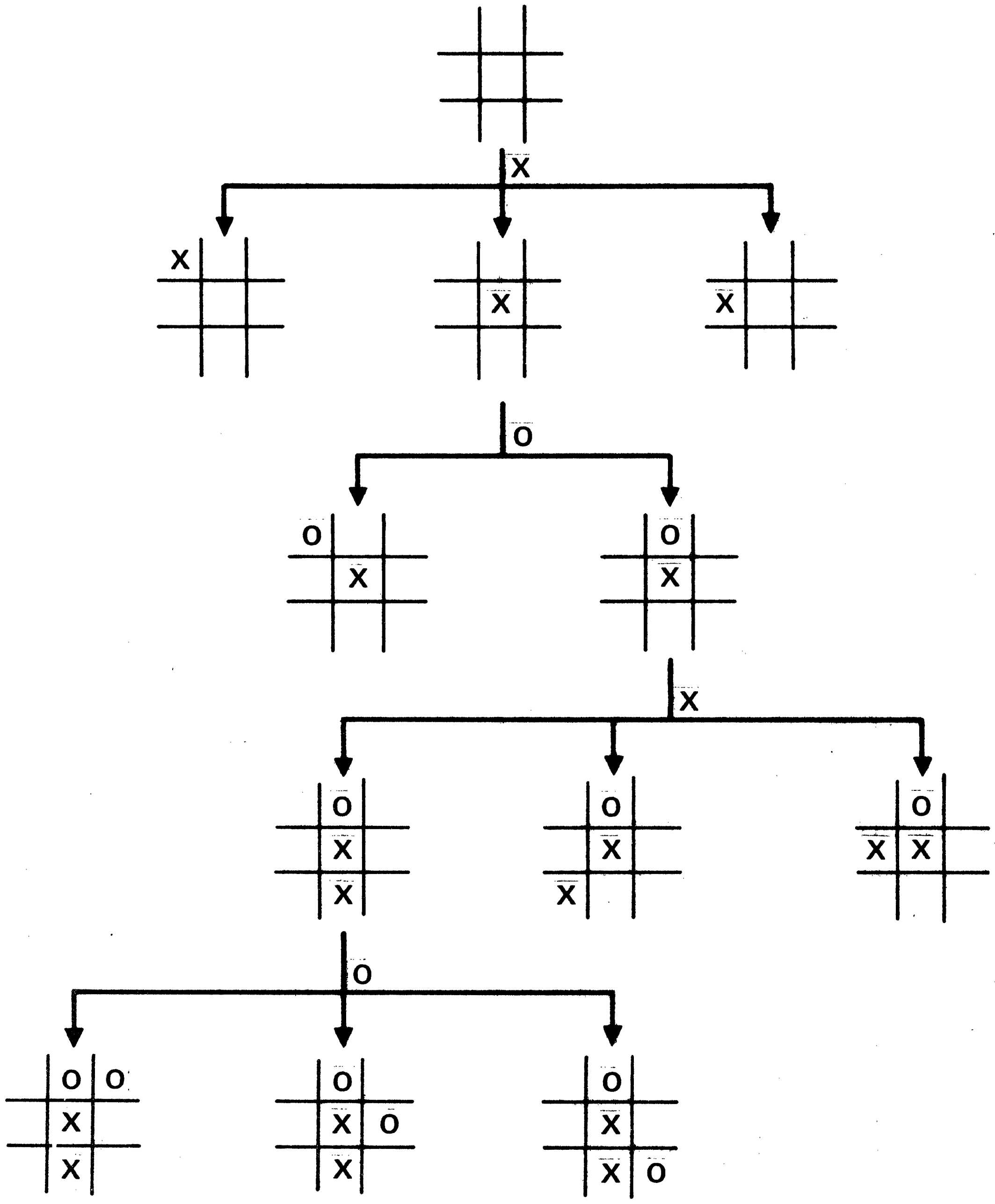


Figure 4.24. Game Tree for Tic-Tac-Toe

algorithm above is knowing when to quit, i.e., knowing when no solution is forthcoming.

Figure 4.25 shows a different kind of move graph. The example is a formation of a plan to get a new table (the goal node). Each node in the graph is a subgoal of its parent node. (Since this is a graph, a node may have more than one parent.) The goal "get a table" can be satisfied by satisfying either the subgoal "make a table" or the subgoal "buy a table" and is thus called an OR node. The subgoal "buy a table" is satisfied by satisfying both the subgoal "get money" _and_ "select a store" (at which to buy the table), and is thus called an AND node. AND nodes in the figure are shown by connecting emanating edges with an ampersand (&). (The operator names that caused each goal to be expanded into subgoals have been omitted in the figure but would be present in an actual implementation.) A workspace representation such as this is called an AND/OR graph and is used in many systems with a predicate-calculus or production-rule knowledge sources.

It is interesting to note that the move graph shown in Figure 4.24 is built with the start state as its root and is expanded until a goal node is produced (or the system gives up). The AND/OR graph in Figure 4.25, on the other hand, is built with the goal state as its root. This raises the question of how to terminate node expansion for a goal-rooted graph. The general procedure is to continue expansion until a satisfying (to the and/or relations, in this case) set of nodes have been generated, all of which are primitive. A primitive node is one that poses a problem that is known to be solvable without search by the system.

A quick comparison of MOVE graphs with the CMU blackboard described in the last section uncovers the facts that (1) MOVE graphs have a more uniform structure that can sometimes be exploited for efficiency, and (2) the CMU blackboard has a better structure if the problem decomposes into levels of representation and the system has many knowledge sources. Like the CMU

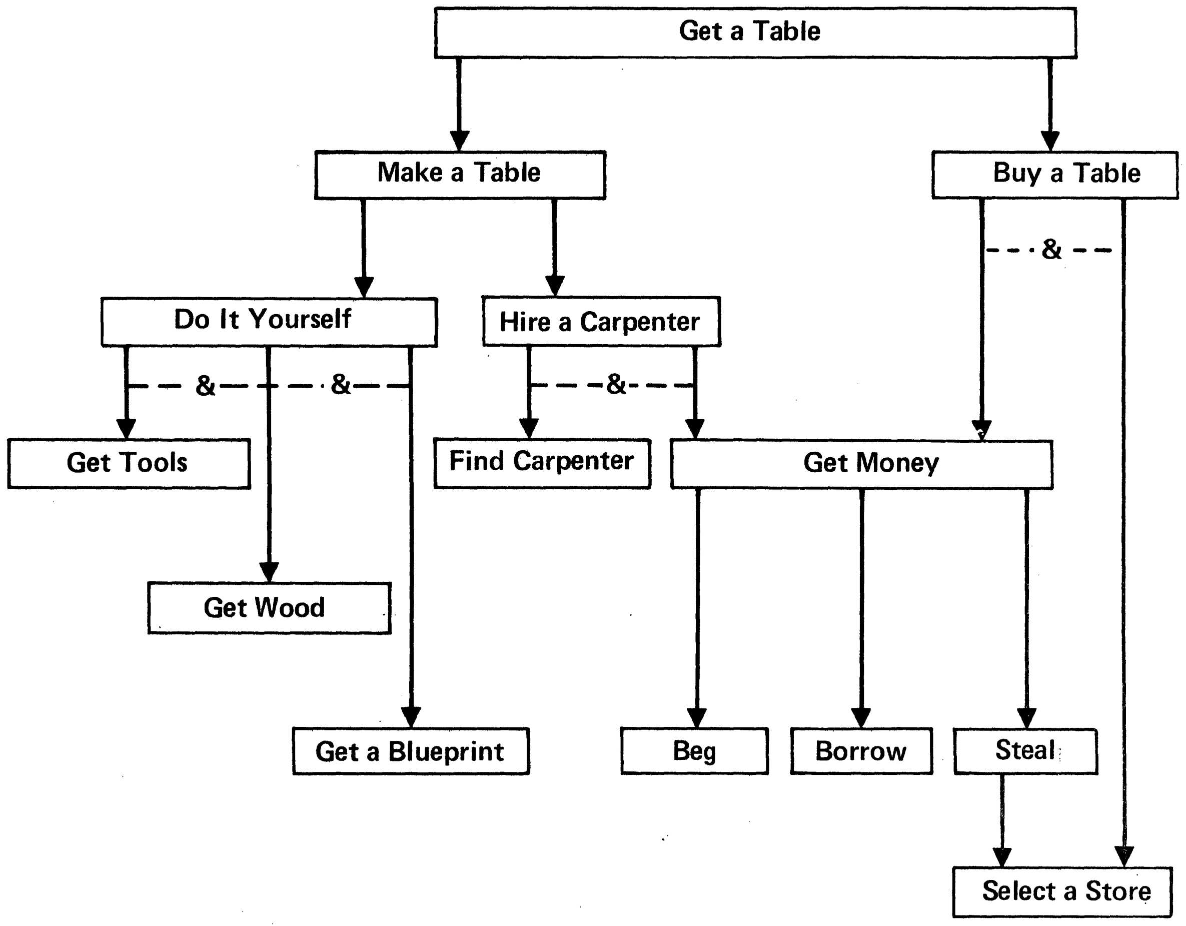


Figure 4.25. Example AND/OR Graph.

blackboard, move graphs fill our requirements for a workspace representation: (1) global variables--the graph represents the global data structure and includes goals and partial results, (2) an agenda--the agenda is the set of unexpanded nodes, and (3) a history--the labeled links in the graph give a reason for the existence of each entity.

4.2.3 KS Format and Special Methods

In some systems, the workspace can be represented by the same (or a similar) formalism used to represent a KS. This has the obvious advantage of allowing the system to assimilate derived knowledge into the KS in a natural manner. In general, well-formed predicate calculus formulae and production rules do noi lend themselves to this because the structures are too linear; i.e., there is no explicit connectivity with other structures, so connectivity must be provided by some method external to the formalism. This is necessary to form an agenda, provide historical information, and collect the dynamic state into a viewable whole.

Programs and FSMs also do not make an appropriate workspace representation, even though there is some connectivity. The problem here is that much of the state, in a system using these KS representation techniques, is buried in the interpreter and must be explicated in order for the system to make global decisions and maintain a history of processing. In some of the AI programming language systems, a global data base is used to contain deduced information, current goals, and invocation patterns for the procedures. See, for example [HEWITT 72]. Semantic networks and frames provide a better representational formalism than the above four, for representing a workspace. This is in fact one of the key advantages of frames for a recognition task--namely, partial information about an entity is kept in a form that is virtually identical to the definitional knowledge in the system. Links and connectivity are thus automatically provided. See the example in Section 4.1.2.1. Also, see [BOBROW 77a]. For a discussion of the possibilities and problems with using semantic networks as a workspace representation and assimilating knowledge, see [WOODS 75].

A few interesting approaches and formalisms for representing workspaces are described in: [BARNETT 75a], [BERNSTEIN 76], [J. MOORE 73], and [WOODS 76].

4.3 THE COGNITIVE ENGINE

In a KBS, the primary function of the cognitive engine (CE) is to perform the task of problem solving. A secondary function of the CE is to explain the behavior of the system and support its derived solutions. To accomplish these functions, the CE must (1) control and coordinate system activities and resources; (2) plausibly reason about domain-specific problems by having access to and using the contents of the KB, the contents of the workspace, and knowledge and procedures embedded in the CE; and (3) link the KB with the interface module(s). This section discusses only (1) and (2). Section 4.4 describes technologies used by the interface modules.

The CE is the most active component of a KBS. That is, it is always instantiated, and it controls the activation and utilization of other system modules. (For an elaboration of the concept of an active component, see Section 4.1.1.4.) Another characterization of a CE is that it is the intelligence or understanding component of a KBS. This follows from "S understands knowledge K if S uses K whenever appropriate."--a position espoused and defended in [J. MOORE 73]. This characterization of CE function is appropriate even though its activity may, to a degree, be guided by higher-level (control and/or meta) knowledge in a KS because, at some point, the CE must still resolve any residual conflicts at whatever level they occur. In an "ultimate" KBS, where all knowledge resided in the KB and the CE was just a rudimentary interpreter, this would not be the case; the understanding capability would be distributed throughout the system. However, no such ultimate system exists today, nor is it likely to exist in the foreseeable future.

The remainder of this section introduces some of the terminology used to describe CEs and details a few general techniques used in their construction. (Some CE technology used only with a particular KS representation is discussed in Sections 4.1 and 4.2.)

4.3.1 Terminology, Measurements, and Characterizations of the CE

4.3.1.1 Effectiveness Terminology

A CE is called sound if it produces only correct or "I don't know" solutions, i.e., it does not produce incorrect solutions. Soundness is of dubious value for a KBS operating in a domain that includes many problems that can be (approximately) solved only by inexact methods. It is often better for the system to make a good guess or derive a small set of the most probable solutions. As used here, soundness is a property of the CE, not of the KB. That is to say, that the CE rating is independent of the veracity of the knowledge chunks used to generate a problem solution.

A CE is called complete if it can always produce a solution to a posed problem when a solution exists. A CE can be, in principle, complete even though it contains some arbitrary limit on resource expenditure. This is necessary when the domain may contain undecidable problems (see Section 4.1.2.3 for a short discussion of the undecidability of the first-order predicate calculus). As with soundness, we are defining completeness to be a property of the CE, i.e., in order to be complete, the CE is not required to solve problems for which necessary knowledge has been omitted from a KS. It is also possible to talk about completeness of the KB and completeness of the entire KBS in a domain. However, it is unlikely that there are interesting domains for KBSs where these latter kinds of completeness can be achieved.

A CE is admissible if it always finds a minimal-cost solution when a solution exists. The cost is taken to mean the cost of using the solution, not necessarily the cost of finding it. A typical criterion in a state-space search problem is to find the shortest sequence of operators that transform the initial state into a goal state. A CE that can always find a solution sequence of operators (when one exists) is complete--one that always finds a shortest sequence (when one exists) is admissible. Hence, admissibility implies completeness; either implies soundness. A problem occurs at this point because, while soundness may not be a desirable property of a KBS, some weaker form of admissibility is. That is, even though the CE cannot guarantee that all solutions are correct (perhaps only plausible), it is still proper and necessary for a system acting as an expert to derive optimal or near-optimal manifestations of the proposed solutions. We make no attempt herein to improve the included definitions in light of the demands of KBS technology, but merely note the issue. (These definitions originate from aspects of the formal theory of predicate calculus and theorem proving.)

4.3.1.2 Efficiency Terminology

Besides the above definitions, which relate to problem-solving potential, there is another class of characterizations that deal with efficiency. Efficiency is not as easy to measure for KBS as it is for other kinds of computer systems, because run time and dynamic memory consumption of a KBS are often highly non-linear functions of some problem parameter. In many cases, the function is not known, making theoretical comparison difficult. In other cases, the parameter is something like the number of operators necessary to transform the initial state into a goal state--a parameter that is not available until after the system has solved the problem.

Two efficiency measures, defined for systems using move graphs as their work-space representation or any kind of state-space search technique, are <u>penetrance</u> and <u>branching factor</u>. The penetrance, P, is defined as

$$P = \frac{L}{T}$$

where L is the length of the derived path from the initial state (or node) to the goal, and T is the total number of states (or nodes) generated while searching for a solution. If the CE proceeds directly to a solution without generating any false paths or unused states, the penetrance achieved its maximum value, 1. The smaller the value of P, the less directly the system proceeds to problem solution. Since performance is usually non-linear with L, the value of P generally decreases with increasing L. Therefore, P is often considered as a function of L, and the values of P(L) are estimated to characterize performance.

Computation of branching factor is made by assuming the existence of a tree with the same total number of nodes, T, as states produced by the system in solving a problem. The tree is further assumed to be one in which (1) every expanded node has B descendants and (2) the tree has paths of length, L, the number of operators in the solution path of the original problem. Therefore,

$$T = \sum_{i=0}^{L} B^i$$

This can be rewritten as

$$T = \frac{B^{L+1} - 1}{B - 1}$$

and solved for B, the branching factor, by iteration. By definition, B can never be less than one. Further, small values of B indicate that the system has made direct progress toward the solution of the problem, while large values of B indicate that the system has wasted time expanding nodes not used in the final solution or has included states that have not been further expanded. In general, the branching factor of admissible systems will be larger than others because (1) it clearly takes more work to find an optimal solution, and (2) the value of L in the above formula will be smaller.

Figure 4.26 shows a sketch of a move graph (on the left) with T=15 nodes and a solution path (shown by the darkened line) of length L=3. Therefore, the penetrance P = L/T = 1/5. To the right is shown a balanced tree with T=15 and L=3. As can be observed, B=2. (And this is the solution to the above definition,

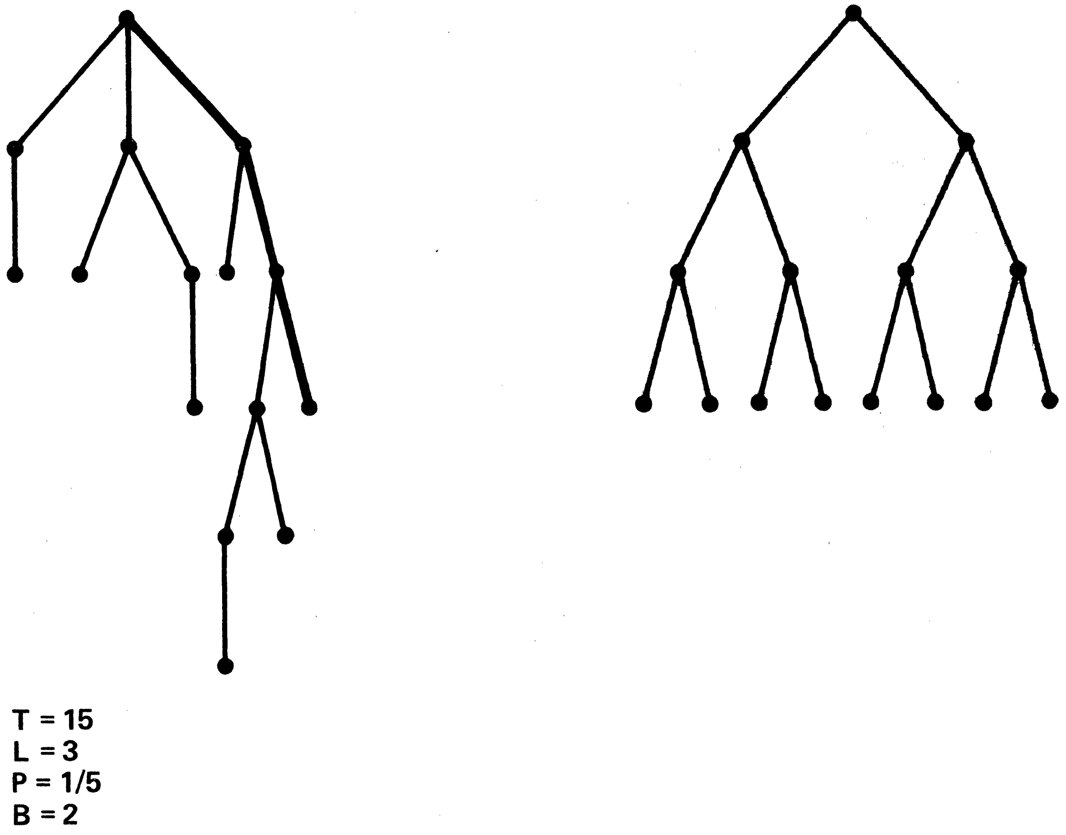


Figure 4.26. Example Move Graph and Balanced Tree

i.e., $15 = (2^4-1)/(2-1)$.) A measure closely related to these is computation time as a function of input length--e.g., the number of words in a sentence input to a natural language understanding system.

Usually, the branching factor for a system varies less as a function of L than does the penetrance because, except for small or trivial problems, the procedures used tend to be more exponential in L than linear. Because of this fact, research into CE performance does not look for ways to gain linear speedups by, say, recoding a LISP program into assembly language. Rather, methods are sought that tamp down the relative branching factor or exponential by some technique. For example, the alpha-beta search algorithm, discussed below can, in cooperation with a good KS, search for a solution in a time proportional to approximately $B^{L/2}$, where a straightforward technique will take a time proportional to B^L.

The terminology introduced in this and the last subsection is discussed in more detail in [NILSSON71].

4.3.1.3 Control Terminology

Another class of terminology used to describe a CE details features of its control and error-handling mechanisms. These two mechanisms are inseparable in most CEs because the knowledge in the KB is often soft (and in fact confidence rated), and the rules of interference are often only plausible. The simplest approaches are backup and simulated non-determinism; see Sections 4.1.2.1 and 4.1.2.2 for more information.

The input to a CE is usually a set of initial conditions and a goal. The KB is used in some manner to find a method of obtaining the goal given the constraints imposed by the initial conditions. There are four ways of doing this:
(1) _Forward chaining_--apply the KB to the givens to infer new conditions; continue in this manner until the goal is satisfied. (2) _Back chaining_--apply the KB to the goal to produce new _subgoals_; continue in this manner until the

initial constraints or primitive conditions (known to be solvable) are reached. (3) Chain both ways--forward chain from the initial conditions and backward chain from the goal until a common middle term is produced. (4) Middle term chaining--using the KB, guess a middle term and solve separately the problem of getting from the initial conditions to the middle term and from the middle term to the original goal; continue in this manner until a solution in terms of primitives is generated. (This method is also called problem reduction.) In parsing systems, method 1 is often called bottom-up and methods 2 and 4 are called top-down strategies.

Figure 4.27 shows an example of the first three of these techniques. The KB contains three rules: (1) any integer, X, can be replaced by 2X ($X \rightarrow 2X$); (2) any even integer, 2X, can be replaced by X ($2X \rightarrow X$); and (3) any integer, X, can be replaced by 3X+1 ($X \rightarrow 3X+1$). The problem is to transfer 4 into 20 using the permitted operations. The top figure shows forward chaining--i.e., start with 4 and apply the operators until 20 is produced. The middle figure shows back chaining--i.e., start with the goal, 20, and use the inverse of the above rules and continue until 4 is produced. The bottom figure shows the chain-both-ways technique. First, one step of back chaining produces the nodes labeled 10 and 40. Then one step of forward chaining produces the nodes labeled 8, 2, and 13. Finally, one more step of back chaining is done to produce the nodes labeled 5, 3, 13, and 80. Since 13 is on both the forward and backward grown "wave fronts", the process can terminate; otherwise, the steps of forward and backward chaining would continue to alternate until either a solution was found or the system gave up.

Another method of classifying a CE is by its directionality. This type of classification is used only when the problem input is linearly ordered, such as the waveform input to a speech recognition system or the two-dimensional array of picture information for a vision system. There are two major varieties: fixed directionality and variable directionality. Fixed directionality is usually described by terminology such as left-to-right or right-to-left. The idea is that the system processes its input data in the predetermined direction

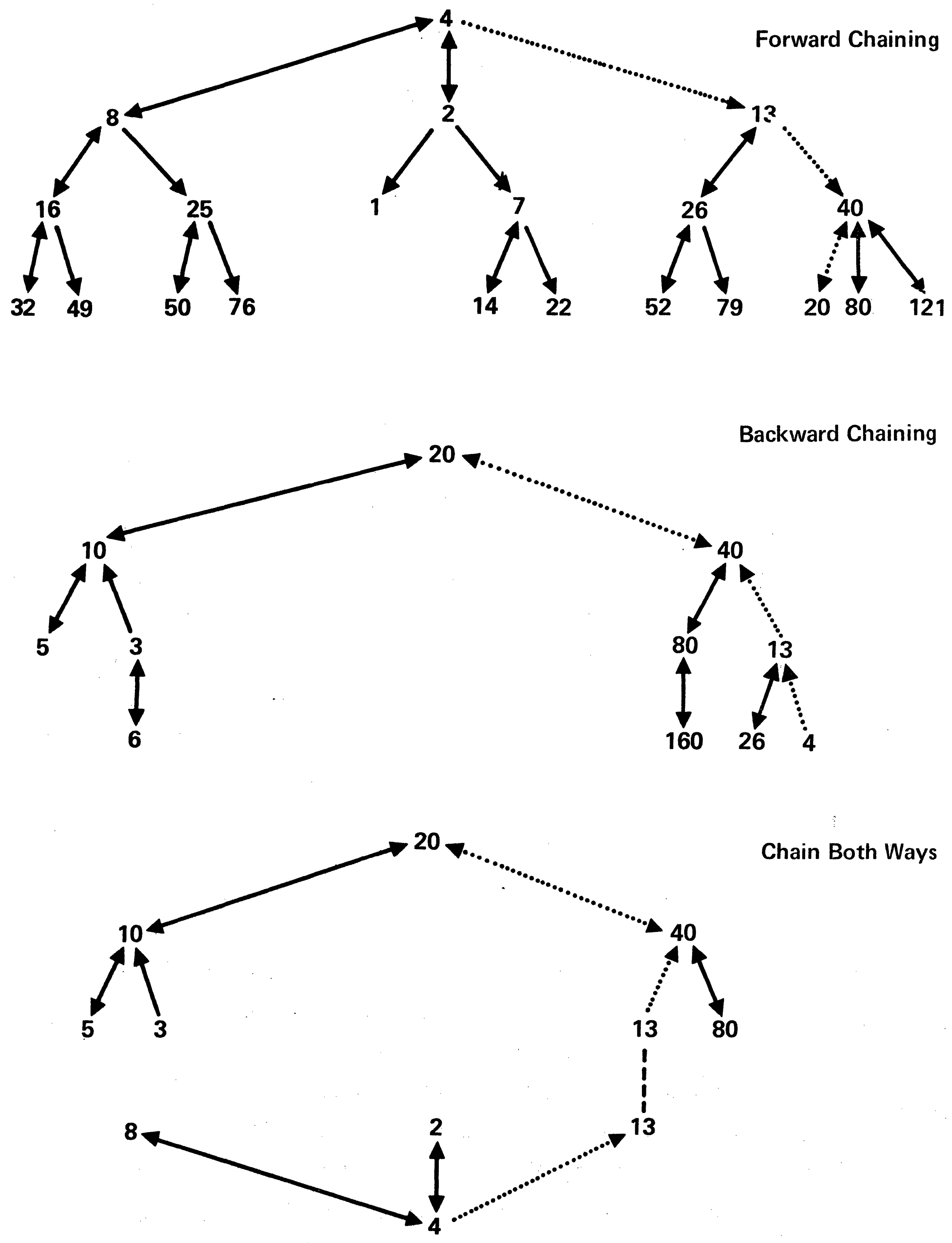


Figure 4.27. Chaining Examples

until either (1) all data have been consumed and the problem successfully solved or (2) a block is reached and no further progress can be made. In the latter case, the system backs up to a point before the block occurred at which an option was available. At this point, an alternative path is assumed, and processing of the input is continued in the original direction. This technique is iterated until either the problem is solved or no more alternatives exist.

A completely variable directionality in a system is often called island driving. The idea is to start processing the input at the point or points deemed to be the least ambiguous or contain the most robust clues as to their identity. These "islands of reliability" are then grown, middle outward, until they collide or a block occurs. If a block occurs, another set of start points are determined in the unprocessed areas. The rationale behind island driving is that by starting in areas containing the more certain information, part of the combinatorial explosion of fixed-directionality schemes will be avoided because backup will rarely occur across the islands, but only between them. Mixed strategies have been tried in several systems; e.g., proceed in a fixed direction until the system gets caught up in significant backup activity. At this point, hop forward in the fixed direction of processing and attempt to locate a good region to work in. After completing processing of the forward region, fill in the hole. This often helps because the region that caused the backup is now bounded on both sides and because the contents of the bounding regions may supply additional clues about what ought to be in the hole.

A final way of differentiating CE strategies is via the terminology breadth-first vs. depth-first. In a breadth-first system, all possible methods of continuing are attempted in parallel. This is exemplified in Figure 4.27, where each (horizontal) level of the graph was generated by a single cycle of the system. In a depth-first system, some path (node, state, etc.) is selected and a single continuation is attempted, i.e., the node is not fully expanded all at once. This path continues growing until either the path reaches a solution or some path-length constraint is violated. In the latter case, the path is backed up to the deepest node at which an alternative expansion exists. At that point, another path continuation is generated. This process continues until either a solution is produced or the alternatives that could produce a solution within the length constraint are exhausted. A depth-first strategy can be more efficient than a breadth-first one if a good technique exists for ordering production of path extensions. Figure 4.28 shows an example of a depth-first strategy combined with back-chaining for the prior problem. A maximum path length of 4 was used as a constraint, and the order of (inverse) operator application was (1) $N \rightarrow 2N$, (2) $2N \rightarrow N$, and (3) $N \rightarrow 3N+1$. Each node has a superscript that denotes the order in which the nodes were generated.

For further information on the concepts discussed in this section see: [P. KLAHR77], [MILLER73], and [NILSSON71].

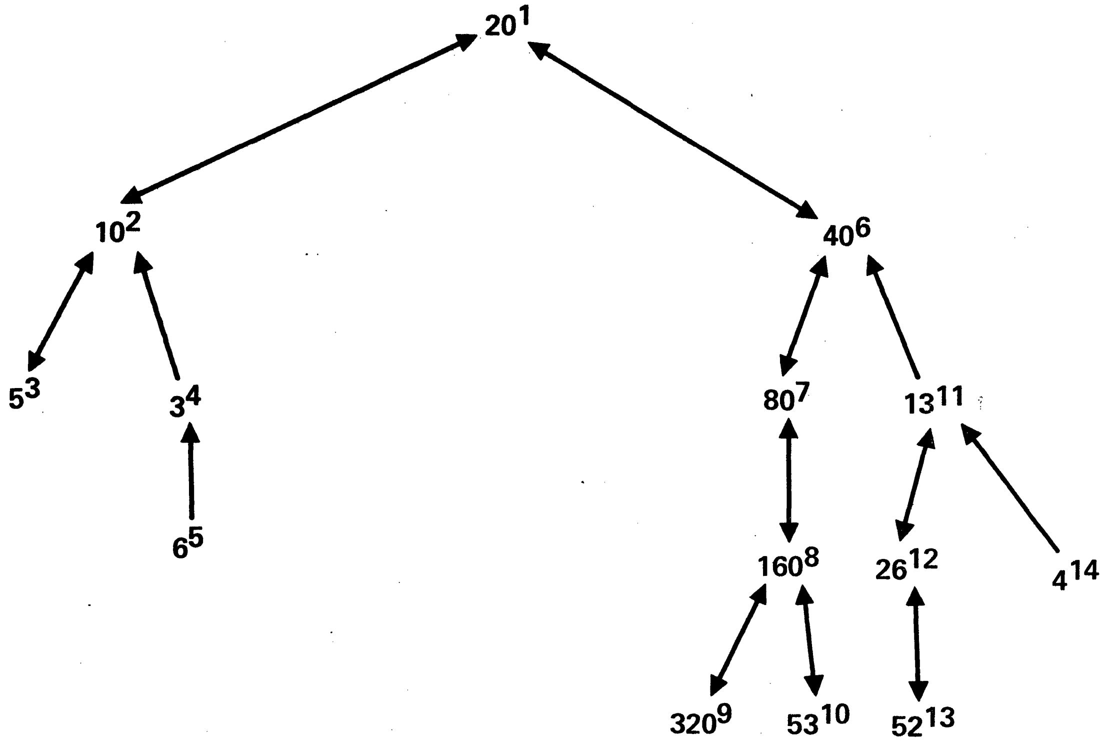


Figure 4.28. Depth-First Back Chaining

4.3.2 Methods of Implementing the CE

Most methods and techniques used to implement a CE are intimately married to the choice of a representation technique for the KB. However, there are a few methods that are general enough to be used with a variety of KB representations. Two classes of them are discussed below: search methods and modeling/simulation methods. The former appear in some way in nearly all KB and AI systems; the latter are used as an alternative to closed-form solution or search when efficiency considerations so dictate. Pattern-matching techniques, though widely used, are not described herein. For a general introduction to the topic, see [KANAL68].

4.3.2.1 Search Techniques

Search techniques form the core technology of KB and AI systems. It is a cover name for a variety of methods used to look for problem solutions in an orderly manner. The field is today in a state where, for most problems, a simple, albeit inefficient, CE can be constructed that represents any point in the space with dimensions of: (1) depth or breadth first; (2) chaining methods--backward, forward, etc.; and (3) directionality--fixed or island driven. In fact, it is usually possible to construct a CE for points in the above space that are sound, complete, or admissible.* The set of CE algorithms that makes this statement true is domain independent. In general, this means that each is subject to the effects of combinatorial explosion. Thus, the real problem with search technology is not merely to find an algorithm with a specified set of characteristics, but to find one that is efficient and does not suffer from combinatorics when handling problems in the intended area of application. To accomplish this, it is necessary to incorporate domain-specific knowledge.

*Without some problem-specific knowledge, some choices are incompatible; e.g., depth-first is generally not compatible with completeness or admissibility.

In order to describe places where such knowledge can be used, it is first necessary to examine a paradigm of the search technique. Search consists of five major components: (1) select--pick the next activity to be performed from the agenda of possible next activities, (2) expand--perform the selected activity, which often means enumeration of some or all of the predecessor activities; (3) evaluate--compute merit scores for activities created by the expansion process; (4) prune--discard hopeless cases or those that appear to have little promise; and (5) terminate--determine whether to continue processing and whether the problem has been sufficiently solved. Given these five components of a search method, it is obvious that a KS providing accurate guidance (incorporating domain-specific knowledge) for each can improve system performance, often by orders of magnitude.

In many search methods, the selection, evaluation, and prune (if any) are combined into a uniform numerical technique. The function used for this purpose is called an evaluation function. Its job is to estimate the likelihood, f(x), that activity x will be useful in finding a solution. Figure 4.29 shows a simplified algorithm for finding a solution to a search problem using an evaluation function, f; a termination checker, solution; and an expansion function, expansion-of. (As shown, no provision is made for picking out the solution path; provision is made only for checking to see whether one exists.) If the evaluation function were perfect, then the only activities expanded would be solutions or activities with at least one descendent that is used in the ultimate solution path. Assuming that the termination checker, solution, is accurate, this algorithm is sound no matter how bad f and expansion of are. However, f either is poor or the domain admits of unbounded path lengths, then a solution may never be found. Thus, even though the algorithm is sound, unless expansion-of and f are spectacular, it is neither complete nor admissible.

```
        IF solution(start) THEN exit with success;
        $b \leftarrow$ list({start, f(start)});
loop:   IF empty b THEN exit with failure;
        $n \leftarrow$ member of b with highest f value;
        delete n from b;
        add n to m;
        $s \leftarrow$ expansion_of(n);
        FOR x IN s
            DO IF solution(x) THEN exit with success;
               IF x in b OR x in m THEN do nothing
                  OTHERWISE add {x, f(x)} to b;
            END FOR;
        GO TO loop;
```

Figure 4.29. Search with Evaluation Function

An algorithm that improves on the above by being admissible and in some cases optimally efficient is called A*. The evaluation function, f(x) is the cost of a solution path constrained to go through node x; hence, its value is to be minimized. Further, f is assumed to be additive in the cost of going from one node in a path to another. Thus, if start = $n_1 \ldots n_m$ = goal is an optimal solution path, then

$$f(n_i) = \sum_{j=1}^{m-1} K(n_j, n_{j+1}) \qquad 1 \leq i \leq m$$

where K(x,y) is the cost of going from state x to state y in one step. For any node, n, f can be expressed as

$$f(n) = f(start,n) + f(n,goal)$$

where f(x,y) is the minimal cost of a path (of perhaps many steps) from x to y. The above is normally written as

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

where

$g(n) = f(start,n)$ and $h(n) = f(n,goal)$

The function, g, is relatively easy to calculate during search. However, h is not, because it presumes knowledge of the part of the search not yet completed.* The approach to overcoming this difficulty is to use an approximation, called a heuristic estimator, $\hat{h}$, in place of the exact function, h. (For some applications, an estimator, $\hat{g}$, is used in place of g as well.) Thus, f is approximated by $\hat{f}$ as

$$\hat{f}(n) = g(n) + \hat{h}(n)$$

The A* algorithm is given in Figure 4.30. To guarantee admissibility, a necessary condition is that $\hat{h}(n) \leq h(n)$ for all n. Another necessary condition on $\hat{h}$ is that $\hat{h}(x)-\hat{h}(y) \leq k(x,y)$.** Given a particular choice of h, this algorithm has been proved optimal in the sense that no other admissible algorithm will expand fewer nodes.

*The terminology assumes a forward-chaining search; for backward chaining, g and h reverse roles.

**This is called the consistency condition. Without this constraint, A* will still be admissible but no longer optimal.

```
        b ← list ({start, $\phi$, 0, 0});
loop:   n ← member of b with smallest $\hat{f}$ value;
        IF solution(n) THEN exit with success;
        delete n from b;
        add n to m;
        s ← expansion_of(n);
        FOR x IN s
            DO g ← third(x) + k(n, x);
               $\hat{f}$ ← g+$\hat{h}$(x);
               IF x already in b
                  THEN  IF new f < old f
                              THEN replace old f with new f
                              ELSE do nothing
                  ELSE add {x, n, g, f} to b;
            END FOR
        IF not empty b THEN GO TO loop;
        exit with failure;
```

Figure 4.30. A* Search Algorithm

The notation {u,v,w,x} in the figure is a data-structure representation of a node in the workspace--u is a state description, v is the prior node (from which this one was derived), w is the value of g for this node, and x is the value of $\hat{f}$ for this node. As opposed to the algorithm shown in Figure 4.29, this one keeps a back-trace on history of the solution path (using the v field).

The final example of a search technique given in this section is the alpha-beta algorithm. It is normally used for searching game trees and AND/OR graphs, respectively, for best moves or minimal-cost solutions. The description herein is for a two-person competitive game, e.g., tic-tac-toe. Then a game tree (simple move graph), like that shown in Figure 4.24, can be generated so that alternating vertical levels correspond to moves by each of the players. Suppose, further, that the game is complicated enough so that the entire game tree cannot be generated and analyzed (for example, it has been estimated that there are about 10^{120} unique chess games). An approach to this kind of problem is to limit the length of the move sequences that are examined either to a fixed maximum length or by an evaluation that can increase the length somewhat by the intensity of the position generated after a fixed-length look-ahead. After a length-limited path has been generated, a position is reached that usually does not represent the termination of the actual game. Therefore, the value of the position reached can only be approximated--such an approximation function is called a static evaluator. Given this setup--a move tree and a static evaluator--the alpha-beta algorithm can be used to prune away some of the move sequences that must be examined.

Consider the move tree fragment shown in Figure 4.31. Nodes represent possible (board) configurations and the arcs are labeled with move names. (Values of the game are written in the nodes.) The two players are named MAX and MIN. It is MAX's turn to move, and he wishes to find the move that promises the largest value of the game (as calculated by the static evaluator). MIN, on the other hand, is naturally trying to make the value of the game as small as possible. Now assume that move A has been evaluated, and it has been found that the value of the game is 4 if both MAX and MIN make their best moves from that point on. Now consider move B by MAX. MIN is shown to have two possible responses, moves C and D. C has been evaluated and has a value of 3. Therefore, if MAX chooses move B, MIN can guarantee that the value of the game will never be more than 3 (by choosing move C). A smart MAX would not choose such a move when move A is available to him. Hence, there is no need to evaluate move D or, for that matter, any other continuations of move B. In a nutshell, this is the essence of the alpha-beta algorithm. Of course, this pruning activity (e.g., not evaluating move D or its consequences) is done on the whole game tree as it is generated and is done in a dual manner for subtrees where MIN's moves emanate from the root.

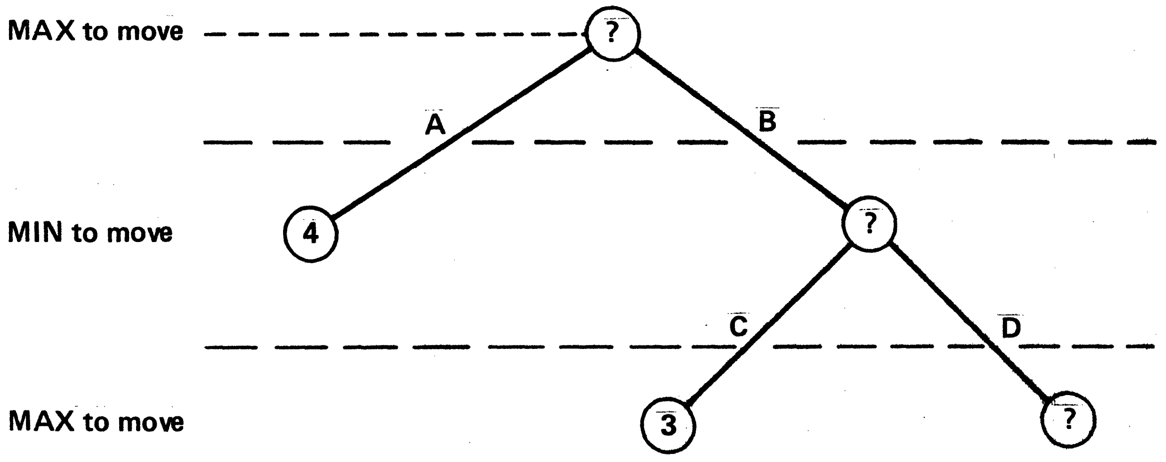


Figure 4.31. Alpha-Beta Pruning Example

As is obvious from the example in Figure 4.31, the order in which moves are considered can have a strong effect on pruning. For instance, if move B is analyzed before move A, then move D would have to be analyzed because the acceptable lower bound (4) would not yet have been generated. The pruning activity of alpha-beta does not itself use any domain-specific knowledge. Such knowledge can, of course, be used in two places: (1) in the static evaluator and (2) by establishing the order in which moves are considered. If the moves are ordered in the best possible manner (i.e., best from each player's point of view when it is his turn to play), then the static evaluator will be called only a number of times approximately proportional to $B^{L/2}$, where L is the look-ahead depth and B is the average branching factor in an unpruned move tree. Without alpha-beta, the number of static evaluations is approximately proportional to B^{L}. The merit of the move ordering determines where in this range the search algorithm will perform.

It is interesting to note that, for a given static evaluator and a fixed polic on path length before using it, alpha-beta is an admissible search strategy; that is, it finds the best move. This algorithm has many uses other than game playing. It can be applied to any situation where conflict can be identified and strategies are sequences of steps, each of which has consequences, and impacts the merit of the entire plan. Because of the potential pruning power, it often pays to formulate a problem in such a way that alpha-beta can be directly incorporated.

For more information on search methods, the interested reader should consult: [FULLER73], [GELPERIN77], [MINSKY63], [NILSSON71 and 69], [SIMON74b], and [SLAGLE69].

4.3.2.2 Modeling and Simulation Techniques

A model is a representation or abstraction of an entity. Thus, a KB is a model. Section 4.1.2.3 gives a formal definition of a model (also called an interpretation) using the predicate calculus. A model has components that correspond to "real word" things, properties of things, and dependencies among them. The properties and dependencies are of two kinds: (1) generic--something that is true for all members of a class, e.g., the behavior of any resistor obeys Ohm's law, and (2) specific--a property or dependency that is only true in a particular problem, e.g., the temperature of x is 10^{0}C. The division between generic and specific model information is a somewhat arbitrary but useful distinction because the generic information can be used to solve a variety of problems. In a sense, the generic information is a KS and the specific information is a combination of fact file(s) and problem-specific input parameters.

A simulation is a procedure for manipulating a model in a manner consistent with its properties and dependencies. The generic dependencies are stated as transformation rules in terms of the specific information. The model has a given initial state, and some independent variable, usually time, is moved towards a goal by the simulator. The transformation rules describe changes in the model, i.e., new states of the entities in terms of prior states of themselves and other entities. Available data from the simulation process are the intermediate as well as final states. Thus, modeling and simulation provide an ideal method of observing behavior because all reached states are necessary and reflect conditions of the entity being modeled (assuming the model is fateful). (Contrast this to search techniques where the set of generated states may or may not be realistic.)

A simple example is now given to clarify some of the above concepts and terminology. The model is of a domain that contains three kinds of components:

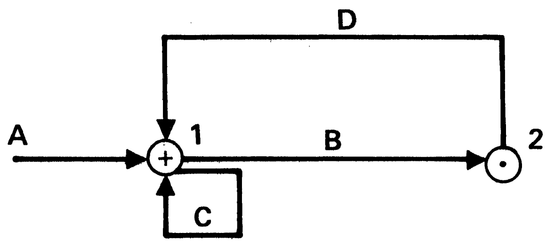
(1) $\oplus$ --a device that has one or more inputs. The output of $\oplus$ at time t+1 is the sum of its inputs at time t.

(2) $\odot$ --a device with one input. Its output at time t+1 is its input at time t.

(3) Wires--devices used to connect together the inputs and outputs of the other two kinds of components.

The generic properties are those described above. The specific information is the configuration of a circuit built from these kinds of components.

Figure 4.32 shows one such circuit. The top of the figure is a schematic. The specific information is that there is one component of type $\oplus$ (labeled 1 in the schematic), one component of type $\odot$ (labeled 2 in the schematic), and four wires (A, B, C, and D) with the connections shown. Wire A comes from the outside and carries a value of 1 at time 0 and a value of 0 thereafter. The other wires have an initial value of 0. Using generic information, the simulation equations can be derived. For this simple example, these equations can be solved to yield $B_i = C_i = F_i$ and $D_i = F_{i-1}$ where F_i is the ith Fibonacci number. However, for more complicated systems, the derived equations



Initial Conditions

$A_0 = 1 \quad A_i = 0 \quad 1 < i$

$B_0 = C_0 = D_0 = 0$

Specific Information

$\oplus$ 1

$\odot$ 2

Wire $A\{\ ,1\}$ $B\{1,2\}$ $C\{1,1\}$ $D\{2,1\}$

Simulation Equations

$B_i = C_i = A_{i-1} + C_{i-1} + D_{i-1}$

$D_i = B_{i-1}$

Time	0	1	2	3	4	5	6	7	8	9	10
A_t	1	0	0	0	0	0	0	0	0	0	0
$B_t = C_t$	0	1	1	2	3	5	8	13	21	34	55
D_t	0	0	1	1	2	3	5	8	13	21	34

Figure 4.32. Modeling and Simulation Example

may not be solvable in closed form, so some form of iteration or simulation technique would need to be used. The bottom of the figure shows the output of a simulation run. The value on each wire is given as a function of the independent variable, time. Even for such a simple example of this, where a closed-form solution is available, a simulation technique can be valuable--for instance, to examine system behavior in errorful conditions such as glitches on one of the wires. It may in fact be faster and cheaper to perform the simulation than to solve the set of difference equations, particularly if the errors are introduced via a statistical technique.

In one sense, all of AI and KBS technology is a modeling and simulation technology. The intent of systems built under these banners is, with some exceptions, either to duplicate an expert's performance (perhaps with some improvements) in a particular problem domain, or to simulate some aspect of human mental activity and behavior.* In spite of this affinity, only a few KB and AI systems directly borrow technology from the modeling and simulation field. There are many reasons for this, the most important of which are that no well-formed theory exists for the entities and kinds of situations that are modeled, and that size and connectivity are too large for the use of off-the-shelf techniques. It can further be said that many KB and AI systems themselves are models, and it is not necessary to use simulation methods to make this true.

It is interesting to note the pervasive use of the word model in discussing KBS. For example, the term word model is used to describe a computer's internalized representation of a situation or state in the external environment. The terms semantic model and memory model are used to describe the underlying assumptions in a workspace and/or knowledge-representation scheme. Also, the terms performance model and model of problem-solving behavior

*The systems of the latter type belong to a subfield of AI called psychological modeling.

(or simply behavior model) are used to describe assumptions built into the processing components and organization of a system. The list of such terms is virtually endless. In fact, it is our guess that if all the articles written about KB and AI systems were researched, more than half of them would use the terms modeling and/or simulation. We would even go on to guess that about half of these articles coin a new term, to describe their "unique" approach, that includes the word model.

As stated above, modeling and simulation techniques have an advantage over search techniques by making behavior more visible. Another advantage is efficiency. Sadly, however, many, if not most, KBS problems do not lend themselves to simulation, because these techniques demand a set of transformation rules that can move the system model from one state to the next in an orderly and fateful manner. In a way, simulation is a middle point on a scale, with closed-form solution and search as the end points. In this light, an interesting trend is developing--special-purpose languages developed for AI and for simulation are beginning to look more like each other, with SIMULA and its derivities being used in both fields. To the marriage, AI is bringing more flexible control structures (so that inexact transformation rules can be tolerated) and better data representations, while simulation is contributing proven efficient and workable technology.

The interested reader should consult a text such as [REITMANN71] for a general introduction to simulation methodology. For a description of what we believe to be the most interesting of the special purpose simulation languages, see [DAHL73]. Other articles of more direct relevance to KBS are: [BOBROW75b], [BROWN75b], [GRIGNETTI75], [MARK76], and [MORAN73a].

4.3.3 CE Issues

There are many open problems associated with methods of implementing a CE for a knowledge-based system. However, most of these problems and issues are a dual of a problem or issue in representing a KS. For example, a CE problem, focus of attention, concerns itself with the issue of how to guide search so that the system expands its resources on important things and ignores others. The corresponding KS problem is methods of representing higher-level meta and control knowledge.

In some sense, the KB can be viewed as the encapsulation of knowledge in a KBS using orderly and consistent theories, and the CE is what is left over. Thus, the CE is the representation of all the knowledge that could not be captured appropriately in the KB but must still be approximated somewhere in the system in order to solve problems. It is no surprise, then, that the open problems are duals. Therefore, it is clear that the major problems and issues in KBS technology are those concerning the KS and their techniques of representation.

A CE problem that is very mundane yet important is knowing when to give up. For a domain without a decidable problem-solving procedure (such as theorem proving), the issue is theoretical. But even in a domain where all problems could be solved given enough resources, it may not be worth the effort because of cost. A generalization of this is how to terminate fruitless paths during search, e.g., knowing when to give up a subproblem. This is slightly different from the focus-of-attention issue, which asks for methods that refrain from ever starting down bad paths.

Another CE problem is knowing how to approximate or reason plausibly. An examination of the preceding sections leaves one with the impression that a KBS reasons plausibly, because the contents of the KS are an approximation. The actual reasonsing techniques in the CE are fairly brittle and formal.

Thus, the issue is how to reason plausibly as a CE option whether or not the KB is exact. Interrelated with this is development of procedures and the mathematical theory for generating and using confidence factors as merit estimators.

A final issue, raised by Moore and Newell [MOOREJ73], is that there exists no space of problem-solving methods. Only a handful of general techniques are now available. Further, no good taxonomy exists so that untried methods can quickly be identified as points in the space (spanned by the taxonomy). It may be the case that this is the way it must be; methods of applying and using knowledge are actually ad hoc, and no underlying general set of descriptors can have the necessary organizational power. If this is a valid statement, then KBS technology will continue to evolve slowly rather than taking a few quantum jumps to the "ultimate" systems.

4.4 THE INTERFACE

The interface component of a KBS provides the necessary connections and communications with the environment and user set. It is not always engineered as a separate module in the system; usually it is integrated into the CE and accesses the KB. This section, for reasons of clarity, discusses the interface as if it were distinct. A good introduction to the topics covered in this section may be found in [DAVIS76].

The interface has three logical parts: the user interface, the expert interface, and the external data interface. The user interface accepts problem statements from the user, his responses to system generated queries, and his requests for explanations. Further, it displays results, prompts the user, and provides him with the requested explanations. The expert interface is the system's port for knowledge acquisition and is, therefore, used to augment or modify the KB. It is also sometimes used by an implementor for system debugging. The external data interface is used to input problem parameters, communicate with external files, and send results or directions to other automated systems. Since the functions of the external interface do not generally make use of singular or interesting technology, they are not further discussed below. However, this interface will grow in importance as KB systems start working in expanded problem domains. For example, a complete medical KBS for hospital use would be connected with pharmacy records, patient history files, patient monitoring devices, duty rosters, etc.

Usually, the user and expert interfaces communicate through a common physical device such as a time-sharing console. Also, the entire interface borrows many capabilities and services provided by the host operating system, such as access methods and editors. The interesting technology associated with the interface is not those; rather, it is the methods used to transform between the human-engineered external formats and the more intricate internal physical and logical formats needed for the CE, KS, and workspace representations.

4.4.1 The User Interface

The user interface is the most important component of a KBS in determining its acceptability to the practitioners of the intended domain. In general, they are neither computer scientists nor programmers, and may view the computer, especially a KBS, as a feared and unworthy competitor. The quickest way to prove the point is by demonstrating simple linguistic stupidity at the interface. Besides guarding against (perhaps justified) paranoia, a properly functioning user interface will smooth out and minimize the problems associated with learning any new system, and in the long run improve system productivity by making it possible for the users to be more cooperative in problem-solving activities.

In order to qualify as a KBS, the user interface must, at a minimum, be interactive and use domain-specific jargon. The reason for requiring the system to be interactive is simply that the state of the art does not provide techniques for going from problem statement to "best" answer without additional information that must be solicited from the user. The problem with providing initially, along with the problem statement, all information that might conceivably be needed, is that most of it is unnecessary. Such a procedure would be a waste of time and irksome to the user. Another reasoning for wanting a KBS to be interactive is so that explanations of system behavior and results can be solicited. Even though we have not made availability of explanations a prerequisite for calling a system a KBS, we strongly feel that without such a capability, no system will gain field acceptance. Such a system can be of immense interest and value to a computer scientist but still not fulfill the purpose of a KBS--namely, to be used in a problem-solving domain by practitioners in that domain.

Besides using domain-specific jargon, many KB systems accept and output information using an English-like natural language. Since handling natural language and all of its complexities is equivalent to solving the entire problem of machine understanding and natural intelligence simulation, it should not be expected in a KBS. Fortunately, some inherent constraints and simple methods make the implementation of a suffucient subset tractable. An example is that the system usually takes the initiative in asking questions. When questions to the user are of the form "What is the value of x?" rather than "What's new?", understanding the input is relatively straightforward. Also, much of the ambiguity of natural language disappears when the dialog is restricted to a particular domain and it is known that the user is engaged in goal-oriented problem-solving activity.

Another desirable characteristic of the user interface is soft failure. That is, a KBS should not blow up because the user makes a mistake, nor should it conceal its problems. For example, when an input is not fully understood by the system, the user should not only be told, but should also be given guidance as to what are acceptable responses. A useful technique to smooth over some problems of this sort is a spelling corrector. (See [TEITLEMAN72].)

Besides the above-mentioned capabilities of the user interface, it is desirable that it be able to provide some self-knowledge to the user. By this it is meant that the system be able to explain how it is used and that the user be able to ask questions such as "Can you handle problems about x?" or "What do you know about Y?" A system with self-knowledge available has the potential to accommodate new users in a reasonable manner.

4.4.1.1 User Input

Several techniques have been used to implement the input side of the user interface. The simplest one, mentioned above, is for the system to maintain the initiative so that the form of the user's input can be anticipated with a great deal of certainty. Another, ad hoc but reasonably powerful, technique is that introduced in ELIZA [Weizenbaum 66]. ELIZA is a system that (humorously) simulated a Rogerian psychiatrist. Inputs are matched to patterns like

$$\$_i\ x_i\ \{\text{IS}|\text{ARE}\}\ \text{NOT}\ \$_2.$$

where $\$_i$ matches any string of words and x_i matches any single word. Responses are built up by giving corresponding output patterns such as

WHAT IF x_1 WERE $\$_2$?

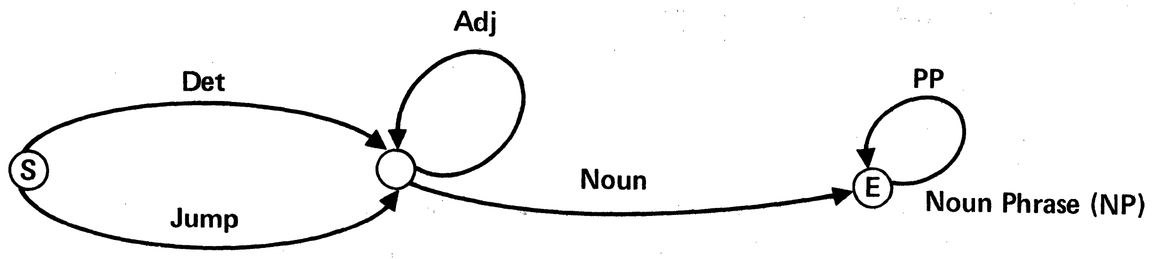
Given the input "Joe's wife, Mary, is not at home", the system could produce the response, "What if Mary were at home?" This is accomplished by matching $\$_2$ to "Joe's wife," x_1 to "Mary," and $\$_2$ to "at home." ELIZA and other systems using this kind of matching technique can do a better job of "understanding" than one might imagine as long as the domain and style of dialog is sufficiently constrained so that the pattern writer can properly anticipate what kind of language will be used.

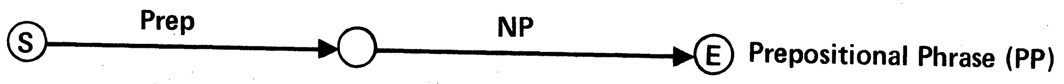
More powerful techniques using better language models are also available. In fact, the purpose of many AI systems is just to deal with natural language input. See, for example [NORMAN75], [SCHANK75a], and [WOODS71]. Perhaps the best known and most widely used technique for parsing natural language is the Augmented Transition Network, ATN (See [WOODS73 and 70]). A simple transition network is depicted in Figure 4.33. The top of the figure shows the grammar definition for a simplified version of a noun phrase (NP) as a finite-state machine. The initial and final states are labeled, respectively, S and E. State transitions are allowed when the next word in the input is of the part of speech shown on the labeled arrows. (A jump means that a transition is made without using an input word.) A sequence of words that causes transition from the initial to the final state is accepted. A parse of the input is generated in a straightforward manner. For example, the input

"the big brown cow"

would cause the output parsing

[NP [DET the] [ADJ big] [ADJ brown] [NOUN cow]]





Det	A Determiner
Adj	An Adjective
Prep	A Preposition

Figure 4.33 Transition Network Example

State transitions can be made for reasons other than word class. The transition network for an NP allows an optional determine (DET) followed by zero or more adjectives (ADJ), followed by a NOUN, followed by zero or more prepositional phrases (PP). A PP is itself defined as the network shown by the bottom figure. Thus, a state transition can be caused by finding a phrase, accepted by another named network, as well as satisfaction of a word class. Using the full definition, the input

"the brown cow in the red barn"

can be parsed as

```
[NP [DET the] [ADJ brown] [NOUN cow]
    [PP [PREP in]
        [NP [DET the] [ADJ red] [NOUN barn]]]]
```

As described so far, transition networks cannot handle many of the frequent complexities that occur in natural language. To behave in a satisfactory manner, the network must operate in a nondeterministic manner, so that ambigui- ties can be resolved. Also, the transition network is normally augmented (the it is called an ATN) with a set of registers associated with each usage of one of the graphs (like the two shown in the figure). As states are entered, the values of the registers can be set, and tests of these register values can be added to the state-transition conditions. This makes it possible to enforc such constraints as number agreement; e.g., accept "he goes" but not "he go". Also, the register test and sets can use information from a definitional knowl- edge source (usually a semantic network) help to disambiguate phrases such as "typed blood".

Recently, a class of methods for understanding natural language has been developed that use no explicit syntax, but instead rely on a semantic abstrac- tion of the problem domain. For example, see [BURGER77 and 75]. A system is described that uses an abstract of a data base description including the inter- relationships of important words (e.g., item-names). This abstraction, along with knowledge of English key words (e.g., of) forms a parser. This kind of technology has the advantage of being efficient and easy to use in a variety o domains. It works well as long as the domain is reasonably bounded (like a front end to a KBS or data management system) but would not appear to be exten- sible to more unrestricted areas.

Another technique for very efficiently handling natural language input in a restricted domain is described in [BROWN75]. The approach is to represent the grammar as a set of context-free rewrite rules. However, the terminals in the grammar are concept classes rather than word classes. (Such a grammar is called a pragmatic grammar.) For example, a concept such as "something that can have a voltage measurement" is used instead of NOUN. The parser has been implemented as a set of procedures coded by hand from the grammar. (One possibility is to use standard meta-compiler methods to make an automatic parser generator.) A clear advantage to this methodology has been the ability to implement a sufficient natural language input capability without getting lost in the general details of language.

4.4.1.2 Output to the User

The other half of the user interface is responsible for output generation. This is needed to query and prompt the user, report results, give explanations, and answer questions about the system. Except for providing explanations, these tasks are easy relative to handling natural language input. In the first place, many output messages are merely formatted data items and tables and thus involve no particularly novel techniques. Further, many output messages can be concocted by simple fill-the-hole techniques with canned templates. (See the ELIZA example in the previous section.)

Providing explanation to the user of the system's behavior is difficult whether or not the output is in natural language (though it usually is). The reason for the difficulty is that the explanation must be in terms of the knowledge chunks, problem parameters, and inference rules used to derive the results. Thus, the explanation mechanism must be able to find these things. Next, the internal representations must be translated to a format suited for human consumption. This is made harder by the fact that not all of the things are of equal importance, and a good explanation mechanism ought to identify and output only those that were most relevant or crucial for solving the problem at hand (unless asked for additional detail, in which case the system should respond appropriately). An aid to providing good explanations to the user is appropriateness of chunk size. That is, the ability to provide acceptable explanations is based upon communication being at the right "clip" level. If the knowledge chunks used are too small, the explanation is laborious and not convincing. On the other hand, if the chunks are too large, they may not appe to apply directly to the solution being described.

The most direct method of capturing elements that are useful for explanations is to use the workspace representation to store a history of the problem-solvi activity. The mechanism can then start from the element(s) of the workspace representing the problem solution and pick out the sequence of events that mov the system from problem definition to solution. Ideally, each element would include as part of its history the rule of inference and what the rule was applied on (other workspace elements, knowledge chunks, confidence factors, etc.) to produce this element. Such an approach has the advantage of making all useful information available to the explanation mechanism, including infor mation about why other solutions were rejected. Further, the history collection can be done uniformly by the CE. The disadvantage is the cost of storing information that may never be used.

Another approach is to let the KS determine what may be most relevant for an explanation. With this method, a knowledge chunk can optionally have an explanation scheme. If it is used to produce a result that is in question, th scheme is instantiated in its local environment to produce an explanation. Th advantages of this are: (1) high-quality explanations can be produced becaise it is possible to take idiosyncratic situations into account, and (2) the explanation mechanism can be used for other purposes, e.g., as part of the complaint depart for a frame. The major disadvantage is that the expert who imparts knowledge to the system must consider the method and necessity of explaining each knowledge chunk--an arduous task.

A third method of providing explanations is to initially solve the problem without keeping a history in the workspace. Then, if the user asks for an explanation, re-solve the problem in careful mode. That is, the mechanism looks over the CE's shoulder during the re-solving activity and picks out the events that are of likely interest. In a sense, the explanation mechanism enables a set of demons that are triggered when special situations occur. At these points, the explanation mechanism can interrupt normal processing to perform the necessary data collection. The disadvantage of this is the inefficiency introduced into the CE so that demon-like execution can occur. The advantage is a possible gain in efficiency if explanations are only seldom needed. Not enough is yet known to evaluate the tradeoffs.

The final problem confronting the explanation mechanism in the user interface is translating explanations into a natural form for the user. Fortunately, the complexity of this task is substantially reduced by uniformities of format in the KB and in the workspace. For example, in a production system, it is straightforward to turn an IF-THEN rule out in reasonable English (see Section 3, for an example). The usual approach is to have a separate scheme for each kind of knowledge chunk in the KB and element in the workspace; most such schemes look like sophisticated fill-in-the-blank formulae.

The main problem in providing explanations is identifying the information to be included--the output formatting is only a secondary issue. When designing a KBS, one must decide early whether an explanation mechanism is to be incorporated. If so, techniques need to be built into the CE for information capture and for assuring that step-by-step behavior is explainable. Were this not part of the original design and specification, it is unlikely to be easily added later, because important parts of the system's behavior would probably be obscured by optimization and condensation. Good explanation mechanisms can only exist in a system whose internal step size is an explainable unit to the user.

Some articles about explanation mechanisms and natural language generation that are of interest are: [DAVIS76], [SHORTLIFFE76], [SWARTOUT77], and [WINOGRAD73].

4.4.2 The Expert Interface and Knowledge Acquisition

The expert interface is the mechanism through which knowledge is added to the KB or the KB is modified. Its intended users are experts in the problem domain and the system implementors who are responsible for building the initial KB. This interface is often called the knowledge acquisition interface. Unlike the user interface, it can be assumed that the user of the expert interface has some knowledge and awareness of the structure and functioning of the KBS. This does not imply that he is a programmer; rather it means that he knows that such things as knowledge are represented by IF-THEN production rules, or that confidence factors are integers in the range -100 to +100.

The remainder of this section describes the knowledge-acquisition process and the expert interface. The material is summarized in Figure 4.34.

The knowledge that goes into a KBS must originate from some external source. The most usual is an expert in the problem domain. He can provide specific facts, rules of thumb, and the rules of reasoning he employe, along with his rating of confidence. By our definition of a KBS, such an expert must be at least one of the originating sources. Members of the implementation staff for many KB systems are experts in the problem domain in which the systems operate and therefore provide much of the initial contents of the KB. In some systems (e.g., MYCIN [SHORTLIFFE76]), the user can also enter knowledge, as an expert, to tailor the system to his own particular needs.

Other originating sources of knowledge commonly used are journal articles, texts, and engineering handbooks. The information from these sources often is hard data and tabular. Therefore, it usually comprises fact files in the KB. In a domain where the kinds of knowledge found in these sources is sufficient for problem solving, a KBS is unlikely to be useful because a closed-form method is known and more appropriately implemented by conventional methods.

ORIGINATING SOURCE

Domain Expert (or User)
Journals
Texts
Protocol Studies
Derived Results

ENTERED BY

Domain Expert
Implementor
User
CE

COMPILED BY

Knowledge Acquisition Interface
Implementor

ISSUES

Interface Language
Consistency
Accommodation
Confidence Factors

MAJOR OUTSTANDING PROBLEMS

Knowledge Acquisition
Learning
Extensibility

Figure 4.34. Knowledge Acquisition

A technique used to collect expert knowledge is the protocol study. An expert is given problems to solve, and an experimenter either observes and records the expert's behavior or asks for explanations of various steps. The experimenter then analyzes the collected information and tries to determine general patterns, knowledge used, and principles of reasoning. Some interesting work has been done in this area. See for example, [COLLINS76], [DEUTSCH74], [DILLER73], and [MALHOTRA76,75a and 75b].

Another originating source for knowledge is from results derived by automatic means. One method is to remember problems that have already been solved by the KBS and to use them to answer further questions. Though this is done by some AI systems [SUSSMAN75] and [LENAT76], it is not done by any systems that fit our definition of a KBS. Such a technique is called computer learning. A problem with blind learning (saving all generated results) is that the system either runs quickly out of space, or runs into a combinatorial explosion of possibilities as the KB grows. What is needed is to save only those things that are interesting or important--this is an open research problem. Another approach to automated knowledge generation is exemplified by META-DENDRAL [BUCHANAN76]. The program is presented with a large set of solved problems. From this corpus, it infers production rules and confidence factors for them. The derived rule set is used by DENDRAL for solving new problems.

Ideally, the knowledge acquisition interface can be used by domain experts and users of the system other than the implementation staff. However, in many KB systems, the complexities of adding to or modifying the KB are such that programming skills are required. For such systems, a computer specialist may need to act as an intermediary between the originating source and the KBS. This is especially true or systems that contain large amounts of procedural knowledge. Domain-specific knowledge becomes intimately intertwined with control logic and programming knowledge that is part of the implementor's intuition instead of the expert's. This can lead to two problems: (1) difficulty in providing explanations of system behavior and solutions in domain-specific terms, and (2) loss of modularity and extensibility.

When a KBS does not make adequate provision for a domain expert to use the knowledge-acquisition mechanism, one should immediately find out why. If the reason is only to avoid the bother of making another human-engineered interfa(then the only problems that will result are those of needing a go-between to modify or extend the system. On the other hand, if the reason is that the cor ception of knowledge chunks, and representations thereof, by domain experts ar the KBS are radically different, then a major trouble area has likely been spotted. Namely, it is very possible that the system will not be extensible in ways that were not clearly understood initially by the implementors. (What you see is all you can expect.) On the other hand, if the KBS and experts agree on the knowledge model (representation) used, extensibility is clearly possible within that framework in ways that are known and obvious to the exper but not necessarily to the implementer.

The knowledge-acquisition interface has three major tasks: (1) accepting knowledge in external format and translating it into internal format, (2) vali dating consistency of new and old knowledge, and (3) storing the knowledge int the KB. This three-step process is called compilation. Often the translation component is built using a part of the input mechanism from the user interface and can handle restricted natural language.

The real difficulty begins with validation of consistency and checking for redundancy--a nontrivial task--particularly when confidence factors are included. If knowledge is represented as well-formed predicate calculus formulas, the methods for checking are straightforward. To check for redundan(simply try to prove the new knowledge from the existing KB. To check for incoı sistency, add the new to the old and try to prove something that is patently false, say $A \wedge \sim A$; if there is an inconsistency, the proof will succeed. For other kinds of knowledge representations, the checking is not so easy. Whethei or not the new knowledge is inconsistent or redundant (believe it or not, it cá be both when added to a KB that presently has neither problem) depends upon hov it is going to be used as well as what the knowledge is. For a more detailed account of the problems of maintaining consistency (see [McDERMOTT74].)

The third task performed by the knowledge acquisition interface is storing the new knowledge into the KB, a process called accommodation. See [JMOORE74]. A problem can occur if the system has several knowledge sources and fact files in the KB--where should it be stored? Even if the question of where is solved, the problem of how should not be underestimated. Storing can be a violent activity--it is not often that the representation of new knowledge is just copied into permanent memory and left for later use. Rather, the internal (physical) representation is usually a structure with links between chunks, and the acquisition mechanism must insert the new chunk into this plexus. For example, in MYCIN, each production rule that concludes something about feature F is linked to every rule that tests F in its antecedent. In most instances, the linkages computed by the knowledge-acquisition mechanism will determine how and when the knowledge is used during normal operation of the KBS. Thus, the insertion (as well as deletion and modification) of knowledge chunks can be a complex operation that depends upon many things such as confidence factors, conflict-resolution strategies, existing KB content, etc.

Surprisingly, today's principal outstanding problem with knowledge acquisition is not computer related. It is that most disciplines (other than mathematics and computer science) do not understand their own fundamentals in a formal way. Therefore, in order to build a KBS for that discipline, it is necessary to find a pool of expertise whose members are willing to rethink their methods and procedures. One catalytic method for starting this process is the protocol study mentioned above. The other outstanding problems are more technical in nature, the principal ones being (1) methods for systems learning to perform better by having solved similar problems, and (2) techniques to allow systems to be gracefully extended. The latter point is particularly important, even in the short term. If a system can easily be extended, then it can start performing useful tasks before all the knowledge necessary to it has been collected. This set of features (or lack thereof)--formalization, learning, and extensibility--together create a problem called the knowledge-acquisition bottleneck. The solution of this problem is key to bringing KBS technology to widespread dissemination.

5. APPLICATION CONSIDERATIONS

Though KBS technology represents a maturing technology, it has not advanced to the state where it is a paucity of writinq related to application selection And, though there is a fair body of literature on a variety of topics related to KBS technology, there is paucity of writing related to application selection and the design decision process. With minor exceptions (e.g., [BUCHANAN75]), there is no documentation of failures, thus denying the community one of the most potent educational opportunities--learning from the mistakes of others. Therefore, what follows will not, nor can it be, more than a set of general guidelines. Put another way, we do not believe that it is possible, given the present state of knowledge, to design and build a KBS that would support KBS developers in selecting and designing applications over the spectrum of potential applications.

The following is an amalgamation of material contained in [BUCHANAN75], discussions with KBS implementers, and our own observations resulting from experience with similar and related software developments.

Though there exists a relatively diverse collection of existing and developing KBS applications, the selection process for each new application requires consideration of a variety of issues. We have divided these into three major groups. First, there is a set of initial considerations that address the issues of the problem domain itself and the people associated with it, the experts and the practitioners. Next, are the technology considerations that focus on the availability of usable technology for implementing a KBS that has successfully met the first criterion. Last, there are the equally important considerations that are directed at determining whether or not the development environment and the user environment are properly supportive. Each of these groups is elaborated below in the form of a set of questions and the underlying rationale or concern that each addresses.

5.1 INITIAL CONSIDERATIONS

Does the problem have a closed-form solution? If a closed-form solution exists and can be implemented using conventional computer technology, then there is no reason to consider the problem as suitable for KBS technology. On the other hand, the closed-form solution may be so inefficient computationally, because of the number of steps involved or because of uncontrollable combinatorial explosions, that an algorithmic implementation is unthinkable. In this case, and when no known closed-form solution exists, the problem remains a candidate.

Is there an expert who knows how to solve problems in the domain? If there is no expert or group of experts to whom the typical practitioner would turn for advice or no one who is recognized as an outstanding performer for the type of problems involved, the likelihood of constructing a successful KBS is quite small and not worth considering. The existence of such an expert or experts is mandatory.

Is an expert available and can he be motivated to work on the development of a KBS? The existence of an expert is necessary but not sufficient. He must be an integral participating member of the development team. Without the full cooperation of such an expert, the effort is not likely to succeed. On the other hand, he must not be expected to become an expert in computer science and KBS technology. The computer scientists and technologists must be equally cooperative in meeting the expert at least half way. Each must be willing to learn the essentials of the others' discipline so that effective communication can be established. The lack of such cooperation may be difficult to determine before the actual implementation begins, but every effort should be made to assure it early.

There are several ways to impart the domain-specific knowledge to the KBS. One way is for the system implementer and the expert to work as a team, with the expert providing the knowledge and the implementer encoding and inserting it in the system; this can be a very lengthy and time consuming process, but is probably the best and possibly the only way to get a new application off the ground. Another way is for the implementer to provide an interactive subsystem (via the knowledge-acquisition interface) that allows the expert to impart knowledge to the system without intervention by anyone else; this would be diff cult to do without having partially used the first alternative to determine the specific design specifications for the interface.

Thirdly, a separate system could be built to abstract the knowledge from observ tions and experimental results. This would be a theory-formation program that could infer the rules about the domain from the data, as Meta-DENDRAL does. It is doubtful that such a system could be created without considerable experience for it requires modeling the theory-formation abilities of the experts. Such knowledge can be considered to reside on a higher conceptual plane than the problem-solving knowledge required for a KBS. One of the more difficult aspect of constructing such a system, even if it were deemed feasible, would be providing the necessary constraints that would limit it to generating only knowledge that is plausible within the theory, rather than all possible knowledge derivable from the data. The generated knowledge must not only be coherent within the theory, it must be consistent within itself.

Does the expert know--or have a model in his mind of--how he solves problems? Given that all of the above conditions can be met, one now faces the problem of determining whether or not the expert's problem-solving knowledge can be transferred to the proposed KBS. If the expert cannot bring forth the steps, processes, rationale, heuristics, etc., that he uses in a reasonably orderly manner, the chances of producing a KBS that emulates his ability are nil.

Is the domain well bounded? In other words, is the task domain limited in scope and independent of other knowledge about the world? Though an operational KBS may require large amounts of domain-specific knowledge, the existing techniques are insufficient to cope with domains that require significant amounts of general or world knowledge. Thus, in our hypothetical example in Section 2, the amount and kinds of knowledge required to adequately diagnose the cause of failures in an automobile given a set of observed conditions or symptoms may be quite large. The system need not have knowledge about automobile manufacturing, driving habits, or the sales philosophy of the agency to perform its function.

Are the intended users professionals? First, it is not likely that people in non-professional occupations confront problems of the kind for which KBS technology is appropriate. Therefore, in the appropriate areas for which there is an expert and for which the other considerations hold, the professional practitioners must have a thorough grounding in the field, understand what theory does exist, be able to converse with the expert in the jargon of the field, and confront significant problems within the domain in their daily activities. To attempt to provide a KBS that would permit a casual user or non-professional to perform at the level of the professional practitioners is not feasible today.

Do the intended users agree on an underlying "theory" and its application? It is not sufficient that there exist a coherent theory; it must be widely accepted and agreed upon by the intended users and the chosen experts. It is of little consequence that there are competing and even conflicting theories, so long as the competition and conflict are external to the intended users and experts concerned with a specific implementation. It is highly unlikely that one could successfully design and implement a KBS that could incorporate a diversity of theoretical views. A slightly simpler form of this issue is whether or not the intended users agree on who is and is not an expert, and whether there is general agreement on what is a correct result or answer.

Is a plausible or reasonable solution acceptable to the intended users? Since the power of a KBS stems from its ability to reason plausibly using incomplete or inexact information, there is no guarantee that it will always produce the "correct" result; but, under the conditions, it can be expected to produce the most reasonable or plausible result. If the intended users are unwilling to accept such results, even when the system is able to produce credible explanations, the system will not be successful. One possible way (though at some risk of wasted effort) of convincing such recalcitrant users is to bring up a limited version of the system that performs very well on what is generally agreed to be a very difficult problem set.

Will the system be required to provide unanticipated support over its lifetime? Is the domain a dynamic one such that the problems that the users must solve, though within the domain, are constantly shifting in unpredictable ways? It is expected that there will be orderly growth in the knowledge base of a KBS, but drastic shifts in emphasis may be quite difficult to accommodate. A KBS is not a general problem-solver. Even within a well-bounded domain, it may be necessary to restrict the KBS to a subset of the problems. Though it may be possibl to provide the necessary flexibility, there will be added risk in doing so.

Do problem-solving protocols exist or can they be formulated? The transfer of knowledge from the expert to the KBS can be accomplished in various ways. One way of obtaining a working model of the expert's problem-solving method is by taking protocols while he is actually solving a problem or collection of problems. Though this is not an exact science, it has been found workable in a good many cases. This is one aspect of KBS technology that is the least developed, though KBS implementers and computer scientists in other, related, areas are attempting to bring more formality and structure to the process.

Do the protocols show a reasonable consistency of reasoning--do principles surface? Does the expert approach each problem in an ad hoc manner, or does he apply a set of heuristics and reduction processes that rapidly focus his attention on the key subproblems? Unless one can extract the essence of an orderly reasoning process from the expert, the likelihood of producing a viable KBS is quite small or non-existent. Even if all of the other considerations are met, this remains one of the most critical.

<u>Are the economics right</u>? Would the users be willing to pay a human consultant what the computer solution cost is likely to be? This embraces a number of issues. The problems that are solved must be useful in that people expend a fair amount of effort trying to solve them, and the solutions are worth the effort. It should be expected that the use of the KBS will raise the level of the average practitioner significantly, either by making him more productive (less time or effort spent on each problem) or by improving the quality of his output. Another aspect of the economics is related to data gathering and recommended actions. A KBS that incorporates the proper knowledge can reduce the cost of the information-gathering process by possibly providing adequate solutions with less (or lower-quality) input, but there are limits beyond which no system or person can properly perform. A KBS can also recommend the lowest-cost, lowest-risk action to be taken, but in all cases, such recommendations should be tempered with human judgment. The role of the human practitioner is not to be subsumed by the KBS. The KBS is a tool for his use, and the final disposition of its results is his responsibility.

5.2 TECHNOLOGY CONSIDERATIONS

In what follows we address issues that relate to the design and implementation from a technological viewpoint. Some of the issues are directly related to those above, but have a different intent in that it is assumed that, if one has reached the point in the decision process for a KBS application where all of the initial considerations have been satisfied, then the concerns must turn to those of determining whether the available technology will support a design and implementation without undue risk.

Can a first-order model be constructed from protocols and/or with the help of expert(s)? The design of the KBS must be predicated on the model, extracted from the expert, of his knowledge and of his reasoning process. Unless it is obvious that there are no unforeseeable problems, it is wise to construct a first-order model that is either a hand simulation or (preferably) a rapidly constructed program to determine whether the proposed method will work. If not, it may indicate a re-examination of the design or a reappraisal of the selection. Though the model may perform well, it does not guarantee that the final KBS will do as well across the spectrum of problems it was designed for, but will indicate that the approach is reasonable.

Is there a knowledge representation that matches the "chunk size" of the expert knowledge? The design of the data structures and procedures should reflect as accurately as possible the expert's conceptualization of the problem domain in order to minimize or eliminate translation requirements and problems in discovering and removing errors and improving the system. This is not to imply th the KBS must necessarily accurately model the expert's reasoning process from a psychological viewpoint, but the expert will be a party to these processes. A mismatch will make the knowledge transfer more difficult and error prone. Having to invent a new knowledge representation technique increases the risk of failure.

<u>What are the necessary knowledge source(s) and their representation(s)?</u> The most successful KBSs to date are also the simplest ones. These represent the most stable existing technology. Therefore, one must not only choose an application area that has the general potential for success based upon domain and expert knowledge criteria, but one for which the existing technology is applicable at minimum risk. Inventing new representational techniques because the new application poses unique requirements increases the risk of failure unless the new technique is an obvious extension of one that is well known. The need for a flexible and extensible system must not introduce inconsistencies in the knowledge and conflicts in the problem-solving process, therefore, there are a variety of techniques for dealing with less than perfect knowledge or with information that is known to contain potential errors. Measures of plausibility and credibility, and of the associated Certainty Functions, must be chosen with the utmost care, for they carry a great deal of inherent prejudice about the domain.

<u>What reasoning or inference methods are needed?</u> There are several problem-solving methods (each associated with the appropriate knowledge representation) that are candidates for implementing CEs. Among the better known are heuristic search (which implies that there exists a constrainable search-space node generator), deductive inference from rules, pattern matching, means-ends analysis, and modeling and simulation (see Section 4). In highly complex systems with multiple levels of abstraction and multiple representations of knowledge, different methods may be required to cope with the problem at each level. In this case, one can view the CE as primarily an agenda mechanism and the local-level problem solvers as knowledge sources. In general the initial design for the CE should be the simplest one possible; the necessary improvements will become obvious as the system grows.

Are the knowledge representation, chunk size, and reasoning method compatible with one another? There are critical design issues related to the technique selected for representing the knowledge in the KSs and the interaction with the CE (see Section 4). It must be clearly understood that the representation technique or techniques selected for knowledge in the KB can strongly prejudice the methodology and thus the problem-solving ability of the CE. For instance, if one chooses to represent the KB's knowledge as production rules, the CE is limited to relatively simple inference-making processes. Overdesign of the CE should be avoided; it has no payoff. Where multiple heterogeneous representations of knowledge are required or chosen, the CE will be relatively complex and most likely require an evaluation and an agenda mechanism. One of the much-discussed issues in KBS technology today is that of the "chunk" size of the knowledge in the system (see Section 4). It is a function, first, of how the expert conceptualizes the theory of the domain and the problem set, and second, of the selected representation technique.

In one way or another, every CE must be imbued with the ability to plan how bes to attack the problem (or subproblem) at hand within the constraints of the applicable method that is compatible with the knowledge representation. Ordering the sequence of events, evaluating the results, and determining what needs to be done next may require only a nominal process or, in complex systems, a sophisticated agenda mechanism.

Will meta-knowledge be required? If the design of the system for the chosen domain is such that the knowledge about reasoning and control must be modular, and such knowledge must be obtained from the expert, the KBS will be quite complex, and thus the risk of failure will be increased. If, on the other hand the reasoning process and control can be incorporated in the CE, then the system will be relatively simple and easier to implement.

Need procedural knowledge be incorporated? Care must be taken not to embed knowledge in code (procedures) that is better left as KSs in the long run, even at the sacrifice of short-run efficiency. Errors of this sort in the design will either reduce the general flexibility of the system or force major modifications as the system grows. As a partial guide, all knowledge should initially be designed to be in the KB except for the absolute minimum that must be incorporated in the control structure of the CE.

Will the user be required to add knowledge to the system to solve his problem? If the problems to be solved require that the user add knowledge (albeit temporary) as contrasted to data, the knowledge-acquisition interface and associated facilities for validating the consistency of added knowledge will be more complex (as will the control mechanism in the CE) and difficult to design and implement, for they will have to serve a large population of users rather than a small set of experts. Thus, the human engineering and language interface will be more difficult to do.

Will the system support growth? A KBS must be viewed as a dynamic system in the sense that it should be designed to "grow" in various ways from its initial conception and implementation. The primary areas for improvement should be: (1) increasing its inferential capabilities both as the theory of the domain evolves and as the user's understanding increases; (2) increasing the Knowledge Base both independently of (1) as well as in conjunction with it, e.g., adding new KSs that broaden the problem set that can be accommodated; (3) improving the flexibility and human engineering aspects based on user experience and needs and the implementers' understanding of the users; (4) increasing the overall reliability of the system by refining the inferential capability and the knowledge stemming from an understanding of the failures or errors observed in use. This means that the design and implementation of the problem-solving procedures must be flexible enough to permit frequent modification, particularly in the early stages. It is most unwise to embed the problem-solving knowledge deep in the code. Thus, an appropriate selection of KSs for the KB and a proper representation are critical to success and growt

One must forego the short-term benefits of an ad hoc initial implementation in order to avoid the downstream costs of redesign and major modification. This may extract an early price in performance and immediately demonstrable results, but we believe the penalty to be worthwhile. The system design must also be flexible enough to accommodate expected changes in knowledge about specific problems as well as the problem-solving strategy that is likely to evolve over time with accumulated experience. One implication of all of this is that the implementer's job is not finished after the first success at solving a user's problem. Most KBSs to date have continued to evolve and improve through several generations before entering a stable maintenance mode. In fact it is not clear that any present-day KBS has reached this state.

5.3 ENVIRONMENTAL CONSIDERATIONS

The environmental considerations are often overlooked or given minimal attention. We believe that they should be as strongly considered as any of the above, because they contribute in their own way to the overall success or failure of a KBS development effort. Though all of the initial and technological considerations bode well for success, the operational and developmental environments, if not of the proper kind, will hinder both developers and users.

Is there an interactive system for the KBS users? Recall that as we have defined a KBS above, it must be useful to workers in the domain of application. To be most useful, a KBS should be interactive. It is conceivable that a KBS could be developed to run in a batch-processing environment, but the circumstances that would dictate such a decision are inconceivable. To quote Buchanan, "A batch system just cannot provide helpful, rapid feedback and immediate error recovery, e.g., from a simple typing error." [BUCHANAN75]. Since KBS, as presently defined, are intended as expert agents to support people, other potential applications, such as process control systems, or "intelligent" robots for remote exploration are not considered here. Thus, we believe that the basic design philosophy for a KBS should be that of a user oriented interactive system.

Is there an interactive development system? We believe that it makes sense to require that the development environment for the KBS be an interactive one, independent of whether the development environment and the user's computer environment are one and the same. An interactive development environment will speed the implementation process. One of the reasons appears to be unique to KBS. It is that the interaction with the domain expert in acquiring and validating the knowledge provided the KBS can be done much more efficiently. It is also generally true that, given the proper set of development support tools, interactive development is more effective and efficient than development in a batch environment, particularly when confronted with developing an interactive application.

Do the necessary tools exist, in particular a properly expressive programming system? The development system must support the required software tools peculiar to KBS development. In addition to the standard tools, e.g., an editor, file management, etc., an adequate programming system is necessary, one that provides the necessary expressive power in the language and commensurate debugging support. For example, the programming system should be interactive and support terminal I/O, permit incremental compilation or be interpretive, provide an evaluate function similar to the EVAL of LISP. For debugging it should support symbolic interaction, break in and tracing; for data structures it should provide lists or pointers (list structures), aggregates, such as tuples and nodes in addition to arrays, symbolic data and identifiers with associations or properties as provided in LISP; for control structures sufficient flexibility to easily process through complex structures such as recurring through a tree, and a backup mechanism. There is a tradeoff that must be confronted at the outset concerning efficiency. A programming system similar to the one just described will permit efficient and flexible development, but not necessarily provide an efficient end product. A programming system that will provide an efficient end product is likely to lack many of the features that we believe are highly desirable to permit flexible and efficient development, thus increasing the time and cost of the implementation.

Will the resultant system perform efficiently? There are two aspects to efficiency. There is efficiency in terms of the software's use of the hardware. This is a function of the programming system used, the operating environment of the computer system, and the ability of the programmers implementing the system. Certain inefficiencies can be tolerated, but the system must provide respectable response time for its users; otherwise, it will not be used. There is no automatic way to assure efficiency unless it is designed in from the beginning. The aspect of whether or not the system will be an efficient problem-solver has been indirectly addressed above.

5.4 SUMMARY

We cannot stress too strongly that the KBS must be properly organized beginning with the design. Not only must it be organized and structured so that the CE and the KB are separately identifiable entities, every element must be structured so that it is easily maintained and improved at minimum cost. Multiple KSs should be considered when designing the KB for reasons of performance and ease of adding new knowledge. The user interface must satisfy the user's perception of his needs and be based on good human engineering principles (including the choice of the proper terminal in some cases) so that it is attractive rather than repulsive. The interface must accommodate the user's views of how the interactions should proceed, e.g., the initial setting of default parameters, even though that may not be best or easiest from the implementer's view. Errors should be overlooked when possible, but at a minimum the response should be supportive and indicative of what is to be done to correct the error. Explanations of the system's behavior must be in the most acceptable and useful form for the user providing sufficient flexibility to accommodate the user's change in perception with experience. Though not intended as a primary source of debugging information, the explanations are useful both to the implementers and the experts and should be designed to incorporate their needs. Too often, conventional software implementations have gone astray because the developers lost sight of the fact that the system is to be developed for the use of and to the benefit of its users who are not likely to be computer sophisticates. The danger is lessened in developing a KBS because of the need for domain experts, but the principle should not be overlooked.

In conclusion, the decision to use KBS technology to solve a user's problem is not a simple one, nor is it without a certain amount of risk. For though much of the technology underlying KBS is well founded and understood, there remains a great deal of craft involved in completing a system, and the skill, knowledge, and even prejudice of the craftsman have an impact on the final outcome.

6. CONCLUSIONS AND RECOMMENDATIONS

The technology of knowledge-based systems has emerged from the laboratory, but it has not achieved the status of being commonly known or commonly understood as a way of implementing computer-based application systems. Systems have been developed in an intriguing spectrum of application areas, from medicine and chemistry to geology and businesses, and some general techniques have been developed that are independent of specific applications--for example, systems that model common-sense reasoning, deductive inference, and image understanding. The general level of accomplishment appears to be high enough to make it worthwhile to begin exploring other areas for immediate potential application.

There remain a number of unresolved issues that increase the difficulty and potential risk of using KBS technology in new applications. Though it is reasonably clear where KBS technology can and cannot be used, to the extent that the high-risk applications can be identified--and, if necessary, eliminated--there is no way of guaranteeing that a selected application is entirely without risk. How to select techniques for representing knowledge in a system and for constructing the control mechanisms are open issues that impact not only specific design choices, but the performance of the system as a whole. Even with a group of domain experts who are cooperative and well motivated, the methodology for transferring their knowledge to the system is, at best, ad hoc; and that transfer process is probably the most crucial process of all. This is the area in which more research is needed to discover (or invent) what amounts to a completely new technology: the acquisition, communication, and representation of <u>expertise</u>, by which we mean the ability to <u>use</u> a body of knowledge effectively in solving a particular problem.

We would not be forthright if we attempted to play down the risks that must be faced in deciding to apply KBS technology to any application that involves supplying assistance to persons involved in sensitive problem-solving activities. On the other hand, the risks can be minimized if our criteria for selecting potential applications are carefully observed. The development of a

KBS application is like any other software-development activity: it carries risks. Therefore, although we do not explicitly say so in Section 5, it is necessary that the best possible practices pertaining to software-development efforts in general be followed in developing a KBS application. Of particular importance is getting the users to participate in the requirements-specification and design processes as early as possible, and keeping those users involved until the system is turned over to them. Also, while the knowledge and skills required of technicians (or technologists) who are developing knowledge-based systems are different from the knowledge and skills required of "systems analysts" or "programmers" developing conventional computer-based systems, the fact that they may be involved in exotic applications of computer technology should not exempt them from normal management scrutiny and control. Common sense, good judgement, and good management can do much to transform what at the outset appears to be a risky endeavor into a highly successful and satisfying one.

7. ANNOTATED BIBLIOGRAPHY

The ideas in this annotated bibliography are, we believe, a cross section of the recent literature on knowledge-based systems and related research. They are intended to give the reader who is unfamiliar with the literature of the field a feeling for the kind of papers that he will encounter in exploring things on his own. Each summary or abstract is headed by the reference pointer into the general bibliography (Section 8) and the article's title as it appears in that entry.

ANDERSON76b - Rand intelligent terminal agent (RITA): design philosophy.
RITA is a production rule based system for constructing small but competent agents or other rule based processes such as TECA (see Appendix B). Users input rules in an English-like form with a restricted syntax. Rules may be executed by one (and only one) of three monitors: LHS scan with ordered rule set, LHS scan with unordered rule set, and RHS scan (goal directed, backward chaining) with implicitly unordered rule set. Several agents have been implemented to perform various user functions. This document discusses the design philosophy and some of the implementation details.

BALLARD76 - A ladder-structured tree for recognizing tumors in chest radiographs.
This paper describes a computer procedure for the detection of nodular tumors in chest radiographs. The recognition process uses a hierarchic structure in the form of a ladder-like decision tree. After locating potential nodules, the procedure classifies them into non-nodules, nodules that are not tumors, and nodules that are tumors. It accurately located tumors in five of six radiographs but missed some obscure ones in the sixth. The system contains a great deal of knowledge about tumors as they appear in radiographs, but it is not pulled together in a central knowledge base and is difficult to add to or modify, being represented as procedures.

BOBROWD77a - GUS, a frame-driven dialog system.
GUS (Genial Understander System) is intended to engage a cooperative human in an English language dialog directed toward a specific goal in a restricted domain of discourse. The authors implemented GUS in order to determine whether a modular approach for a dialog system was at all feasible and to test their notions of reasonable lines of decomposition. GUS provided a context in which to explore tools and techniques for building and integrating independent modules. The major knowledge-oriented processes and structures in GUS--the morphological analyzer, the syntax analyzer, the frame reasoner, and the language generator--were built as independent processes with well defined language or data structures to communicate across the interfaces. They were debugged separately and tied together by an asynchronous control mechanism. The frame reasoner was the focus of most of the research and development. The frame structures (which differ from Minsky's [MINSKY75] used in GUS were a first step toward a more comprehensive Knowledge Representation Language (KRL) [BOBROWD77b].

BOBROWD77b - An overview of KRL, a knowledge representation language.
This paper describes KRL, a Knowledge Representation Language designed for use in understander systems. It outlines both the general concepts which underlie the research and the details of KRL-0, an experimental implementation of some of these concepts. KRL is an attempt to integrate procedural knowledge with a broad base of declarative forms. These forms provide a variety of ways to express the procedures (for memory and reasoning) with specific pieces of knowledge, and to control the relative accessibility of different facts and descriptions. The formalism for declarative knowledge is based on structured conceptual objects with associated descriptions. These objects form a network of memory units with several different sorts of linkages, each having well-specified implications for the retrieval process. Procedures can be associated directly with the internal structure of a conceptual object. This procedural attachment allows the steps for a particular operation to be determined by characteristics of the specific entities involved.

The control structure of KRL is based on the belief that the next generation of intelligent programs will integrate data-directed and goal-directed processing by using multi-processing. It provides for a priority-ordered multi-process agenda with explicit (user-provided) strategies for scheduling and resource allocation. It provides procedure directories which operate along with process frameworks to allow procedural parameterization of the fundamental system processes for building, comparing, and retrieving memory structures. Future development of KRL will include integrating procedure definition with the descriptive formalism.

BOBROWD75b - Dimensions of representation.
In this paper the author proposes a framework for viewing the problems of representation. Each of the design issues (influenced by Moore and Newell [MOORE73]) in the framework defines a dimension of representation--a relatively independent way of looking at representation. The dimensions referred to are: (1) domain and range, (2) operational correspondence, (3) process of mapping, (4) inference, (5) access, (6) matching, and (7) self-awareness.

BOBROWD75c - Some principles of memory schemata.
The form of knowledge structures (schemata) is an amalgam of the principles of semantic networks, actors, and frames. The word schema is drawn from the psychological literature and is most commonly associated with the work on memory by Bartlett [BARTLETT32] and by Piaget. The central thesis is that one schema refers to another only through the use of a description which is dependent on the context of the original reference. These schemata are active processing elements which can be activated from higher level purposes and expectations (top-down) or from input data (bottom-up) that must be accounted for. The desire is to specify a memory structure that allows one schema retrieved from memory to suggest others that should also be retrieved to yield human-like analogical and metaphorical retrieval as a fundamental mode of operation.

BOBROWR75 - Systematic understanding: synthesis, analysis, and contingent knowledge in specialized understanding systems.
The best representation for a body of knowledge depends on how that knowledge is to be used by the program, and thus better characterization of the uses of knowledge is likely to lead to better ways of designing knowledge representations. This paper describes the SCA model, a framework for describing the structure of "conceptually efficient" understanding programs, based on a characterization of three fundamentally different ways in which knowledge is used in such programs. The SCA model can be of use both to those designing understanding systems and to those who wish to study existing systems to develop insights into different approaches to representing knowledge.

BROWN75a - Uses of artificial intelligence and advanced computer technology in education.
Advances in hardware technology will make it economically feasible for each student to have access to computational resources currently available to only a few elite users. The challenge facing educational technologists is to harness these capabilities to provide equal advances in the quality and effectiveness of CAI systems. What is needed are new instructional paradigms, not based on the belief that computation is a scare resource.

CAI systems understand their subject domain and can use their knowledge base to help a student experiment with, debug, and articulate his own ideas and reasoning strategies. These learning environments support a kind of "learning-by-doing" in which a student has freedom to solve problems in his own way, with the instructional system following and criticizing the student's line of reasoning. Examples of such systems are Goldberg's logic teaching system (1973), SOPHIE (1975), Kimball's system for teaching symbolic integration (1973), Goldstein's MYCROFT system for enriching the LOGO environment (1974), and Ruth's system for criticizing sorting programs (1974).

BROWN75b - Multiple representations of knowledge for tutorial reasoning.
This paper provides an overview of SOPHIE, an intelligent instructional system for electronic circuit debugging. Unlike previous AI-CAI systems which attempt to mimic the roles of a human teacher, SOPHIE tries to create a "reactive" environment in which the student learns by trying out ideas rather than by instruction.

SOPHIE's expertise is derived from an efficient and powerful inferencing scheme that uses multiple representations of knowledge including (1) simulation models of its microcosm, (2) procedural specialists which contain logical skills and heuristic strategies for using these models, and (3) semantic nets for encoding time-invariant factual knowledge. In this respect SOPHIE represents a departure from inferencing paradigms (of either a procedural or declarative nature) which use a uniform representation of information.

BUCHANAN76a - Computer assisted chemical reasoning.
Application programs have the immediate goal of serving the scientist. Research and educational programs have longer range goals of changing the way scientists formulate problems and how they solve them. The DENDRAL programs cut across all of these goals. Behind them are the issues involved in turning a computer system into a valued problem-solving assistant.

BUCHANAN76b - Automatic rule formation in mass spectrometry by means of the Meta-DENDRAL program.
The DENDRAL computer program uses established rules of molecular fragmentation to help chemists solve complex structural problems from mass spectral data. This paper describes a computer program, called Meta-DENDRAL, that can aid in the discovery of such rules from empirical data on known compounds. The program uses heuristic methods to search for common structural environments around those bonds that are found to fragment, and abstracts plausible fragmentation rules.

COLLINS76 - Processes in acquiring knowledge.
The objective of this paper is to develop a theory of Socratic tutoring in the form of pattern-action (or production) rules for a computer program. These pattern-action rules are being programmed on a computer system for tutoring causal knowledge and reasoning.

The production rules were derived from analysis of a variety of tutorial dialogs. The analysis accounts for the specific teaching strategies used by the tutors in the dialogs within a content-independent formalism.

The paper includes twenty-three production rules derived from the data analyzed, together with segments of the data showing the actual application of the rules in different tutorial dialogs. The strategies themselves teach students: (1) information about different cases, (2) the causal dependencies that underlie these cases, and (3) a variety of reasoning skills. These include such abilities as forming hypotheses, testing hypotheses, distinguishing between necessary and sufficient conditions, making uncertain predictions, determining the reliability or limitation of these predictions, and asking the right questions when there is not enough information to make a prediction.

DAVIS77 - Production rules as a representation for a knowledge-based consultation program.
The MYCIN system is at the forefront of two important trends in AI research: applications of AI to "real-world" problems of importance, and the incorporation in programs of large amounts of domain-specific knowledge.

This paper examines how the implementation of a knowledge-based consultation program is facilitated or inhibited by the use of production rules as a knowledge representation. The limits of applicability of this methodology are also investigated.

DUDA77 - Semantic network representations in rule-based inference systems. PROSPECTOR is a geological consultant system being developed at SRI. This system is intended to help geologists in evaluating the mineral potential of exploration sites. Authors have been influenced by MYCIN, INTERNIST (nee DIALOG), Trigoboff's work on propagating measures of uncertainty through a semantic network, and by Hendrix's partitioned semantic networks.

This paper describes a way to use semantic network representations in rule-based inference systems. This combination allows a designer to retain the desirable modularity of a rule-based approach, while permitting an explicit, structured description of the semantics of the problem domain. Since semantic nets are among the leading internal representations used in computational linguistics, their use should also simplify the development of a natural language interface between the system and its users.

ENGLEMORE77 - A knowledge-based system for the interpretation of protein x-ray crystallographic data.
The broad goal of this project is to develop intelligent computational systems to infer the three-dimensional structures of proteins from x-ray crystallographic data. The computational systems under development use both formal and judgmental knowledge from experts to select appropriate procedures and to constrain the space of plausible protein structures. The hypothesis generating and testing procedures operate upon a variety of representations of the data, and work with several different descriptions of the structure being inferred. The system consists of a number of independent but cooperating knowledge sources which propose, augment and verify a solution to the problem as it is incrementally generated.

GOLDSTEIN76 - Artificial intelligence, language and the study of knowledge. This paper studies the relationship of Artificial Intelligence to the study of language and the representation of the underlying knowledge which supports the comprehension process. It develops the view that intelligence is based on the ability to use large amounts of diverse kinds of knowledge in procedural ways, rather than on the possession of a few general and uniform principles. The paper also provides a unifying thread to a variety of recent approaches to natural language comprehension. It concludes with a brief discussion of how Artificial Intelligence may have a radical impact on education if the principles which it utilizes to explore the representation and use of knowledge are made available to the student to use in his own learning experience.

GORRY74 - Research on expert systems.
Society faces a shortage in expert systems (increased demand, knowledge explosion). Human experts are in short supply; they are not properly distributed with respect to the needs of society, and the mechanisms that society has developed for maintaining supply are now inadequate.

Computer-based expert systems could improve the supply and distribution of expert services to society. They can be mass produced, either in fact or in principle through time-sharing systems. This will alleviate the problems of non-uniform access and improper distribution. Secondly, computer-based expert systems will alleviate the problem of intellectual obsolescence due to the long time to nurture a human expert. The expert computer program is relatively insensitive to the time at which knowledge is added to it. It is possible to add knowledge to the program and instantly disseminate it. In order to build a computer-based expert system, we will have to know what constitutes expertise; therefore, the system itself in large part will represent a theory of expertise, one that can be poked and prodded with various experimental techniques. This investigation is easier with a separate store of knowledge. By contrast, the formal education of human experts is by example; it does not use a corpus of knowledge.

GORRY believes the day of computer-based experts "is a long way off", citing intrinsic and technological problems for the distribution of expertise by computer-based expert systems.

GRIGNETTI75 - An "intelligent" on-line assistant and tutor--NLS-SCHOLAR.
NLS-SCHOLAR is a system that teaches computer-naive people how to use NLS, a powerful and complex text editor. It has been designed with the belief that procedural knowledge is best learned by doing. NLS-SCHOLAR can be used as an on-line help system outside the tutorial environment. Thus the system can take the lead at first, and fade smoothly into the background as users become proficient. This capability of integrating on-line assistance and training is an extension to the traditional notion of CAI.

KAHN75 - Mechanization of temporal knowledge.
Despite the importance of understanding time in many problem-solving situations, AI research has largely ignored the temporal characteristics of problems. The application areas have been chosen to illuminate only particular aspects of a current theory of intelligence, sidestepping the "messiness" of time-related problems. This paper considers one way in which knowledge about time can be incorporated into problem-solving programs. Time knowledge can be embodied in a set of problem-solving routines which are referred to as the <u>time specialist</u>. The time specialist can then be placed in the service of a larger problem-solving program to deal with the temporal questions that arise in the latter's domain of expertise. The problem-solving program can ask the time specialist to make inferences and to answer questions concerning temporal specifications; these queries and requests are phrased in a language that is determined by the time specialist.

KLAHRD74 - Understanding understanding systems.
Makes general comments related to two papers, "How can MERLIN understand?" [MOOREJ73] and "Knowledge and its representation in a speech understanding system" [REDDY74] that describe general features of understanding systems, and deal with the relation between knowledge and cognition.

KUIPERS74 - A frame for frames: representing knowledge for recognition.
How can we represent in a computer program the kind of knowledge people

manipulate easily and effectively? One of the significant discoveries of AI has been how computationally difficult are the simple tasks of vision, language, and common sense reasoning. New frame mechanisms have been proposed by which the organization of previously accumulated knowledge can assist active perception and understanding (and recognition). The idea is that if there is too little computation time when a problem comes up, do some of the work in advance and keep the computed results available. This focuses our attention on the relationship between immediate perception, understanding, and long-term knowledge.

This paper provides an intuitive discussion of frames which can serve as a foundation for more precise statements.

KULIKOWSKI76 - Clinical consultation and the representation of disease processes: some artificial intelligence approaches.
With an ever-increasing rate of growth in medical knowledge, the need for expert consultant services grow apace. Building a flexible and sophisticated computer-based expert consultation system is a formidable task because of the complexity and heterogeneity of medical knowledge and our very limited understanding of clinical reasoning processes.

Looking back over the past five years we can detect the evolution of a new phase of computer consultation systems, marked by the building of models of patients and diseases that combine knowledge from a variety of sources with a diversity of structural representations, and the experimentation with a varied array of inferential problem-solving strategies. These systems all use AI methods in attempting to simulate the activities of an expert consultant, although they differ substantially in scope and choice of task and methodological approaches. The CASNET program has been developed to incorporate the knowledge of a network of clinical researchers in glaucoma. It involves a causal-associational representation for evolving disease processes and can be used by a variety of reasoning strategies to provide diagnostic, prognostic, and therapeutic recommendations, together with explanations and references to diverse expert opinions. This representational scheme (being implemented) generalizes the semantic description of disease processes and extends the scope of the control strategies.

MALHOTRA75b - Design criteria for a knowledge-based English language system for management: an experimental analysis.
The main result of this thesis is to show the utility and feasibility of a knowledge-based conversational English language support system for managers. Though an actual system was not implemented, Malhotra supports this feasibility contention through a detailed experimental analysis of the problem-solving behavior of 23 subjects with a "hand-simulated" perfect English language system. These experimental protocols figure prominently in the discussion of the design of a prototype management support system, one which is technologically feasible. The utility of such a support system to managers was confirmed by the test subjects in their responses to a questionnaire. The prototype management support system was designed around a simplifying assumption of an array-structured data base and a hypothetical lead battery manufacturing company faced with the problem of lower profits despite increased sales. It is a big step from a simulated prototype system operating with simple problem conditions to an implementation of a management support system over a real world problem. Nevertheless, this thesis serves as a valuable introduction to a worthwhile application area for knowledge-based support systems.

MELDMAN75 - A preliminary study in computer-aided legal analysis.
This paper describes a prototype computer system that can perform a simple kind of legal analysis, the logical derivation of a legal conclusion from a particular factual situation in the light of some body of legal doctrine. In an analysis session, the lawyer user sits at a computer terminal and enters a description of a hypothetical factual situation. A cursory reading of this dissertation does not explain the motivation of the system: education (hypothetical situation) or consultation (real situation). The system explores its

internal representations of various legal doctrines, and determines the extent to which the hypothetical facts fall with (syllogism), or next to (analogy) these doctrines. Often the system asks the user to supply additional facts that it needs in order to make these determinations. The system then informs the user of its conclusions and explains to the user the logic behind its reasoning. Whenever possible, it supports its conclusions with references to judicial decisions and to other authoritative assertions of law.

This kind of KBS requires explicit machine representations for specific factual situations that are to be analyzed. It is also necessary that the system have similar representations for more generalized situations in terms of which legal doctrines can be expressed. Finally, the legal analysis KBS must have procedures for matching the specific facts being analyzed to the more general facts contained in the doctrine.

MOOREJ73 - How can MERLIN understand?
This paper addresses the question of "How is it possible to understand?" as a series of design issues that must be met by any understanding program (the authors' claim). It illustrates the issues by means of current work in artificial intelligence and data from psychology. It then discusses MERLIN and the design decisions that characterize it and attempts to answer how the authors expect Merlin to understand.

The paper introduces Bloom's Taxonomy of Knowledge [BLOOM56] but dismisses it as not useful for their task. The design issues that are put forth to characterize understanding are: Representation, Action, Assimilation, Accommodation, Directionality, Efficiency, Error, and Depth of Understanding.

MORAN73a - The symbolic imagery hypothesis: a production system model.
This dissertation puts forth the general hypothesis that human visual imagery is symbolic in nature. Assuming that imagery operates in the context of a cognitive system that is basically a symbolic information processor, this is

the most parsimonious (and radical) symbolic imagery hypothesis from the standpoint of system architecture. This claim is limited to the particular kind of constructive visual imagery called synthetic imagery (visualization)--novel images produced by the interpretation of verbal descriptions of an unfamiliar spatial situation. Interest is in the information structure and content of visual images, for this makes a useful cognitive skill.

NILSSON74 - Artificial intelligence.
This paper is a survey of Artificial Intelligence (AI). It divides the field into four core topics (embodying the base for a science of intelligence) and eight application topics (in which research has been contributing to core ideas). The paper discusses the history, the major landmarks, and some of the controversies in each of these twelve topics. Each topic is represented by a chart citing the major references. These references are contained in an extensive bibliography. The paper concludes with a discussion of some of the criticisms of AI and with some predictions about the course of future research.

Nilsson's guess is that we still have a good deal of work to do on the problem of how to obtain, represent, coordinate, and use the extensive knowledge we now know is required. But these ideas will not come to those who merely think about the problem. They will come to those who both think and experiment with much larger systems than we have built so far. To build really larger, "knowledgeable" systems, we will have to "educate" existing programs rather than attempt the almost impossible feat of giving birth to already competent ones. It is expected that the combined man-machine strategy which has given high performance results will be expanded to allow the human expert to transfer skills and knowledge to the machine.

RUBIN75b - Hypothesis formation and evaluation in medical diagnosis.
The structure of medical knowledge necessary for diagnosis is a cause-effect net. The effects (findings) have a structure which consists of a main-concept and a set of one or more property values. When a piece of data is asserted to

the system, an attempt is made to fit it into various slots or finding specifications; several relationships between an actual finding and a finding-specification are possible: sufficient, insufficient, further, and contradictory specification. The fitting process is complicated by time considerations.

Need relationships like CAUSE, COMPLICATION, and DEVELOPS INTO between the causes (elementary hypotheses) also play an important role in the global stage of processing. Elementary hypotheses may be related to more and less specific etiologies by CHOICE SET and ISA links, respectively. Elementary hypotheses may have properties associated with them, such as EPISODIC DISEASE and a TIME-INDEX, both of which help to interpret RECURRENT-SYMPTOMS.

The processing for diagnosis proceeds as follows: first try to dispose of the new finding by attributing it to an already-established etiology. Triggering is the next step, creating active instantiations of previous inactive hypotheses. Local evaluation determines which of the active hypotheses are to be accepted, which rejected, and which deferred. Global assembling tries to combine many of the local hypotheses into a more complex one which is both coherent (the ways hypotheses can be combined are limited), and adequate (to explain all the data). Heuristics at the various processing stages of diagnosis serve to reduce the number of concurrently active hypotheses.

There is no implementation of the theory.

RYCHENER75 - The Studnt production system, a study of encoding knowledge in production systems.
This paper is concerned with Studnt, a production system implementation of the STUDENT program of Bobrow (1964). The approach is to make explicit and analyze the knowledge embodied in STUDENT, and to measure the degree to which that knowledge is understood by STUDENT; then determine what parts of the knowledge represent <u>methods</u>, what parts contribute <u>intelligence</u>, and so on.

An important motivation behind the analysis of STUDENT is to explore the properties of production systems (PSs) as an AI language. A PS program specifies its behavior in terms of condition-action rules. The conditions all refer to a common working memory which is the complete dynamic knowledge state of the program, and actions are simply changes to that knowledge state. In practice, the numbers of conditions and actions within a production are both in the range of half a dozen to a dozen. There are no control primitives as such, but rather control is achieved through explicit elements of the working memory. Features of this abstract formulation: (1) uniformity and explicitness of representation of knowledge; (2) flexibility and intelligence in the sense of doing a significant amount of condition-testing for each small sequence of actions; (3) flexibility also in the sense of being able to respond to unexpected items in the knowledge state; (4) modularity of knowledge organization, following from the way knowledge is encoded in small, independent units. In addition to these attractive properties, there is evidence that a PS-like organization is prominent in human cognition [NEWELL72a].

Studnt is designed to do only the translation from English-subset expressions into algebraic equations, which is the most interesting segment of STUDENT from the view of problem solving and natural language processing. Given an algebra word problem, Studnt outputs: a set of equations; the set of variables in those equations as represented by the input text; and a set of variables to be solved for. Studnt is implemented in Psnlst (PS analyst), a PS language specifically designed for AI applications.

SHORTLIFFE75b - Computer-based consultations in clinical therapeutics: explanation and rule acquisition capabilities of the MYCIN system.
This report describes progress in the development of an interactive computer program, MYCIN, that uses the clinical decision criteria of experts to advise physicians who request advice regarding selection of appropriate antimicrobial therapy for hospital patients with bacterial infections. Since patients with infectious diseases often require therapy before complete information about the

organism becomes available, infectious disease experts have identified clinical and historical criteria that aid in the early selection of antimicrobial therapy. MYCIN gives advice in this area by means of three subsystems: (1) A Consultation System that uses information provided by the physician, together with its own knowledge base, to choose an appropriate drug or combination of drugs; (2) An Explanation System that understands simple English questions and answers them in order to justify its decisions or instruct the user; and (3) A Rule Acquisition System that acquires decision criteria during interactions with an expert and codes them for use during future consultation sessions. A variety of human engineering capabilities have been included to heighten the program's acceptability to the physicians who will use it. Early experience indicates that a sample knowledge base of 200 decision criteria can be used by MYCIN to give appropriate advice for many patients with bacteremia. The system will be made available for evaluation in the clinical setting after its reliability has been shown to approach that of infectious disease experts.

SHIDHARAN73 - A heuristic program to discover syntheses for complex organic molecules.
The challenge of this work arises from the complexity of the task of organic chemical synthesis, the large base of scientific knowledge and vocabulary required, and the abstruse rules of reasoning employed by experts.

Synthesis involves (1) the choice of a molecule to be synthesized; (2) the formulation and specification of a plan for synthesis, involving a valid reaction pathway leading from readily available compounds to the target compound; (3) the selection of specific steps of reaction and their temporal ordering for execution; (4) the experimental execution of the synthesis; and (5) the redesign of syntheses, if necessary, depending upon the results. Step (2) above, formal synthesis, is the only concern of this paper.

The program takes as input a canonical representation (Wiswesser linear name) of the target compound together with a list of conditions that must govern the solution of the problem. The program has at its disposal a list of compounds, the "shelf library", that can be assumed available with some indication of cost and availability. The program uses a reaction library containing generalized procedures for the synthesis of functional groups of compounds. The output is a set of proposed synthesis procedures; each proposed synthesis is to be a valid reaction pathway from the available compounds, where each step is annotated with estimated yields, by-product predictions, and target molecule separation procedures.

The more challenging aspect of the problem is constructing a rational basis for the reasoning process involved in designing organic chemical syntheses. Syntheses are not brought forth in a flash of understanding but are developed one step at a time. The development of these steps can be learned. Authors have watched and interacted with a chemist (W. Fowler, one of the authors) developing syntheses in an effort to isolate useful components of expert's problem-solving activity. Some of these techniques are incorporated as heuristics within the synthesis search algorithm.

STANSFIELD76 - Wumpus advisor 1. A first implementation of a program that tutors logical and probabilistic reasoning skills.
The Wumpus advisor program offers advice to a player involved in choosing the best move in a game for which competence in dealing with incomplete and uncertain knowledge is required. The design and implementation of the advisor explores a new paradigm in Computer Assisted Instruction, in which the performance of computer-based tutors is greatly improved through the application of artificial intelligence techniques. This report describes the design of the Advisor and outlines directions for further work. Experience with the tutor is informal and psychological experimentation remains to be done.

WATERMAN74 - Adaptive production systems.
This paper presents recent results in the design and use of adaptive or self-modifying production systems (PSs). The PSs are written in PAS-II and each is

represented as a set of ordered production rules. The control cycle consists of selecting one rule from the set and executing its actions. The first rule (from top to bottom) whose conditions match the working memory is the one selected. After the actions associated with the selected rule are executed the cycle starts again, from the top. This process continues until no conditions match, or until an explicit stop is executed.

An adaptive PS is defined to be one which, through its actions, can modify its own production rules. There are three principal ways such modifications can take place: (1) by adding new rules, (2) by deleting old rules, and (3) by changing existing rules. The adaptive PSs (APSs) of this paper use just one of these three modification techniques: addition of new rules. In summary, the APS not only contains actions which can modify the contents of working memory but also actions which can add new rules to the system.

WEISS60 - Knowledge: a growth process.
Our knowledge grows the way a living body does. Author's metaphor: Scientific knowledge grows like an organic tree, not as a compilation of collector's items. Facts, observations, discoveries, as items of information, are but the nutrients on which the tree of knowledge feeds, and not until they have been thoroughly absorbed and assimilated, have they truly enlarged the body of knowledge.

The main steps of the growth process are diagrammed in the paper. The model is abridged and oversimplified. However, it illustrates the essence of the growth process, which is that in its growth an organism never adopts foreign matter outright but reorganizes and assimilates it to fit its own peculiar pattern. Organic growth is by assimilation, not accretion. It is appropriate to think of the etymology of "assimilate" in connection with frames.

WILKS74 - Natural language understanding systems within the artificial intelligence paradigm, a survey and some comparisons.
This paper gives a survey of some important natural language understanding systems (SHRDLU and second generation systems: Charniak's children stories, PARRY, Simmons' semantic network, Schank's conceptual dependencies, Wilks' templates) focusing on the problems of word sense and pronoun reference ambiguities. Also mentions, in passing, some general background issues--failure of generative paradigm of transformational grammarians and their successors the generative semanticists. This generative paradigm of understanding natural language has been superseded by the AI paradigm.

There is no agreement over the content of the AI paradigm. Two viewpoints of "understanding" as applied to the computer: empirical (ability to sustain some dialog long enough and sensibly enough to pass Turing's test) and epistemologica ("methods and representations of knowledge by which the performance is achieved must be of the right formal sort"). What is the most appropriate form of an inference system: deductive inference or some other sort of inference closer to common sense reasoning?

WINOGRAD75 - Frame representations and the declarative/procedural controversy.
The first half of this paper examines the essential features of the opposing viewpoints of knowledge representation and provides some criteria for evaluating ideas for representation. The second half contains a rough sketch of a particular version of a frame representation (suited for understanding natural language), and suggests the ways in which it can deal with the issues raised.

The proceduralists assert that our knowledge is primarily "knowing how". The declarativists see intelligence as resting on two bases: a general set of procedures for manipulating facts of all sorts, and a set of specific facts describing particular knowledge domains. From a strictly formal view there is no distinction between the positions. We can think of the interpreter or the hardware as the only program in a computer system, and everything else as data; or we can view everything as a program--a fact is a simple program which accepts input questions such as "Are you true?" and answers "true" or "false". We must go beyond these labels to see what can be gained by looking at a piece of knowledge from one viewpoint or the other.

8. BIBLIOGRAPHY

ABELSON73
Abelson, R. P. The structure of belief systems, in Computer models of thought and language, R. C. Schank and K. M. Colby (eds.), pp. 287-339. San Francisco: Freeman, 1973.

ANDERSON76a
Anderson, R. H.; Gallegos, M.; Gillogly, J. J.; Greenberg, R. B.; and Villaneuva R. RITA reference manual. Report R-1808-ARPA. Santa Monica CA: The Rand Corporation, 1976.

ANDERSON76b
Anderson, R. H., and Gillogly, J. J. Rand intelligent terminal agent (RITA): design philosophy. Report R-1809-ARPA. Santa Monica CA: The Rand Corporation, 1976.

ANDERSON72
Anderson, R. H. The linguistic approach to pattern recognition as a basis for adaptive program behavior. Proc. tenth Allerton conference on circuit and system theory, 1972.

BALLARD76
Ballard, D. H., and Sklansky, J. A ladder-structured decision tree for recognizing tumors in chest radiographs. IEEE transactions on computers, 1976.

BARNETT75a
Barnett, J. A. Module linkage and communication in large systems, in Speech recognition, D. R. Reddy (ed.), pp. 500-520. New York NY: Academic Press, 1975.

BARNETT75b
Barnett, J. A. A phonological rules system. Technical Memorandum TM-5478/000/00. Santa Monica CA: System Development Corporation, 1975.

BARNETT74
Barnett, J. A., and Pintar, D. L. CRISP: a programming language and system. Technical Memorandum TM-5455/000/00. Santa Monica CA: System Development Corporation, 1974.

BARTLETT32
Bartlett, C. F. Remembering: a study in experimental and social psychology. Cambridge, England: Cambridge University Press, 1932.

BERNSTEIN76
Bernstein, M. I. Interactive systems research: final report to the director, Advanced Research Projects Agency, for the period 16 September 1975 to 16 September 1976. Technical Memorandum TM-5243/006/00. Santa Monica CA: System Development Corporation, 1976.

BLEDSOE73
Bledsoe, W. W., and Bruell, P. A man-machine theorem proving system. Proc. of the third international joint conference on artificial intelligence. Menlo Park CA: Stanford Research Institute Publications, 1973.

BLOOM56
Bloom, B. S. (ed.). Taxonomy of educational objectives. New York NY: McKay, 1956.

BOBROWD77a

Bobrow, D.G.; Kaplan, R.M.; Kay, M.; Norman, D.A.; Thompson, H.; and Winograd, T GUS, a frame-driven dialog system. Palo Alto CA: Xerox Palo Alto Research Center, 1977.

BOBROWD77b

Bobrow, D.G., and Winograd, T. An overview of KRL, a knowledge representation language, to appear in Cognitive science 1(1), 1977.

BOBROWD75a

Bobrow, D.G., and Collins, A.M. (eds.). Representation and understanding: studies in cognitive science. New York NY Academic Press, 1975.

BOBROWD75b

Bobrow, D.G, 1975b. Dimensions of representation, in Representation and understanding: studies in cognitive science. D.G. Bobrow and A.M. Collins (eds.), pp. 1-34. New York NY: Academic Press, 1975.

BOBROWD75c

Bobrow, D.G., and Norman, D.A. Some principles of memory schemata, in Representation and understanding: studies in cognitive science, D.G. Bobrow and A.M. Collins (eds.), pp. 131-149. New York NY: Academic Press, 1975.

BOBROWD75d

Bobrow, D.G., and Raphael, B. New programming languages for artificial intelligence research. Computing surveys 6(3), pp. 155-174, 1975

BOBROWD73

Bobrow, D.G., and Wegbreight, B. A model and stack implementation of multiple environments. Comm. ACM 16(10), pp. 591-603, 1973.

BOBROWD68
Bobrow, D.G. Natural language input for a computer problem-solving system, in Semantic information processing, M. Minsky (ed.), pp. 146-226. Cambridge MA: MIT Press, 1968.

BOBROWR75
Bobrow, R.J., and Brown, J.S. Systematic understanding: synthesis, analysis, and contingent knowledge in specialized understanding systems, in Representation and understanding: studies in cognitive science. D. G. Bobrow and A.M. Collins (eds.), pp. 103-129. New York NY: Academic Press, 1975.

BROWN76
Brown, J.S.; Rubinstein, R.; and Burton, R.R. Reactive learing environment for computer-assisted electronics instruction. Report No. 3314. Cambridge MA: Bolt Beranek and Newman, Inc., 1976.

BROWN75a
Brown, J.S. Uses of artifical intelligence and advanced computer technology in education. Presented at the NSF-sponsored conference, "Ten-year Forecast for Computers and Communications: Implications for Education," 1975.

BROWN75b
Brown, J.S., and Burton, R.R. Multiple representations of knowledge for tutorial reasoning, in Representations and understanding: studies in cognitive science, D.G. Bobrow and A.M. Collins (eds.), pp. 311-349. New York NY: Academic Press, 1975.

BROWN75c
Brown, J.S.; Burton, R.R.; and Bell, A.G. SOPHIE: a step toward creating a reactive learning environment. International journal of man-machine studies 7, pp. 675-616, 1975.

BROWN75d

Brown, J.S.; Burton, R.R.; Miller, M.; DeKleer, J.; Purcell, S.; Hausman, C.; and Bobrow, R.J. Steps toward a theoretical foundation for complex knowledge-based CAI. Report No. 3135. Cambridge MA: Bolt Beranek and Newman, Inc., 1975

BROWN74

Brown, J.S., and Burton, R.R. SOPHIE - a pragmatic use of artificial intelligence in CAI. Proc. ACM 1974 annual conference, pp. 571-579, 1974.

BRUCE75

Bruce, B. 1975. Case systems for natural language. Artificial intelligence 6, pp. 372-360.

BUCHANAN76a

Buchanan, B.G., and Smith, D.H. Computer assisted chemical reasoning. Proc. III international conference on computers in chemical research, education and technology, 1976.

BUCHANAN76b

Buchanan, B.G.; Smith, D.H.; White, W.C.; Griter, R.S.; and Feigenbaum, E.A. Automatic rule formation in mass spectrometry by means of the meta-DENDRAL program. Journal of the American Chemical Society, March 1976.

BUCHANAN75

Buchanan, B.G. Applications of artificial intelligence to scientific reasoning. Proc. second USA-Japan computer conference. Tokyo, Japan, 1975.

BUCHANAN72

Buchanan, B.G.; Feigenbaum, E.A.; and Sridharan, N.S. Heuristic theory foundation: data interpretation and rule formation. Machine intelligence 7, B. Metzler and D. Michie (eds.). Edinburgh, Scotland: Edinburgh University Press, 1972.

BUCHANAN69
Buchanan, B.G.; Sutherland, G.L.; and Feigenbaum, E.A. Toward an understanding of information processes of scientific inference in the context of organic chemistry. Machine intelligence 5. B. Metzler and D. Michie (eds.). Edinburgh, Scotland: Edinburgh University Press.

BURGER77
Burger, J.F. Data base semantics in the EUFID system, to appear in Proc. second Berkeley workshop on distributed data management and computer networks, 1977.

BURGER75
Burger, J.F.; Leal, A.; and Shoshani, A. Semantic-based parsing and a natural language interface for interactive data management. Proc. 13th annual meeting of the Association for Computational Linguistics. Boston MA, 1975.

CARBONELL73
Carbonell, J.R., and Collins, A.M. Natural semantics in artificial intelligence. Proc. third international joint conference on artificial intelligence, pp. 344-351. Menlo Park CA: Stanford Research Institute Publications, 1973.

CARBONELL70
Carbonell, J.R. AI in CAI: an artificial intelligence approach to computer-aided instruction. IEEE transactions on man-machine systems, MM-II, pp. 190-202, 1970.

CARLSON74
Carlson, E.D., and Sutton, J.A. A case study of non-programmer interactive problem solving. Report RJ 1382. San Jose CA: IBM Research Laboratory, 1974.

CARNAP50

Carnap, R. Two concepts of probability, in Logical foundations of probability, pp. 19-51. Chicago IL: University of Chicago Press, 1950.

CARNIAK75

Charniak, E. A partial taxonomy of knowledge about actions. Proc. fourth international joint conference on artificial intelligence, pp. 91-98. Cambridge MA: Publications Dept., MIT AI Laboratory, 1975.

CHASE73

Chase, W.G. (ed.). Visual information processing. New York NY: Academic Press, 1973.

CHOMSKY63

Chomsky, N. Formal properties of grammars, in Handbook of mathematical psychology, vol. 2, R.D. Luce, R.R. Bush and F. Galanter (eds.), pp. 323-418. New York NY: John Wiley and Sons, 1963.

CHURCHMAN69

Churchman, C.W., and Buchanan, B.G. On the design of inductive systems: some philosophical problems. British journal for the philosophy of science, Autumn 1969.

COLLINS76

Collins, A. Processes in acquiring knowledge, in Schooling and the acquisition of knowledge, R.C. Anderson, R.J. Spiro, and W.E. Montagne (eds.). Hillsdale NJ: Erlbaum Associates, 1976.

COLLINS75a

Collins, A., and Grignetti, M.C. Intelligent CAI. Report No. 3181. Cambridge MA: Bolt Beranek and Newman, Inc., 1975.

COLLINS75b
Collins, A.; Warnock, E.H.; Aiello, N.; and Miller, M.L. Reasoning from incomplete knowledge, in Representation and understanding: studies in cognitive science, D.G. Bobrow and A.M. Collins (eds.), pp. 383-415. New York NY: Academic Press, 1975.

DAHL73
Dahl, O.-J.; Bertwistle, G.M.; Myhrhaug, B.; and Nygaard, K. SIMULA BEGIN Philadelphia PA: Auerbach, 1973.

DANIEL74
Daniel, L.M. AND-OR graphs and critical paths. Information processing 74, vol. 4, pp. 818-822. Amsterdam, Netherlands: North-Holland, 1974.

DAVIS77
Davis, R.; Buchanan, B.G.; and Shortliffe, E. Production rules as a representation for a knowledge-based consultation program. Artificial intelligence 8(1), pp. 15-45, 1977.

DAVIS76
Davis, R. Application of meta level knowledge to the construction, maintenance and use of large knowledge bases. Stanford AI Laboratory Memo AIM-283, Computer Science Dept. Report No. STAN-CS-76-552. Stanford CA: Stanford University, 1976.

DAVIS75
Davis, R., and King, J. An overview of production systems. Stanford AI Laboratory Memo AIM-271, Computer Science Dept. Report No. STAN-CS-75-524. Stanford CA: Stanford University, 1975.

DEUTCH74
Deutch, B.G. The structure of task oriented dialogs. Proc. IEEE speech symposium. Pittsburgh PA: Carnegie-Mellon University, April 1974.

DILLER73
Diller, T.C. Prosodics appearing in the SDC vocal data management dialogues. Technical Memorandum TM-5171/005/00. Santa Monica CA: System Development Corporation, 1973.

DUDA77
Duda, R.O.; Hart, P.E.; Nilsson, N.J.; and Sutherland, G.L. Semantic network representations in rule-based inference systems, to appear in Proc. workshop on pattern-directed inference systems, 1977.

DUNLAVEY75
Dunlavey, M.R. An hypothesis-driven vision system. Proc. fourth international joint conference on artificial intelligence, pp. 616-619. Cambridge MA: Publications Dept., MIT AI Laboratory, 1975.

ENGLEMORE77
Englemore, R.S., and Nii, H.P. A knowledge-based system for the interpretation of protein x-ray crystallographic data. Computer Science Dept. Report No. STAN-CS-77-589. Stanford CA: Stanford University, 1977.

ERMAN75
Erman, L.D., and Lesser, V.R. A multi-level organization for problem solving using many, diverse, cooperating sources of knowledge. Proc. fourth international joint conference on artificial intelligence, pp. 483-490. Cambridge MA: Publications Dept., MIT AI Laboratory, 1975.

EVANS64
Evans, A. An ALGOL compiler. Annual review in automatic programming, vol. 4, R.E. Goodman (ed.), pp. 87-124. London, England: Pergammon Press, 1964.

FAHLMAN75
Fahlman, S. A system for representing and using real-world knowledge. MIT AI Laboratory memo 331. Cambridge MA: Massachusetts Institute of Technology, 1975.

FARLEY74
Farley, A. VIPS: a visual imagery and perception system; the result of a protocol analysis. Computer Science Dept. doctoral thesis. Pittsburgh PA: Carnegie-Mellon University, 1974.

FEIGENBAUM71
Feigenbaum, E.A.; Buchanan, B.G.; and Lederberg, J. On generality and problem solving: a case study of the DENDRAL program. Machine intelligence 6, B. Metzler and D. Michie (eds.), pp. 165-190. Edinburgh, Scotland: Edinburgh University Press, 1971.

FEIGENBAUM63a
Feigenbaum, E.A., and Feldman, J. (eds.). Computers and thought. New York NY: McGraw-Hill, 1963.

FEIGENBAUM63b
Feigenbaum, E.A. The simulation of verbal learning behavior, in Computers and thought, E.A. Feigenbaum and J. Feldman (eds.), pp. 297-309. New York NY: McGraw-Hill, 1963.

FIKES71
Fikes, R.E., and Nilsson, N.J. STRIPS: a new approach to the application of theorem proving in problem solving. Artificial intelligence 2, pp. 189-208, 1971.

FISHER70
Fisher, D.A. Control structures for programming languages. Computer Science Dept. doctoral thesis. Pittsburgh, PA: Carnegie-Mellon University, 1970.

FLOYD61
Floyd, R.A. A descriptive language for symbol manipulation. Journal of the ACM 8, pp. 579-584, 1961.

FORGY76
Forgy, C.L. A production system monitor for parallel computers. Computer Science Dept. technical report. Pittsburgh PA: Carnegie-Mellon Univerity, 1976.

FREUD 60
Freud, S. Jokes and their relation to the unconscious. New York NY: Norton, 1960.

FREUD50
Freud, S. The interpretation of dreams. New York NY: Random House, 1950.

FULLER73
Fuller, S.H.; Gaschnig, J.C.; and Gillogly, J.J. Analysis of the alpha-beta pruning algorithm. Computer Science Dept. technical report. Pittsburgh PA: Carnegie-Mellon University, 1973.

FURUGORI74
Furugori, T. Pass the car in front of you: a simulator of cognitive processes for understanding. Proc. ACM 1974 annual conference, pp. 380-386. San Diego CA, 1974.

GALLER70

Galler, B., and Perlis, A. A view of programming languages, chapters 1 and 2. Reading MA: Addison-Wesley, 1970.

GELPERIN77

Gelperin, D. On the optimality of A*. Artificial intelligence 8(1), pp. 69-76, 1977.

GELERNTER63

Gelernter, H. Realization of a geometry-theorem proving machine, in Computers and thought, E.A. Feigenbaum and J. Feldman (eds.), pp. 134-142. New York NY: McGraw-Hill, 1963.

GOGUEN74

Goguen, J.A. Concept representation in natural and artificial languages: axioms, extensions and applications for fuzzy sets. Int. journal of man-machine studies 6(5), pp. 513-561, 1974.

GOGUEN68

Goguen, J.A. The logic of inexact concepts. Synthese 19, pp. 325-373, 1968.

GORRY74

Gorry, G.A. Research on expert systems. MAC technical memorandum 56. Cambridge MA: Massachusetts Institute of Technology, 1974.

GOLDSTEIN76

Goldstein, I., and Papert, S. Artificial intelligence, language and the study of knowledge. MIT AI Laboratory memo 337. Cambridge MA: Massachusetts Institute of Technology, 1976.

GREEN68
Green, B., and Raphael, B. The use of theorem-proving techniques in question-answering systems. Proc. ACM 23rd national conference, pp. 169-181. Princeton NJ: Brandon, 1968.

GREGG74
Gregg, L.W. (ed.). Knowledge and cognition. Potomac MD: Lawrence Erlbaum Associates, 1974.

GRIGNETTI75
Grignetti, M.C.; Hausmann, C.; and Gould, L. An "intelligent" on-line assistant and tutor--NLS-SCHOLAR. Proc. ACM 1975 national computer conference, pp. 775-781, 1975.

HARRE70
Harre , R. Probability and confirmation, in Principles of scientific thinking, pp. 157-177. Chicago IL: University of Chicago Press, 1970.

HAWKINSON75
Hawkinson, L. The representations of concepts in OWL. Proc. fourth international joint conference on artificial intelligence, pp. 107-114. Cambridge MA: Publications Dept., MIT AI Laboratory, 1975.

HAYES74
Hayes, J.R., and Simon, H.A. Understanding written problem instructions, in Knowledge and cognition, L.W. Gregg (ed.), pp. 167-200. Potomac MD: Laurence Erlbaum Associates, 1974.

HAYES-ROTH77
Hayes-Roth, F., and Lessor, V.R. Focus of attention in the HEARSAY-II speech understanding system. Computer Science Dept. technical report. Pittsburgh PA: Carnegie-Mellon University, 1977.

HAYES-ROTH76a
Hayes-Roth, F. Patterns of induction and associated knowledge acquisition algorithms. Computer Science Dept. technical report. Pittsburgh PA: Carnegie-Mellon University, 1976.

HAYES-ROTH76b
Hayes-Roth, F., and McDermott, J. Knowledge acquisition from structural descriptions. Computer Science Dept. technical report. Pittsburgh PA: Carnegie-Mellon University, 1976.

HAYES-ROTH76c
Hayes-Roth, F., and Mostow, D.J. Syntax and semantics in a distributed speech understanding system. Computer Science Dept. technical report. Pittsburgh PA: Carnegie-Mellon University, 1976.

HEDRICK76
Hedrick, C.L. Learning production systems from examples. Artificial intelligence 7 (1), 1976.

HEDRICK74
Hedrick, C.L. A computer program to learn production systems using semantic nets. Graduate School of Industrial Administration doctoral thesis. Pittsburgh PA: Carnegie-Mellon University, 1974.

HEMPEL45
Hempel, C.G. Studies in the logic of confirmation, in Aspects of scientific explanations and other essays in the philosophy of science, pp. 3-51, New York NY: the Free Press, 1965.

HEWITT75
Hewitt, C. How to use what you know. Artificial Intelligence Laboratory working paper 93. Cambridge MA: Massachusetts Institute of Technology, 1975.

HEWITT73
Hewitt, C.; Bishop, P.; and Steiger, R. A universal modular ACTOR formalism for artificial intelligence. Proc. third international joint conference on artificial intelligence, pp. 235-245. Menlo Park CA: Stanford Research Institute Publications, 1973.

HEWITT72
Hewitt, C. Description and theoretical analysis (using schemata) of PLANNER: a language for proving theorems and manipulating models in a robot. Mathematics Dept. doctoral thesis. Cambridge MA: Massachusetts Institute of Technology, 1972.

HEWITT71
Hewitt, C. Procedural embedding of knowledge in PLANNER. Proc. second international joint conference on artificial intelligence, pp. 167-182. London, England: British Computer Society, 1971.

KAHN75
Kahn, K.M. Mechanization of temporal knowledge. MAC technical report MAC-TR-155. Cambridge MA: Massachusetts Institute of Technology, 1975.

KANAL68
Kanal, L.N. Pattern recognition. Washington D.C.: Thompson, 1968.

KAY73
Kay, M. The MIND system, in Natural language processing, F. Rustin (ed.), pp. 155-188. New York NY: Algorithmics Press, 1973.

KELLOG77
Kellogg, C.; Klahr, P.; and Travis, L. Deductive methods for large data bases, to appear in Proc. fifth international joint conference on artificial intelligence.

KLAHRD74
Klahr, D. Understanding systems, in Knowledge and cognition, L.W. Gregg (ed.), pp. 295-300. Potomac MD: Laurence Erlbaum Associates, 1974.

KLAHRD73
Klahr, D. A production system for counting, subsidizing and adding, in Visual information processing, W.G. Chase (ed.), pp. 527-546. New York NY: Academic Press, 1973.

KLAHRP77
Klahr, P. Planning techniques for rule acquisition in deductive question answering, to appear in Proc. workshop on pattern-directed inference systems, 1977.

KLAHRP75
Klahr, P. The deductive pathfinder: creating derivation plans for inferential question-answering. Special paper SP 3842. Santa Monica CA: System Development Corporation, 1975.

KOCHEN74
Kochen, M. Representation and algorithms for cognitive learning. Artificial intelligence 5(3), pp. 199-216, 1974.

KUIPERS75
Kuipers, B.J. A frame for frames: representing knowledge for recognition, in Representation and understanding: studies in cognitive science, D.G. Bobrow and A.M. Collins (eds.), pp. 151-184. New York NY: Academic Press, 1975.

KULIKOWSKI76

Kulikowski, C.A.; Weiss, S.; Trigoboff, M.; and Safir, A. Clinical consultation and the representation of disease process: some AI approaches. Computer Science Dept. Report No. CBM-TR-58. New Brunswick NJ: Rutgers University, 1976.

LEDERBERG68

Lederberg, J., and Feigenbaum, E.A. Mechanization of inductive inference in organic chemistry, in Formal representation of human judgement, B. Kleinmuntz (ed.). New York NY: John Wiley & Sons, 1968.

LENAT76

Lenat, D.B. AM: an artificial intelligence approach to discovery in mathematics as heuristic search. Computer Science Dept. Report No. STAN-CS-76-570. Stanford CA: Stanford University, 1976.

LENAT75

Lenat, D.B. BEINGS: knowledge as interacting experts. Proc. fourth international joint conference on artificial intelligence, pp. 126-133. Cambridge MA: Publications Dept., MIT AI Laboratory, 1975.

LUCE65

Luce, R.D., and Suppes, P. Performance, utility, and subjective probability, in Handbook of mathematical psychology, R.D. Luce, R.R. Bush, and F. Galander (eds.), New York NY: John Wiley and Sons, 1965.

MALHOTRA76

Malhotra, A., and Sheridan, P.B. Experimental determination of design requirements for a program explanation system. Research report RC 5831. Yorktown Heights NY: IBM Research Center, 1976.

MALHOTRA75a
Malhotra, A. Knowledge-based English language system for management support: analysis of requirements. Proc. fourth international joint conference on artificial intelligence, pp. 842-847. Cambridge MA: Publications Dept., MIT AI Laboratory, 1975.

MALHOTRA75b
Malhotra, A. Design criteria for a knowledge-based English language system for management: an experimental analysis. MAC technical report MAC-TR-146. Cambridge MA: Massachusetts Institute of Technology, 1975.

MARK76
Mark, W.S. The reformulation model of expertise. Technical report LCS/TR-172. Cambridge MA: Massachusetts Institute of Technology, 1976.

MCDERMOTTD74a
McDermott, D. Assimilation of new information by a natural language understanding system. MIT AI Laboratory memo 298. Cambridge MA: Massachusetts Institute of Technology, 1974.

MCDERMOTTD74b
McDermott, D. Assimilation of new information. MIT AI Laboratory report AI-TR-291. Cambridge MA: Massachusetts Institute of Technology, 1974.

McDERMOTTJ76a
McDermott, J., and Forgy, C. Production system conflict resolution strategies. Computer Science Dept. Technical report. Pittsburgh PA: Carnegie-Mellon University, 1976.

MCDERMOTTJ76b
McDermott, J.; Newell, A.; and Moore, J. The efficiency of certain production system implementations. Computer Science Dept. technical report. Pittsburgh PA: Carnegie-Mellon University, 1976.

MELDMAN75
Meldman, J.A. A preliminary study in computer-aided legal analysis. MAC technical report MAC-TR-57. Cambridge MA: Massachusetts Institute of Technology, 1975.

MICHIE73
Michie, D., and Buchanan, B.G. Current states of the heuristic DENDRAL program for applying artificial intelligence to interpretation of mass spectra. School of artificial intelligence experimental programming report No. 32. Edinburgh, Scotland: University of Edinburgh, 1973.

MILLERPB75
Miller, P.B. Strategy selection in medical diagnosis. MAC technical report MAC-TR-153, Cambridge MA: Massachusetts Institute of Technology, 1975.

MILLERPL73
Miller, P.L. A locally-organized parser for spoken input. Doctoral thesis. Cambridge MA: Massachusetts Institute of Technology, 1973.

MINSKY75
Minsky, M. A framework for representing knowledge, in The psychology of computer vision, P. Winston (ed.). New York NY: McGraw-Hill, 1975.

MINSKY68
Minsky, M. (ed.). Semantic information processing. Cambridge MA: MIT Press, 1968.

MINSKY67
Minsky, M. Computation: finite and infinite machines. Englewood Cliffs NJ: Prentice Hall, 1967.

MINSKY63
Minsky, M. Steps toward artificial intelligence, in _Computers and thought_, E.A. Feigenbaum and J. Feldman (eds.), pp. 406-450. New York NY: McGraw-Hill, 1963.

MOOREJ73
Moore, J., and Newell, A. How can MERLIN understand?, in _Knowledge and cognition_, L.W. Gregg (ed.), pp. 201-252. Potomac MD: Lawrence Erlbaum Associates, 1973.

MOORER75
Moore, R.C. _Reasoning from incomplete knowledge in a procedural deduction system._ MIT AI Laboratory technical report AI-TR-347, Cambridge MA: Massachusetts Institute of Technology, 1975.

MORAN73a
Moran, T.P. _The symbolic imagery hypothesis: a production system model_ (vols 1 and 2). Computer Science Dept. doctoral thesis. Pittsburgh PA: Carnegie-Mellon University, 1973.

MORAN73b
Moran, T.P. The symbolic nature of visual imagery. _Proc. third international joint conference on artificial intelligence_, pp. 472-477. Menlo Park CA: Stanford Research Institute Publications, 1973.

MOSES74
Moses, J. The evolution of algebraic manipulation algorithms. _Information processing 74_: vol. 3, pp. 483-488. Amsterdam, Netherlands: North-Holland, 1974.

MYLOPOULOS75a
Mylopoulos, J.; Borgida, A.; Cohen, P.; and Roussopoulos, N. TORUS--a natural language understanding system for data management. _Proc. fourth international joint conference on artificial intelligence_, pp. 414-421. Cambridge MA: Publications Department, MIT AI Laboratory, 1975.

MYLOPOULOS75b
Mylopoulos, J.; Cohen, P.; Borgida, A.; and Sugar, L. Semantic networks and the generation of context. _Proc. fourth international joint conference on artificial intelligence_, pp. 134-142. Cambridge MA: Publications Dept., MIT AI Laboratory, 1975.

NASH-WEBBER75
Nash-Webber, N. The role of semantics in automatic speech understanding, in _Representation and understanding: studies in cognitive science_, D.G. Bobrow and A.M. Collins (eds.), pp. 351-382. New York NY: Academic Press, 1975.

NEWELL76a
Newell, A. Production systems: models of control structures, in _Visual information processing_, W.G. Chase (ed.), pp. 463-526. New York NY: Academic Press, 1976.

NEWELL76b
Newell, A., and Simon, H.A. Computer science as empirical inquiry: symbols and search. _Comm. ACM_ 19(3), 1976.

NEWELL73
Newell, A., and McDermott, J. _PSG manual_. Computer Science Dept. technical report. Pittsburgh PA: Carnegie-Mellon University, 1973.

NEWELL72a
Newell, A., and Simon, H.A. Human problem solving. Englewood Cliffs NJ: Prentice Hall, Inc., 1972.

NEWELL 72b
Newell, A. A theoretical exploration of mechanisms for coding the stimulus, in Coding procedures in human memory, A.W. Melton and E. Martin (eds.), pp. 373-434. Winston and Sons, 1972.

NII77
Nii, H.P., and Feiganbaum, E.A. Rule-based understanding of signals, to appear in Proc. workshop on pattern-directed inference systems, 1977.

NILSSON74
Nilsson, N.J. Artificial intelligence, in Information processing 74, vol. 4, pp. 778-801. Amsterdam, Netherlands: North-Holland, 1974.

NILSSON71
Nilsson, N.J. Problem solving methods in artificial intelligence. San Francisco CA: McGraw-Hill, 1971.

NILSSON69
Nilsson, N.J. Searching, problem-solving and game-playing trees for minimal cost solutions. Information processing 68, vol. 2, pp. 1556-1562. Amsterdam, Netherlands: North-Holland, 1969.

NORMAN75
Norman, D.A., and Rumelhart, D.E. Explorations in cognition. San Francisco CA: Freeman, 1975.

NORMAN72
Norman, D.A. Memory, knowledge and the answering of questions. La Jolla CA: University of California of San Diego, 1972.

NOVAK76
Novak, G.S. Computer understanding of physics problems stated in natural language. Doctoral thesis. Austin TX: University of Texas, 1976.

OSHEA73
O'Shea, T. Some experiments with an adaptive self-improving teaching system. Computer Science Dept. technical report NL-18. Austin TX: University of Texas, 1973.

POST43
Post, E.L. Formal reductions of the general combinational decision problem. American journal of mathematics, vol. 65., pp. 197-215, 1943.

POST36
Post, E.L. Finite combination processes: formulation I. Journal of symbolic logic, vol. I, pp. 103-105, 1936.

PAXTON76
Paxton, W.H. Experiments in speech understanding system control. AI Center technical note 134. Menlo Park CA: Stanford Research Institute, 1976.

POPLE, H.E., Jr.; Myers, J.; and Miller, R. DIALOG: a model of diagnostic logic for internal medicine. Proc. fourth international joint conference on artificial intelligence, pp. 848-855. Cambridge MA: Publications Dept. MIT AI Laboratory, 1975.

POPLE75b

Pople, H.E., Jr. Artificial-intelligence approaches to computer-based medical consultation. IEEE intercon conference. New York NY: IEEE, 1975.

POPLE73

Pople, H.E., Jr. On the mechanization of abductive logic. Proc. third international joint conference on artificial intelligence, pp. 147-152. Menlo Park CA: Stanford Research Institute Publications, 1973.

QUILLIAN68

Quillian, M.R. Semantic memory, in Semantic information processing, M. Minsky (ed.), pp. 227-270. Cambridge MA: MIT Press, 1968.

RABIN74

Rabin, M.O. Theoretical impediments to artificial intelligence. Information processing 74, vol. 3, pp. 615-619. Amsterdam, Netherlands: North-Holland, 1974.

RAMANI73

Ramani, S., and A. Newell. On the generation of problems. Computer Science Dept. technical report. Pittsburgh PA: Carnegie-Mellon University, 1973.

REDDY76

Reddy, D.R. Speech understanding systems: summary of results of the five-year research effort. Computer Science Dept. technical report. Pittsburgh PA: Carnegie-Mellon University, 1976.

REDDY75

Reddy, D.R. (ed.). Speech recognition. New York NY: Academic Press, 1975.

REDDY74
Reddy, D.R., and Newell, A. Knowledge and its representation in a speech understanding system, in Knowledge and cognition, L.W. Gregg (ed.), pp. 253-285. Potomac MD: Lawrence Erlbaum Associates.

REITMAN71
Reitman, J. Computer simulation applications. New York NY: Wiley-Interscience, 1971.

REIGER76
Reiger, C. One system for two tasks: a commonsense algorithm memory that solves problems and comprehends language. Technical Report 435. College Park MD: University of Maryland, 1976.

RUBIN75a
Rubin, A.D. The role of hypothesis in medical diagnosis. Proc. fourth international joint conference on artificial intelligence, pp. 856-863. Cambridge MA: Publications Dept. MIT AI Laboratory, 1975.

RUBIN75b
Rubin, A.D. Hypothesis formation and evaluation in medical diagnosis. Technical report AI-TR-316. Cambridge MA: Massachusetts Institute of Technology, 1975.

ROBINSON70
Robinson, J.A. An overview of mechanical theorem proving, in Theoretical approaches to non-numerical problem solving, R. Banerji and M. Mesarovic (eds.), pp. 2-20. New York NY: Springer-Verlag, 1970.

ROBINSON65
Robinson, J.A. A machine-oriented logic based on the resolution principle. Journal of the ACM 12(1), pp-23-41, 1965.

RUSTIN73
Rustin, R. Natural language processing. New York NY: Algorithmics Press, 1973.

RYCHENER76
Rychener, M.D. Production systems as a programming language for artificial intelligence applications. Computer Science Dept. doctoral thesis. Pittsburgh PA: Carnegie-Mellon University, 1976.

RYCHENER75
Rychener, M.D. The studnt production system, a study of encoding knowledge in production systems. Computer Science Dept. technical report. Pittsburgh PA: Carnegie-Mellon University, 1975.

SAMUEL63
Samuel, A.L. Some studies in machine learning using the game of checkers, in Computers and thought, E.A. Feigenbaum and J. Feldman (eds.), pp. 71-105. New York NY: McGraw-Hill, 1963.

SANDEWALL70
Sandewall, E. Representing natural language information in predicate calculus. Machine intelligence 6. B. Metzler and D. Michie (eds.). Edinburgh, Scotland: University of Edinburgh, 1970.

SCHANK75a
Schank, R.C. (ed.). Conceptual information processing. New York NY: American Elsevier, 1975.

SCHANK75b
Schank, R.C. The structure of episodes in memory, in Representation and understanding: studies in cognitive science, D.G. Bobrow and A.M. Collins (eds.), pp-237-272. New York NY: Academic Press, 1975.

SCHANK73
Schank, R.C., and Colby, K. (eds.). Computer models of thought and language. San Francisco CA: Freeman, 1973.

SCHANK72
Schank, R.C. Conceptual dependency: a theory of natural language understanding. Cognitive psychology 3, pp. 552-631, 1972.

SCHORRE64
Schorre, D.V. META II: a syntax-oriented compiler writing language. Proc. 19th national conference of the ACM, 1964.

SHANNON49
Shannon, C.E. The mathematical theory of communication. Urbana IL: University of Illinois Press, 1949.

SHORTLIFFE76
Shortliffe, E.H. Computer-based medical consultations: MYCIN. New York NY: American Elsevier, 1976.

SHORTLIFFE75a
Shortliffe, E.H., and Buchanan, B.G. A model of inexact reasoning in medicine. Mathematical biosciences vol. 23, pp. 351-379, 1975.

SHORTLIFFE75b
Shortliffe, E.H.; Davis, R.; Axline, S.G.; Buchanan, B.G.; Green, C.C.; and Cohen, S.N. Computer-based consultations in clinical therapeutics: evaluation and rule acquisition of the MYCIN system. Computers and biomedical research vol. 8, pp. 303 ff., 1975.

SHORTLIFFE73
Shortliffe, E.G.; Axline, S.G.; Buchanan, B.G.; Merigan, T.C.; and Cohen, S.N. An artificial intelligence program to advise physicians regarding antimicrobial therapy. Computers and biomedical research vol. 6, pp. 544-560, 1973.

SIMMONS73
Simmons, R.F. Semantic networks: their computation and use for understanding English sentences, in Computer models of thought and language, R.C. Schank and K. Colby (eds.), pp. 63-113. San Francisco CA: Freeman, 1973.

SIMON74a
Simon, H.A., and Lea, G. Problem solving and rule induction: a unified view, in Knowledge and cognition, L.W. Gregg (ed.), pp. 105-127. Potomac MD: Lawrence Erlbaum Associates.

SIMON74b
Simon, H.A., and Kadane, J.B. Optimal problem-solving search: all-or-none solutions. Computer Science Dept. technical report. Pittsburgh PA: Carnegie-Mellon University, 1974.

SIMON69
Simon, H.A. The sciences of the artificial. Cambridge MA: MIT Press, 1969.

SIROVICH72
Sirovich, F. Memory system of a problem solver generator. Computer Science Dept. report. Pittsburgh PA: Carnegie-Mellon University.

SLAGLE69
Slagle, J., and Dixon, J. Experiments with some programs that search game trees. Journal of the ACM 16(2), pp. 189-207, 1969.

SRIDHARAN74

Sridharan, N.S. A heuristic program to discover synthesis for complex organic molecules. Information processing 74 vol. 4, pp. 828-833. Amsterdam, Netherlands: North-Holland, 1974.

SRIDHARAN73

Sridharan, N.S.; Gebernten, H.; Hart, A.J.; Fowler, W.F.; and Shue, H.J. A heuristic program to discover synthesis for complex organic molecules. Computer Science Dept. technical report STAN-CS-73-370. Stanford CA: Stanford University, 1973.

STALLMAN76

Stallman, R.M., and Sussman, G.J. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. MIT AI Laboratory memo 380. Cambridge MA: Massachusetts Institute of Technology, 1976.

STANSFIELD76

Stansfield, J.L.; Carr, B.P.; and Goldstein, I.P. Wumpus advisor 1. A first implementation of a program that tutors logical and probabilistic reasoning skills. MIT AI Laboratory memo 381. Cambridge MA: Massachusetts Institute of Technology, 1976.

STEFIK77

Stefik, M.J., and Martin, N. A review of knowledge based problem solving as a basis for a genetics experiment design system. Computer Science Dept. report No. STAN-CS-n-596, Heuristic programming project memo HPP-75-5. Stanford CA: Stanford University, 1977.

SUSSMAN73

Sussman, G.J. A computational model of skill acquisition. MIT AI Laboratory report AI-TR-297. Cambridge MA: Massachusetts Institute of Technology, 1973.

SWARTOUT77
Swartout, W.R. A digitalis therapy adviser with explanations. LCS/TR-176. Cambridge MA: Massachusetts Institute of Technology, 1977.

TEITLEMAN72
Teitleman, W. Do what I mean: the programmer's assistant. Computers and automation, April 1972.

TÖRNEBOHM66
Törnebohm, H. Two measures of evidential strength, in Aspects of inductive logic, J. Hintikka and P. Suppes (eds.), pp. 81-95. Amsterdam, Netherlands: North-Holland, 1966.

TRIGOBOFF, M. Propagation of information in a semantic net. Computer Science Dept. Report No. CBM-TR-57. New Brunswick NJ: Rutgers University, 1976.

TVERSKY72
Tversky, A. Elimination by aspects: a theory of choice. Psychological review 79, pp. 281-299, 1972.

UHR74
Uhr, L. EASEY: an English-like programming language for artifical intelligence and complex information processing. Computer Science Dept. TR-233. Madison WI: University of Wisconsin, 1974.

VERE77
Vere, S.A. Relational production systems. Artificial intelligence 8(1), pp. 47-68, 1977.

WANG74
Wang, P.S. Symbolic evaluation of definite integrals by residue theory in MACSYMA. Information processing 74 vol. 4, pp. 823-827. Amsterdam, Netherlands: North-Holland, 1974.

WATERMAN75
Waterman, D.A. Serial pattern acquisition: a production system approach, Psychology Dept. CIP working paper no. 286. Pittsburgh PA: Carnegie-Mellon University, 1975.

WATERMAN74
Waterman, D.A. Adaptive production systems. Psychology Dept. CIP working paper no. 285. Pittsburgh PA: Carnegie-Mellon University, 1974.

WATERMAN70
Waterman, D.A. Generalization learning techniques for automating the learning of heuristics. Artificial intelligence 1, pp. 121-170, 1970.

WEISS60
Weiss, P. Knowledge: a growth process. Science (June 10, 1960) 131, pp. 1716-1719, 1960.

WEIZENBAUM66
Weizenbaum, J. ELIZA — a computer program for the study of natural language communication between man and machine. Comm. ACM 9(1), pp. 36-45, 1966.

WILKS75
Wilks, Y. An intelligent analyzer and understander of English. Comm. ACM 18(5), pp. 264-274, 1975.

WILKS74
Wilks, Y. Natural language understanding systems within the artificial intelligence paradigm. Stanford AI Laboratory memo AIM-237. Menlo Park CA: Stanford University, 1974.

WINOGRAD75
Winograd, T. Frame representations and the declarative/procedural controversy, in Representation and understanding: studies in cognitive science, D.G. Brobrow and A.M. Collins (eds.), pp. 188-210. New York NY: Academic Press, 1975.

WINOGRAD73
Winograd, T. A procedural model of language understanding, in Computer models of thought and language, R.C. Schank and K. Colby (eds.), pp. 152-186. San Franciso CA: Freeman, 1973.

WINOGRAD72
Winograd, T. Understanding natural language. New York NY: Academic Press, 1972.

WOODS76
Woods, W.A. Speech understanding systems: final report November 1974 – October 1976. BBN report no. 3438, vols. 1-5. Cambridge MA: Bolt Beranek and Newman, Inc., 1976.

WOODS75
Woods, W.A. What's in a link: foundations for semantic networks in Representation and understanding: studies in cognitive science, D.G. Bobrow and A.M. Collins (eds.), pp. 35-82. New York NY: Academic Press, 1975.

WOODS73
Woods, W.A. An experimental parsing system for transition network grammars, in Natural language processing, R. Rustin (ed.), pp. 111-154. New York NY: Algorithmics Press, 1973.

WOODS71
Woods, W.A. The lunar science natural language information system. BBN report no. 2265. Cambridge MA: Bolt Beranek and Newman, Inc., 1971.

WOODS70

Woods, W.A. Transition network grammars for natural language analysis. Comm. ACM 13(10), pp. 591-606, 1970.

YAKIMOVSKY76

Yakimovsky, Y., and Cunningham, P. DABI — a data base for image analysis with nondeterministic inference capability. Technical memorandum 33-773. Pasadena CA: Jet Propulsion Laboratory, California Institute of Technology, 1976.

YATES70

Yates, R.; Raphael, B.; and Hart, P. Resolution graphs. Artificial intelligence 1(4), 1970.

ZADEH75

Zadeh, L.A. The concept of a linguistic variable and its application to approximate reasoning — I. Information sciences 8(3), pp. 199-249, 1975.

ZADEH74

Zadeh, L.A. Fuzzy logic and its application to approximate reasoning. Information processing 74 vol. 3, pp. 591-594. Amsterdam, Netherlands: North-Holland, 1974.

ZADEH65

Zadeh, L.A. Fuzzy sets. Information and control 8, pp. 338-353, 1965.

1.00 KNOWLEDGE
- 1.10 Knowledge of specifics
  - 1.11 Knowledge of terminology
  - 1.12 Knowledge of specific facts
- 1.20 Knowledge of ways and means of dealing with specifics
  - 1.21 Knowledge of conventions
  - 1.22 Knowledge of trends and sequences
  - 1.23 Knowledge of classifications and categories
  - 1.24 Knowledge of criteria
  - 1.25 Knowledge of methodology
- 1.30 Knowledge of universals and abstractions in a field
  - 1.31 Knowledge of principles and generalizations
  - 1.32 Knowledge of theories and structures

2.00 COMPREHENSION
- 2.10 Translation
- 2.20 Interpretation
- 2.30 Extrapolation

3.00 APPLICATION

4.00 ANALYSIS
- 4.10 Analysis of elements
- 4.20 Analysis of relationships
- 4.30 Analysis of organizational principles

5.00 SYNTHESIS
- 5.10 Production of unique communication
- 5.20 Production of a plan, or proposed set of operations
- 5.30 Derivation of a set of abstract relations

6.00 EVALUATION
- 6.10 Judgments in terms of internal evidence
- 6.20 Judgments in terms of exernal criteria

Figure A1. Outline of the Taxonomy of Knowledge and Cognitive Skills

APPENDIX A: A TAXONOMY OF KNOWLEDGE AND COGNITIVE SKILLS

INTRODUCTION

This appendix presents, for those readers interested in the topic, a taxonomy of knowledge and cognitive skills taken from [BLOOM56]. It is a functional taxonomy of both knowledge and the related intellectual abilities and skills. The taxonomy was developed in the mid-1950's by a Committee of College and University Examiners as a taxonomy of educational objectives. There is some disagreement as to whether or not the taxonomy met its objectives, but we believe that it is appropriately illuminating for our purposes here, despite the fact that Moore and Newell found it insufficient for theirs: "the very strength of such functional decompositions (to cover without being precise) also constitutes their main disadvantage from our current view (namely, to understand what it is to understand). For there is nothing in the taxonomy that helps discern the attributes of an understanding system." [MOORE75] As Klahr observed, "Perhaps one reason that the Bloom taxonomy has hung around for almost 20 years was that it could be summarized into a one-page hierarchy of important aspects of intellectual functioning. People could respresent a vast and complex area in six principal chunks." [KLAHRD75]. That one-page summary is presented here (Figure A1) both as a guide to the taxonomy itself and to clarify what immediately follows it.

As the reader may have already discovered on his own, and as will be seen from the taxonomy itself, the concept of knowledge neither is simple, in the sense that it can be rigorously defined or bounded, nor can it be divorced from the means of acquiring or using it. The latter is equally true whether we are speaking of humans or computerized knowledge-based systems technology.

Why and how is an understanding of knowledge in general relevant to understanding KBS technology? First and foremost, the central issues are the Knowledge Sources (KSs) of a KBS and how those KSs are used in accomplishing the KBS's function. Imagine, for a moment, a KBS that is to act as a travel agent for aiding a user in planning business trips. To begin with, such a system must have a fair amount of specific knowledge (category 1.10) about schedules for airlines, trains, ships, and other modes of transportation, and knowledge of the terms used when referencing them. It must have knowledge of how to manipulate these data on schedules (category 1.20) in order to determine by what means the traveller can get from the present or proposed location to a desired destination. The system must have knowledge about abstract concepts (category 1.30), such as the fastest way, the least expensive way, or the most convenient way (the latter implies that it can determine the user's prejudice about such things).

But such a system must, above all, have the facility for using the KSs in a goal-directed manner in order to satisfy the demands of its users. Thus, in some sense it must "comprehend" (category 2.00) the knowledge that is available and the domain within which the knowledge is applicable. To accomplish its purposes it must translate the user's request into a sequence of internal processes and must interpret the user's desires in the light of the specific and general knowledge it has available, and ultimately apply (category 3.00) its knowledge to produce the desired result. Since the domain in which such a system would function needs to incorporate any of the higher-level categories of abilities and skills, it goes without saying that it must deal with its users in their terms and produce alternatives upon request with little or no additional data from them.

This is only an example; as yet, such a system does not exist (though one is being attempted [BOBROWD77a]. It is illustrative of how one may functionally describe a particular KBS in terms of the taxonomy, but it does not resolve the problem of how to characterize knowledge-based systems, in general.

Much of the latter portions of the taxonomy are not of direct relevance to the issues we are addressing, though they are relevant to the broader field of artificial intelligence (AI) from whence comes the underlying knowledge and technology for knowledge-based systems. Though it is highly desirable to be able to produce knowledge-based systems that incorporate many of the higher-level cognitive skills embodied in the categories of Analysis, Synthesis, and even Evaluation, the current state of the art in producing computer-based systems that can perform these functions is at best primitive.

The Bloom taxonomy follows, but not in its entirety; we have deleted those portions of the explanatory material that did not appear relevant or necessary, but have included considerably more than the condensed version the authors provided in their appendix. Further, we have changed some wording and added remarks of our own (identified as such by inclusion in braces { }) to clarify or make the particular topic more specific to knowledge-based systems. The reader should keep in mind that this is a taxonomy of educational objectives, not a purely abstract taxonomy of knowledge and the associated intellectual skills.*

Taxonomy of the Educational Objectives for the Cognitive Domain

KNOWLEDGE

1.0 KNOWLEDGE

Knowledge as defined here involves the recall of specifics and universals, the recall of methods and processes, or the recall of a pattern, structure, or setting. For evaluation purposes, recalling involves little more than locating the appropriate material. Although some alteration of the material may be required, this is a relatively minor part of the task. The knowledge objectives emphasize most the process of recall. The process of relating is also involved in that a knowledge test situation requires the organization and reorganization of a problem such that it will furnish the appropriate signals and cues for the knowledge the system possesses. For example, if one thinks of a knowledge

*Some readers in 1977 may find the exclusive use of the masculine pronoun in this material unsuitable or offensive (as in the constant reference to the "knower" as "he"). We have not attempted to neutralize references to moderate them in accordance with contemporary sensibilities, simply because our editor has not had the time to help us to do so. We trust that our readers will acquiesce in this tolerance of the sensibilities of what amounts, in this respect, to an earlier age. We must go on record as <u>not</u> subscribing to a view of knowledge as a masculine trait.

file, the problem in a knowledge test situation is that of finding in the problem or task the appropriate signals, cues, and clues which will most effectively bring out whatever knowledge is stored.

In the classification of knowledge, the arrangement is from the relatively concrete to the more complex and abstract. Thus the knowledge of specifics refers to types of information or knowledge which can be isolated and retrieved separately, while the knowledge of universals and abstractions emphasizes the interrelations and patterns in which information can be organized and structured.

While it is recognized that knowledge is involved in the more complex major categories of the taxonomy (2.00 to 6.00), the knowledge category differs from others in that retrieving is the major process involved here, while in the other categories retrieving is only one part of a much more complex process of relating, judging, and reorganizing.

1.10 KNOWLEDGE OF SPECIFICS

Knowledge of specifics is the recall of specific and isolable bits of information. The emphasis is on symbols that have concrete referents and are, for the most part, at a relatively low level of abstraction. There is a tremendous wealth of these specifics and there must always be some selection. For classification purposes, the specifics may be distinguished from the more complex classes of knowledge by virtue of their very specificity, that is, they can be isolated as elements or bits which have some meaning or value by themselves. This material may be thought of as the elements from which more complex and abstract forms of knowledge are built. {In information processing we often, though not always, refer to this kind of information as data.}

1.11 Knowledge of Terminology

Knowledge of terminology is the knowledge of the referents for specific symbols (verbal and non-verbal). This may include knowledge of the most generally accepted symbol referent, knowledge of the variety of symbols which may be used for a single referent, or knowledge of the referent most appropriate to a given use of a symbol.

Probably the most basic type of knowledge in a particular field is its terminology. In any attempt by workers to communicate with others about phenomena within the field, they find it necessary to make use of some of the special symbols and terms they have devised. In many cases it is impossible for them to discuss problems in their field without making use of some of the essential terms of that field. Quite literally, they are unable to even think about many of the phenomena in the field unless they make use of these terms and symbols. Some illustrative examples are:

- To define technical terms by giving their attributes, properties, or relations.
- To be familiar with a large number of words in their common range of meanings.
- To acquire an understanding of the vocabulary used in quantitative thinking.

1.12 Knowledge of Specific Facts

Knowledge of specific facts is the knowledge of dates, events, persons, places, sources of information, etc. This may include very precise and specific information, such as the exact date of an event or the exact magnitude of a phenomenon. It may also include approximate information, such as a time period in which an event occurred or the general order of magnitude of a phenomenon. Knowledge of specific facts refers to those facts which can be isolated as separate, discrete elements in contrast to those which can only be known in a larger context.

In every field there are a large number of dates, events, persons, places, etc., known by the specialist, which represent findings or knowledge about the field. These can be distinguished from the terminology in that the terminology generally represents the conventions or agreements within a field, while the facts are more likely to represent the findings which can be tested by other means than determining the unanimity of workers in the field or the agreements they have made for purposes of communication. Such specific facts also represent basic elements which the specialist must use in presenting communications about the field and in thinking about specific problems or topics in the field. It should also be recognized that this classification includes knowledge about particular books, writings, and sources of information on specific topics and problems. Thus, knowledge of a specific fact as well as knowledge of the source which deals with the fact are both classifiable under this heading.

Some illustrative examples are:

- The recall of major facts about particular systems.
- Recall and recognition of what is characteristic of particular periods.
- Knowledge of physical and chemical properties of common elements and their compounds.
- Knowledge of reliable sources of information.

1.20 KNOWLEDGE OF WAYS AND MEANS OF DEALING WITH SPECIFICS

Knowledge in this category is of the ways of organizing, studying, judging, and criticizing ideas and phenomena. This includes the methods of inquiry, chronological sequences, and the standards of judgment within a field as well as the patterns of organization through which the areas of the fields themselves are determined and internally organized.

At a somewhat more abstract level than the specifics are the methods of organizing and dealing with them. Each subject field has a body of techniques, criteria, classifications, and forms which are used to discover specifics as well as to deal with them once they are discovered. These differ from the specifics in that they form the connecting links between specifics, the operations necessary to establish or deal with specifics, and the criteria by which specifics are judged and evaluated. It must be made clear that this class of behaviors is only a very limited one as used here. It does not involve use of the ways and means so much as it does a knowledge of their existence and possible use. The actual skills and abilities which involve their use are described in the 2.00 to 6.00 classes of the taxonomy.

Although it will frequently be found difficult to distinguish knowledge of ways and means from knowledge of specifics for purposes of classification, several characteristics will be useful in making these distinctions. Ways and means will refer to processes rather than products. They will indicate operations rather than the results of operations. They will include knowledge which is largely the result of agreement and convenience rather than the knowledge which is more directly a matter of observation, experimentation, and discovery. They will more commonly be reflections of how workers in the field think and attack problems rather than the results of such thought or problem solving.

Many of the ways and means may represent relatively arbitrary and even artificial forms which are meaningful only to the specialist who recognizes their value as tools and techniques in his work.

1.21 Knowledge of Conventions

Knowledge of conventions is of the characteristic ways of treating and presenting ideas and phenomena. These are the usages, styles, and practices which are employed in a field because the workers find they suit their purposes or because they appear to suit the phenomena with which they deal. This may include such varied phenomena as conventional symbols used in map making and dictionaries, rules of social behavior, and rules, styles, or practices commonly employed in scholarly fields.

There are many conventions and rules which the workers in a field find extremely useful in dealing with the phenomena of a field. Although many such conventions may be retained because of habit and tradition rather than usefulness, at some point in time they were found to be especially significant in giving some structure to the phenomena. Generally these conventions will have an arbitrary existence since they were developed or retained because of general agreement or concurrence of workers in the field. They are usually true only as a matter of definition and practice rather than as a result of discovery or observation.

In some fields these conventions make up the largest proportion of the knowledge of the field. {Part of the human engineering that makes a knowledge-based system acceptable to its users is the incorporation of these kinds of conventions.}

Some examples of knowledge of conventions are:

- Knowledge of acceptable forms of language.
- Knowledge of the ways in which symbols are used to indicate the correct pronunciation of words.
- Knowledge of the standard representational devices and symbols in maps and charts.
- A knowledge of the rules of punctuation.

1.22 Knowledge of Trends and Sequences

Knowledge of trends and sequences includes the processes, directions, and movements of phenomena with respect to time. It includes trends as attempts to point up the interrelationship among a number of specific events which are separated by time. It also includes representations of processes which may involve time as well as causal interrelations of a series of specific events. Many of the trends and sequences are difficult to communicate because they involve highly dynamic actions, processes, and movements which are not fully represented by static verbal, graphic, or symbolic forms. {Few knowledge-based systems incorporate this kind of knowledge, and then only in the simplest form.}

Knowledge of trends and sequences would include, for example:

- The basic knowledge of the evolutionary development of man.
- Knowledge of how Greek philosophy has affected the contemporary world.
- Knowledge of trends in computer architecture during the last fifteen years.

1.23 Knowledge of Classifications and Categories

Knowledge of the classes, sets, divisions, and arrangements which are regarded as fundamental or useful for a given subject field, purpose, argument, or problem are included here. As a subject field, problem, or topic becomes well developed, individuals working on it find it useful to develop classifications and categories which help to structure and systemize the phenomena. Under the present heading is included only knowledge of the classifications and categories, while the application of these to new problems is dealt with in other parts of the taxonomy. {Most knowledge-based systems incorporate knowledge of this kind.} Some examples are:

- Knowledge of the classifications for organic chemicals and compounds.
- Knowledge of the characteristics of organisms that can cause infectious diseases.

1.24 Knowledge of Criteria

Knowledge of the criteria by which facts, principles, opinions, and conduct are tested or judged. Here again is systemization which is found useful by workers attacking the problems of a field. The utilization of the criteria in actual problem situations will be found in 6.00 - Evaluation. The criteria will vary markedly from field to field. They are likely to appear complex and abstract and to acquire meaning only as they are related to concrete situations and problems. {This kind of knowledge is central to plausible reasoning.} This classification of knowledge includes:

- Knowledge of the criteria for judgment appropriate to work in a particular field.
- Knowledge of the criteria by which valid sources of information in computer sciences can be recognized.

1.25 Knowledge of Methodology

Here is included knowledge of the methods of inquiry, techniques, and procedures employed in a particular subject field, as well as those employed in investigating particular problems and phenomena. Here, again, the emphasis is on knowledge of the methods rather than on ability to use the methods in the ways defined by categories 3.00 to 6.00. However, one is frequently required to know about methods and techniques and to know the ways in which they have been used. Such knowledge is most nearly of an historical or encyclopedic type. This knowledge, although simpler and perhaps less functional than the ability to actually employ the methods and techniques, is an important prelude to such use. {It is not clear whether or not this kind of knowledge is in current systems. A somewhat simpler form than implied here is represented.} This category includes:

- Knowing the methods of attack relevant to kinds of problems of concern to the social sciences.
- Knowledge of the scientific methods for evaluating a new medical treatment.

1.30 KNOWLEDGE OF THE UNIVERSALS AND ABSTRACTIONS IN A FIELD

Knowledge of the major ideas, schemes, and patterns by which phenomena and ideas are organized are covered in this category. These are the large structures, theories, and generalizations which dominate a subject field or which are quite generally used in studying phenomena or solving problems. These are at the highest levels of abstractions and complexity.

These concepts bring together a large number of specific facts and events, describe the processes and interrelations among these specifics, and thus enable the worker to organize the whole in a parsimonious form.

These tend to be very broad ideas and plans which are rather difficult to comprehend. {They are also the most difficult to incorporate in a computer-based system. The problem of proper representation that will permit effective and efficient problem solving is most important.}

1.31 Knowledge of Principles and Generalizations

Knowledge of principles and generalizations is of the particular abstractions which summarize observations of phenomena. These are the abstractions which are of greatest value in explaining, describing, predicting, or determining the most appropriate and relevant action or direction to be taken. The actual application of these abstractions in problem situations is included in 3.00 - Application. {This kind of knowledge is mandatory in any system that is intended to solve useful problems, even though restricted to a narrow domain.}

Within this category, the following are representative:

- Knowledge of propositions, of fundamental logical principles, of propositional functions and quantifiers, and of sets.
- Knowledge of biological laws of reproduction and heredity.
- Understanding of some of the principal elements in the heritage of Western civilization.
- Understanding of such basic biological principles as cell theory, osmosis, and photosynthesis.
- Knowledge of the principles of federalism.

1.32 Knowledge of Theories and Structures

Knowledge of theories and structures is the body of principles and generalizations, together with their interrelations, which present a clear, rounded, and systematic view of a complex phenomenon, problem, or field. They can be used to show the interrelation and organization of a great range of specifics. This category differs from 1.31 in that here the emphasis is on a body of principles and generalizations which are interrelated to form a theory of structure, while the principles and generalizations in 1.31 are treated as particulars which need not be related to each other. {It is this category of knowledge that distinguishes systems that "understand" from those that "merely" solve problems. Most knowledge-based systems fall in the latter category.}

Examples of knowledge in this category are:

- Knowledge of the philosophic bases for particular judgments.
- Understanding of the interrelations of chemical principles and theories.

- To understand the structure and organization of Congress.
- Knowledge of a relatively complete formulation of the theory of evolution.

INTELLECTUAL ABILITIES AND SKILLS

Abilities and skills refer to organized modes of operation and generalized techniques for dealing with materials and problems. The materials and problems may be of such a nature that little or no specialized and technical information is required. {This is not a likely condition with respect to knowledge-based systems of the near future.} Such information as is required can be assumed to be part of the general fund of knowledge {of which knowledge-based systems have little or none.} Other problems may require specialized and technical information at a rather high level such that specific knowledge and skill in dealing with the problem and materials are required. The abilities and skills categories emphasize the processes of organizing and reorganizing material to achieve a particular purpose.

2.00 COMPREHENSION

This represents the lowest level of understanding. It refers to a type of understanding or apprehension such that the individual knows what is being communicated and can make use of the material or idea being communicated without necessarily relating it to other material or seeing its fullest implications. {This describes what one should generally expect as the maximum level of achievement from present-day KBS technology. Though, according to this taxonomy, Application (see category 3.00) ranks higher on the scale of abstraction, we do not mean to imply that KB systems are not capable of being applied, but rather that such systems are not likely to exhibit the higher forms of abilities and skills put forth here in the near future.}

Three types of comprehension are considered here. The first is translation, which means that a communication can be put into other languages, into other terms, or into another form of communication. It will usually involve the giving of meaning to the various parts of a communication, taken in isolation, although such meanings may in part be determined by the context in which the ideas appear. The second type of behavior is interpretation, which involves dealing with a communication as a configuration of ideas whose comprehension may require a reordering of the ideas into a new configuration. This also includes thinking about the relative importance of the ideas, their interrelationships, and their relevance to generalizations implied or described in the original communication. The third type of behavior to be considered under comprehension is extrapolation. It includes the making of estimates or predictions based on an understanding of the trends, tendencies, or conditions described in the communication. It may also involve the making of inferences with respect to implications, consequences, corollaries, and effects which are in accordance with the conditions described in the communication. Extrapolation may include judgments with respect to a universe where the communication characterizes a sample, or conversely with respect to a sample where the communication describes a universe. For the purpose of classification, interpolation may be regarded as a type of extrapolation in that judgments with respect to intervals within a sequence of data presented in a communication are similar to judgments going beyond the data in the usual sense of extrapolation.

2.10 TRANSLATION

Translation behavior occupies a transitional position between the behaviors classified under the category of knowledge and types of behavior described under the headings of interpretation, extrapolation, analysis, synthesis, application, and evaluation. It will usually be found that competence in translation is dependent on the possession of the requisite or relevant knowledge. Translation is comprehension, as is evidenced by the care and accuracy with which the communication is paraphrased or rendered from one language or form of communication into another. Translation is judged on the basis of faithfulness or accuracy, that is, on the extent to which the material in the original communication is preserved although the form of the communication has been altered. {MYCIN's knowledge-acquisition subsystem demonstrates this facility in the way it translates and rephrases an expert's input of a rule in order to verify that the intent has been properly preserved.}

Examples:

<u>Translation from one level of abstraction to another</u>

- The ability to translate a problem given in technical or abstract phraseology into concrete or less abstract phraseology--"state the problem in your own words."
- The ability to translate a lengthy part of a communication into briefer or more abstract terms.
- The ability to translate an abstraction, such as some general principle, by giving an illustration or sample.

<u>Translation from symbolic form to another form, or vice versa</u>

- The ability to translate relationships expressed in symbolic form, including illustrations, maps, tables, diagrams, graphs, and mathematical and other formulas to verbal form and vice versa.
- Given geometric concepts in verbal terms, the ability to translate into visual or spatial terms.
- The ability to prepare graphical representations of physical phenomena or of observed or recorded data.

<u>Translation from one verbal form to another</u>

- The ability to translate non-literal statements (metaphor, symbolism, irony, exaggeration) to ordinary English.

2.20 INTERPRETATION

In order to interpret a communication, one must first be able to translate each of the major parts of it--this includes not only the words and phrases, but also the various representational devices used. He must then be able to go beyond this part-for-part rendering of the communication to comprehend the relationships between its various parts, to reorder, or to rearrange it so as to secure some total view of what the communication contains and to relate it to his own fund of experiences and ideas. Interpretation also includes competence in recognizing the essentials and differentiating them from the less essential portions or from the relatively irrelevant aspects of the communication. This requires some facility in abstracting generalizations from a set of particulars as well as in weighing and assessing the relative emphasis to be given the different elements in the communication. In these respects, interpretation becomes synonymous with analysis and has characteristics in common with evaluation.

Examples:

- The ability to grasp the thought of a work as a whole at any desired level of generality.
- The ability to distinguish among warranted, unwarranted, or contradicted conclusions drawn from a body of data. {Knowledge-based systems do this to a limited extent.}
- Ability in making proper qualifications when interpreting data.

2.30 EXTRAPOLATION

Extrapolation is the extension of trends or tendencies beyond the given data to determine implications, consequences, corollaries, effects, etc., which are in accordance with the conditions described in the original communication. Accurate extrapolation requires that one be able to translate as well as interpret a communication, and, in addition, he must be able to extend the trends or tendencies beyond the given data and findings of the document to determine implications, consequences, corollaries, effects, etc., which are in accordance with the conditions as literally described in the original communication. Extrapolation requires that the reader be well aware of the limits within which the communication is posed as well as the possible limits within which it can be extended. In practically all cases, one must recognize that an extrapolation can only be an inference which has some degree of probability--certainty with respect to extrapolations is rare.

Extrapolation as here defined is to be distinguished from application in that the thinking is characterized by the extension of that which is given to intermediate, past, future, or other conditions or situations. The thinking is usually less abstract than in the case of application where use is made of generalizations, rules of procedure, and the like. {It is in this category that much of the work in knowledge-based systems is concentrated in order to improve performance and make the results more generally acceptable.}

Examples:

- The ability to deal with the conclusions of a work in terms of the immediate inferences made from the explicit statements.
- The ability to draw conclusions and state them effectively (recognizing the limitations of the data, formulating accurate inferences and tenable hypotheses).
- Skills in predicting continuation of trends.
- Skill in interpolation where there are gaps in data.
- The ability to estimate or predict consequences of courses of action described in a communication.
- The ability to be sensitive to factors which may render predictions inaccurate.
- The ability to distinguish consequences which are only relatively probable from those for which there is a high degree of probability.
- The ability to differentiate value judgments from predictions of consequences.

3.00 APPLICATION

Application is the use of abstractions in particular and concrete situations. The abstractions may be in the form of general ideas, rules of procedure, or generalized methods. The abstractions may also be technical principles, ideas, and theories which must be selected and applied.

The whole cognitive domain of the taxonomy is arranged in a hierarchy, that is, each classification within it demands the skills and abilities which are lower in the classification order. The application category follows this rule in that to apply something requires "Comprehension" of the method, theory, principle, or abstraction applied.

One way to clarify the distinction between "Comprehension" and "Application" is this. A problem in the comprehension category requires one to know an abstraction well enough that he can correctly demonstrate its use when specifically asked to do so. "Application," however, requires a step beyond this. Given a problem, an individual will apply the appropriate abstraction without having to be shown how to use it in that situation. A demonstration of "Comprehension" shows that one can use the abstraction when its use is specified. A demonstration of "Application" shows that he will use it correctly given an appropriate situation in which no mode of solution is specified. {There does not exist, as yet, a knowledge-based system general enough to require the ability described here. Though one might conclude that a system that incorporates meta-knowledge fits the description, we believe that the intent here is considerably broader than that.}

Examples:

- Application to the phenomena discussed on one paper of the scientific terms or concepts used in other papers.
- The ability to predict the probable effect of a change in a factor on a biological situation previously at equilibrium.

4.00 ANALYSIS

Analysis is the breakdown of a communication into its constituent elements or parts such that the relative hierarchy of ideas is made clear and/or the relations between the ideas expressed are made explicit. Such analyses are intended to clarify the communication, to indicate how the communication is organized, and the way in which it manages to convey its effects, as well as its basis and arrangement.

At a somewhat more advanced level than the skills of comprehension and application are those involved in analysis. In comprehension the emphasis is on the grasp of the meaning and intent of the material. In application it is on remembering and bringing to bear upon given material the appropriate generalizations or principles. Analysis emphasizes the breakdown of the material into its constituent parts and detection of the relationships of the parts and of the way they are organized. It may also be directed at the techniques and devices used to convey the meaning or to establish the conclusion of a communication.

Although analysis may be conducted merely as an exercise in detecting the organization and structure of a communication and may therefore become its own end, it is probably more defensible to consider analysis as an aid to fuller comprehension or as a prelude to an evaluation of the material.

No entirely clear lines can be drawn between analysis and comprehension at one end or between analysis and evaluation at the other. Comprehension deals with the content of material, analysis with both content and form. One may speak of "analyzing" the meaning of a communication, but this usually refers to a more complex level of ability than "understanding" or "comprehending" the meaning--and that is the intention in the use of "analysis" here. It is true also that analysis shades into evaluation, especially when we think of "critical analysis." As one analyzes the relationships of elements of an argument, he may be judging how well the argument hangs together. In analyzing the form of a communication, or the techniques used, one may express opinions about how well the communication serves its purpose.

Analysis may be divided into three types or levels. At one level one is expected to break down the material into its constituent parts, to identify or classify the elements of the communication. At a second level he is required to make explicit the relationships among the elements, to determine their connections and interactions. A third level involves recognition of the organization principles, the arrangement and structure, which hold together the communication as a whole. {The first two levels are obvious requirements for a language-understanding system. The third is one of the primary goals of much artificial intelligence research.}

4.10 ANALYSIS OF ELEMENTS

A communication may be conceived of as composed of a large number of elements. Some of these elements are explicitly stated or contained in the communication and can be recognized and classified relatively easily. However, there are many other elements in a communication which are not so clearly labelled or identified by the writer. Many of these elements may be of paramount importance in determining the nature of the communication, and until the reader can detect them he may have difficulty in fully comprehending or evaluating the communication.

Examples:

- Ability to recognize unstated assumptions.
- Skills in distinguishing facts from hypotheses.
- Ability to distinguish factual from normative statements.
- Ability to distinguish a conclusion from statements which support it.

4.20 ANALYSIS OF RELATIONSHIPS

Having identified the different elements within a communication, one still has the task of determining some of the major relationships among the elements as well as the relationships among the various parts of the communication. At the most obvious level he may need to determine the relationship of the hypotheses to the evidence, and in turn the relationship between the conclusions and the hypotheses as well as evidence. Analysis would also include the relationships among the different kinds of evidence presented.

At a more difficult level is likely to be the analysis of a communication into the parts which are essential to or which form the main thesis as contrasted with those parts or elements which may help to expand, develop, or support this thesis. Much of analysis of relationships may deal with the consistency of part to part, or element to element; or the relevance of elements or parts to the central idea or thesis in the communication.

Examples:

- Skill in comprehending the interrelationships among the ideas in a passage.
- Ability to recognize what particulars are relevant to the validation of a judgment.
- Ability to recognize which facts or assumptions are essential to a main thesis or to the argument in support of that thesis.
- Ability to check the consistency of hypotheses with given information and assumptions.
- Ability to distinguish cause-and-effect relationships from other sequential relationships.
- Ability to analyze the relations of statements in an argument, to distinguish relevant from irrelevant statements.
- Ability to detect logical fallacies in arguments.

4.30 ANALYSIS OF ORGANIZATIONAL PRINCIPLES

This analysis incorporates the organization, systematic arrangement, and structure which holds the communication together. It includes "explicit" as well as "implicit" structure. It includes the bases, necessary arrangement, and the mechanics which make the communication a unit.

At an even more complex and difficult level is likely to be the task of analyzing the structure and organization of a communication. Rarely will the producer of a communication explicitly point out the organizational principles he has used, and quite frequently he may not be aware of those principles. Thus, his purpose, point of view, attitude, or general conception of a field may be discerned in the writing, and the reader may be unable to fully comprehend or evaluate the communication until he has determined them.

Examples:

- Ability to analyze the relationship of materials and means of production to the "elements" and to the organization.
- The ability to infer the author's purpose, point of view, or traits of thought and feeling as exhibited in his work.
- Ability to recognize the point of view or bias of a writer.

5.00 SYNTHESIS

Synthesis is here defined as the putting together of elements and parts so as to form a whole. This is a process of working with elements, parts, etc., and combining them in such a way as to constitute a pattern or structure not clearly there before. Generally this would involve a recombination of parts of previous experience with new material, reconstructed into a new and more or less well-integrated whole. This is the category in the cognitive domain which most clearly provides for creative behavior.

Comprehension, application, and analysis also involve the putting together of elements and the construction of meanings, but these tend to be more partial and less complete than synthesis in the magnitude of the task. Also there is less emphasis upon uniqueness and originality in these other classes than in the one under discussion here. Perhaps the main difference between these categories and synthesis lies in the possibility that they involve working with a given set of materials or elements which constitutes a whole in itself. They involve studying a whole in order to understand it better. In synthesis, on the other hand, one must draw upon elements from many sources and put these together into a structure or pattern not clearly there before. His efforts should yield a product--something that can be observed through one or more of the senses and which is clearly more than the materials he began to work with.

It is to be expected that a problem which is classified as a task primarily involving synthesis will also require all of the previous categories to some extent.

It seems best to distinguish between different kinds of synthesis primarily on the basis of the product. Such an approach does permit classification into three relatively distinct divisions which have some practical significance. Classification on the basis of product is not inconsistent with the taxonomy, since the construction of different products may well require somewhat different processes. A similar assumption is made in the Knowledge and Analysis categories.

5.10 PRODUCTION OF A UNIQUE COMMUNICATION

Under this we include those objectives in which primary emphasis is upon communication--upon getting ideas, feelings, and experience across to others. The important controlling or limiting factors in such tasks are the following: the kinds of effects to be achieved; the nature of the audience in whom the effects are to be achieved; the particular medium through which one expresses himself; and the particular ideas and experiences that the student can draw upon or that he wishes to communicate.

The product of synthesis is rendered unique because of the great latitude allowed the individual in putting his own ideas, feelings, and experiences into it. In other words, much of the content of the synthesis is not rigorously predetermined by the requirements of the task; it flows from the person and is used by him if he alone deems it worthy of incorporating in his work.

Examples:

- Skill in writing, using an excellent organization of ideas and statements.
- Ability to write creatively a story, essay, or verse for personal pleasure, or for the entertainment or information of others.
- Ability to tell a personal experience effectively.
- Ability to make extemporaneous speeches.

5.20 PRODUCTION OF A PLAN, OR PROPOSED SET OF OPERATIONS

Objectives that fall in this sub-category aim, in general, at the production of a plan of operations. The production of the plan constitutes the act of synthesis.

The product, or plan of operations, must satisfy the requirements of the task. Usually the requirements are laid down in the form of specifications or data to be taken into account. These data or specifications may be given, in which case one may assume that they are sound, or they may have to be worked out by him before he can proceed. But in any case, the specifications do furnish a rather well-defined criterion against which the product may be evaluated. In this sense, the product must always meet an empirical test of its soundness. {Though some work has been done in plan production in some artificial intelligence research projects, particularly those concerned with robotics, they fall short of the intent of this category.}

Examples:

- Ability to propose ways of testing hypotheses.
- Ability to integrate the results of an investigation into an effective plan or solution to solve a problem.
- Ability to design simple tools to perform specified operations.
- Ability to synthesize knowledge of chemistry, knowledge of the unit operations, and data available in the technical literature, and apply these to the design of chemical processes.

5.30 DERIVATION OF A SET OF ABSTRACT RELATIONS

In this sub-category we include objectives that require one to produce, or derive, a set of abstract relations. There seem to be two somewhat different kinds of tasks here: (1) those in which one begins with concrete data or phenomena and which he must somehow either classify or explain; (2) those in which one begins with some basic propositions or other symbolic representations and from which he must deduce other propositions or relations.

The first type of task may also take the form of explaining certain observed phenomena. In this case, there is little emphasis upon developing a classification scheme. The problem is to formulate a hypothesis that will adequately account for a wide range of seemingly interrelated phenomena. As with a classification scheme, the hypothesis or theory must fit the facts and, in addition, be internally consistent--i.e., free from logical contradictions. {Diagnosis}

The second broad type of task clearly begins with abstract symbols, propositions, and the like, rather than with concrete data. The problem is to move from these symbolic representations to deductions that can reasonably be made. In other words, one operates within some theoretical framework, and must reason in terms of it. He is thus quite circumscribed in what he does, although the task can permit him to carry his thinking quite far. But always in the background are rigorous objective criteria which his product of synthesis must meet; subjective standards, of the sort that predominate in the first and second sub-categories, all but vanish here. {Meta-DENDRAL performs such a process}

Examples:

- Ability to formulate appropriate hypotheses based upon an analysis of factors involved, and to modify such hypotheses in the light of new factors and considerations.
- Ability to perceive various possible ways in which experience may be organized to form a conceptual structure.
- Ability to make mathematical discoveries and generalizations. {Lenat's work [LENAT76] has actually been recognized as achieving this level of accomplishment.}

<u>Types of Errors that can occur in Synthesis</u>

In general, a synthesis is faulty to the extent that it lacks "goodness of fit" to the requirements of the problem. Faulty synthesis may be due to one or more of the following factors, many of which seem to reflect faulty comprehension and analysis:

(1) Misinterpreting the purpose or nature of the problem.

(2) Misinterpreting the nature of important elements and their interrelations, confusing basic and subordinate elements.

(3) Omitting important elements.

(4) Applying irrelevant or inaccurate elements.

(5) Over-organizing the synthesis, so that the result is too artificial or inflexible to satisfy varying requirements, as with a plan of investigation or an architectural design.

(6) Otherwise failing to satisfy the requirements of an external theory, framework, or of some other standard.

6.00 EVALUATION

Evaluation is defined as the making of judgments about the value, for some purpose, of ideas, works, solutions, methods, material, etc. It involves the use of criteria as well as standards for appraising the extent to which particulars are accurate, effective, economical, or satisfying. The judgments may be either quantitative or qualitative, and the criteria may be either those determined by the individual or those which are given to him.

Evaluation is placed at this point in the taxonomy because it is regarded as being at a relatively late stage in a complex process which involves some combination of all the other behaviors of Knowledge, Comprehension, Application, Analysis, and Synthesis. What is added are criteria including values. Evaluation represents not only an end process in dealing with cognitive behaviors, but also a major link with the affective behaviors where values, liking, and enjoying (and their absence or contraries) are the central processes involved. However, the emphasis here is still largely cognitive rather than emotive.

Although Evaluation is placed last in the cognitive domain because it is regarded as requiring to some extent all the other categories of behavior, it is not nessarily the last step in thinking or problem solving. It is quite possible that the evaluative process will in some cases be the prelude to the acquisition of new knowledge, a new attempt at comprehension or application, or a new analysis and synthesis.

For the most part, the evaluations customarily made by an individual are quick decisions not preceded by very careful consideration of the various aspects of the object, idea, or activity being judged. These might more properly be termed opinions rather than judgments. Customarily, opinions are made at less than a fully conscious level, and the individual may not be fully aware of the clues or bases on which he is forming his appraisals. For purposes of classification, only those evaluations which are or can be made with distinct criteria in mind are considered. Such evaluations are highly conscious and ordinarily are based on a relatively adequate comprehension and analysis of the phenomena to be appraised. It is recognized that this may be far from the normal state of affairs.

One type of evaluation can be made largely on the basis of internal standards of criticism. Such internal standards are for the most part concerned with tests of the accuracy of the work as judged by consistency, logical accuracy, and the absence of internal flaws. It is recognized that even when a document, product, or work is perfectly accurate or consistent on the basis of internal standards, it does not necessarily constitute a work which can be valued highly unless it also satisfies certain external standards. A second type of evaluation may be based on the use of external standards or criteria derived from a consideration of the ends to be served and the appropriateness of specific means for achieving these ends. Such evaluations are primarily based on considerations of efficiency, economy, or utility of specific means for particular ends. This type of evaluation also involves the use of particular criteria which are regarded as appropriate for members of the class of phenomena being judged, i.e., in terms of standards of excellence or effectiveness commonly used in the field or in a comparison of particular phenomena with other phenomena in the same field. {This certainly describes one of the primary functions of the CE, though one would use less anthropomorphic terms.}

6.10 JUDGMENTS IN TERMS OF INTERNAL EVIDENCE

A communication may be evaluated from such internal evidence as logical accuracy, consistency, and other internal criteria. After an individual has comprehended and perhaps analyzed a work, he may be called upon to evaluate it in terms of various internal criteria. Such criteria are for the most part tests of the accuracy of the work as judged by the logical relationships evident in the work itself. Has the writer (or speaker) been consistent in his use of terms, does one idea really follow from another, and do the conclusions follow logically from the material presented? There are other internal standards which may be used to determine that there are no major errors in the treatment or reporting of data and that statements are made with some precision or exactness. It is also possible to judge a work to determine whether the manner in which the writer cites sources or documents or the care with which particulars are given is likely to yield a high probability of accuracy.

Examples:

- Judging by internal standards, the ability to assess general probability of accuracy in reporting facts from the care given to exactness of statement, documentation, proof, etc.
- The ability to apply given criteria (based on internal standards) to the judgment of the work.
- The ability to indicate logical fallacies in arguments.

6.20 JUDGMENTS IN TERMS OF EXTERNAL CRITERIA

These judgments entail the evaluation of material with reference to selected or remembered criteria. The criteria may be: ends to be satisfied; the techniques, rules, or standards by which such works are generally judged; or the comparison of the work with other works in the field. This type of evaluation involves the classification of the phenomena in order that the appropriate criteria for judgment may be employed. Thus, a work of history is to be judged by criteria relevant to historical works rather than to works of fiction. A work of art may be judged by many different criteria, depending upon the classification of the work (e.g., representational, expressional, as communicating a particular message or idea). All of this involves the assumption that each phenomenon is a member of a class and is to be judged by criteria which are appropriate to that class. This also includes the possibility of comparing a work with other members of the same class of work.

It should be pointed out that the classification of a work and the evaluation of it in terms of the criteria appropriate to the class involve arbitrary judgments. Clearly, a work is at one and the same time a member of many different classes. Thus, an historical work may also be a rhetorical, philosophical, or even poetic work. The decision as to the class in which it is to be evaluated does not preclude it from also being evaluated as a member of another class.

Quite frequently, the external criteria are derived from a member of the class which is considered to be a "model" member (in some respects, not necessarily "ideal" or "best"). This may result in the judgment's focusing on the comparison of the two members of the class rather than on the extent to which one member satisfies selected abstract criteria.

This type of evaluation may also involve the classification of a work with regard to the ends to be achieved by the work, followed by a judgment as to whether the means used are appropriate to the ends in terms of efficiency, economy, and utility. This involves the assumption that particular means serve some specific ends better than others, and that particular ends are best served by some specific means. It should be recognized that the major problem in many judgments of this kind is what ends are to be considered. The ends may be those conceived by the originator of the work or idea, or they may be those deemed appropriate by the critic. It should also be recognized that a particular work or idea may be evaluated in terms of many different means-ends relationships. This may require the answering of the following questions: Do the means employed represent a good solution to the problem posed by the end desired? Are the means the most appropriate ones when the alternatives are considered? Do the means employed bring about ends other than those desired?

Examples:

- The comparison of major theories, generalizations, and facts about particular cultures.
- Judging by external standards, the ability to compare a work with the highest known standards in its field--especially with other works of recognized excellence.

APPENDIX B: CURRENT WORK IN KNOWLEDGE-BASED SYSTEMS AND RELATED AREAS

The following is a list of projects that are presently developing knowledge-based systems or performing research and development in directly related areas. It represents the major activities, but we make no claim that it is complete. The list is ordered by institution, and by principal investigator within each institution, giving the name of the project if one exists and a very brief description.

1. Bolt Beranek and Newman, Inc., Cambridge MA
    a. J. S. Brown--SOPHIE, a knowledge-based instructional system.
    b. A Collins--Follow-on to SCHOLAR, a system that models common-sense reasoning.
    c. G. Rider--An intelligent terminal system based on production rules.
    d. W. A. Woods--Natural-language processing.
2. Carnegie-Mellon University, Pittsburgh PA
    a. A. Newell--PSG, PSH, OPS, production-rule "programming" systems that support KBS implementation, OPS is intended to optimize rule sets.
    b. D. R. Reddy--Image-understanding system based on speech-understanding research and HEARSAY II, in particular; continuation of speech-understanding research.
3. IBM Research Center, Yorktown Heights NY
    a. A. Malhotra--Knowledge-based system for management support.
4. Jet Propulsion Laboratory, California Institute of Technology, Pasadena CA
    a. Y. Yakimovsky--A non-deterministic inference system for image analysis.

5. Massachusetts Institute of Technology, Cambridge MA

    a. I. Goldstein--Developing a production-rule system for aircraft simulation.

    b. W. Martin, L. Hawkinson--OWL, a system for representing knowledge and constructing knowledge-based systems.

    c. W. Martin--Rule-based inventory management system.

    d. M. Minsky, S. Papert--Frames, a method for representing knowledge.

    e. A. Rubin--Knowledge-based system for diagnosing kidney disease.

    f. J. Sussman--System for debugging (troubleshooting) electronic circuits.

    g. P. Winston--System for keeping track of icebergs.

6. The Rand Corporation, Santa Monica CA

    a. R. Anderson--RITA, a production-rule system for intelligent terminal user agents.

    b. D. Waterman, R. Hayes-Roth--TECA a naval threat evaluation and countermeasures agent built on RITA.

    c. D. Waterman--System for creating RITA agents by observing and querying the user.

7. Rutgers University, New Brunswick NJ

    a. C. A. Kulikowski, A. Safir, S. Amarel--CASNET and IRIS, systems for recommending treatment of glaucoma.

    b. C. Schmidt--BELIEVER, a limited model of human intentions based on knowledge of motives and beliefs.

8. Stanford Research Institute, Menlo Park CA

    a. H. G. Barrow, J. M. Tenenbaum--Systems to understand photographic images.

b. R. O. Duda--PROSPECTOR, an exploratory geology program, based on MYCIN plus Hendrix net for organizing the relevance of rules

c. N. J. Nilsson--Writing a book on knowledge-based systems.

9. Stanford University, Stanford CA

a. T. Binford--Stereo vision system similar in intent to speech-understanding systems.

b. B. Buchanan, S. Cohen--MYCIN, a rule-based system for recommending antimicrobial therapy for infectious diseases.

c. R. Davis, J. King--TIRESIUS, augments MYCIN's explanations, abstracts its rule model, critiques rules; has a system of meta-rules.

d. R. Davis, J. King--EMYCIN, empty MYCIN, the framework without any rules.

e. E. A. Feigenbaum, B. Buchanan, J. Lederberg--DENDRAL, a system for analyzing electron emission spectograms for determining chemical structures, and Meta-DENDRAL, a companion system for modeling chemists analytical processes and creating rules for DENDRAL.

f. E. A. Feigenbaum, D. Englemore, H. P. Nii--Protein crystallography, analyzes x-ray crystallographic images to determine stick-and-ball model of molecules.

g. E. A. Feigenbaum, J. Lederberg, N. Martin--MOLGEN, a genetic engineering system to assist geneticists in planning laboratory experiments concerned with manipulation of DNA with restriction enzymes.

h. E. A. Feigenbaum--DIKE, domain-independent knowledge engineering, intended to remove some of the variability in knowledge engineering.

10. System Development Corporation, Santa Monica CA

    a. J. Burger, I. Kameny--EUFID, a natural-language interface for data management systems.

    b. C. Kellogg, P. Klahr--DADM (deductively augmented data management), a deductive processor to interface with existing data management systems and data bases to enhance user functions.

11. University of California at Santa Cruz, Santa Cruz CA

    a. W. T. Wipke--A system for synthesizing chemical compounds.

12. University of Maryland, College Park MD

    a. C. Reiger--Conceptual modeling for natural-language understanding.

13. University of Michigan, Ann Arbor MI

    a. R. Lindsay--System to produce standard record of pathology reports, ameliorates differences in jargon from multiple descriptions of many pathologists.

14. University of Pittsburgh, Pittsburgh PA

    a. H. Pople, J. Myers--INTERNIST (DIALOG), medical diagnosis system for internal medicine using abduction; has more than 1000 rules.

15. Xerox Palo Alto Research Center, Palo Alto CA

    a. D. G. Bobrow, T. Winograd--KRL, a knowledge representation language for their version of Frames.

    b. D. G. Bobrow--GUS, a general understanding system whose first application is a travel agent.

16. Yale University, New Haven CT

    a. R. Schank--a system for understanding written material in context.

SDC

SYSTEM DEVELOPMENT CORPORATION ▪ 2500 COLORADO AVENUE ▪ SANTA MONICA, CALIFORNIA 90406 ▪ (213) 393-9411