S

Stanford Heuristic Programming Project
Memo HPP-77-5

March 1977

Computer Science Department
Report No. STAN-CS-77-596

A REVIEW OF KNOWLEDGE BASED PROBLEM SOLVING AS A BASIS FOR A GENETICS EXPERIMENT DESIGNING SYSTEM

by

Mark J. Stefik and Nancy Martin

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER STAN-CS-77-596	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* A Review of Knowledge Based Problem Solving As a Basis for A Genetics Experiment Designing System		5. TYPE OF REPORT & PERIOD COVERED Technical Report March, 1977 6. PERFORMING ORG. REPORT NUMBER HPP-77-5
7. AUTHOR(s) Mark J. Stefik Nancy Martin		8. CONTRACT OR GRANT NUMBER(s) ARPA DAHC 15-73-C-0435
9. PERFORMING ORGANIZATION NAME AND ADDRESS Stanford Computer Science Department Stanford University Stanford, Calif. 94305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency Information Processing Techniques Office 1400 Wilson Ave., Arlington, VA. 22209		12. REPORT DATE March, 1977 13. NUMBER OF PAGES 91
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* Mr. Philip Surra, Resident Representative Office of Naval Research Durand 165, Stanford University		15. SECURITY CLASS. *(of this report)* UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Reproduction in whole or in part is permitted for any purpose of the U.S. Government

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Genetics, Heuristic Problem Solving, Knowledge Bases, MOLGEN, Planning Systems, Representation of Knowledge, Rule-Based Systems

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

It is generally accepted that problem solving systems require a wealth of domain specific knowledge for effective performance in complex domains. This report takes the view that all domain specific knowledge should be expressed in a knowledge base. With this in mind, the ideas and techniques from problem solving and knowledge base research are reviewed and outstanding problems are identified. Finally, a task domain is characterized in terms of objects, actions, and control/strategy knowledge and suggestions are made for creating a uniform knowledge base management system to be used for knowledge acquisition,

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

problem solving, and explanation.

A Review of Knowledge Based Problem Solving As a Basis for A Genetics Experiment Designing System

STAN-CS-77-596
Heuristic Programming Project Memo 77-5

Mark J. Stefik and Nancy Martin

ABSTRACT

It is generally accepted that problem solving systems require a wealth of domain specific knowledge for effective performance in complex domains. This report takes the view that all domain specific knowledge should be expressed in a knowledge base. With this in mind, the ideas and techniques from problem solving and knowledge base research are reviewed and outstanding problems are identified. Finally, a task domain is characterized in terms of objects, actions, and control/strategy knowledge and suggestions are made for creating a uniform knowledge base management system to be used for knowledge acquisition, problem solving, and explanation.

KEY WORDS

GENETICS, HEURISTIC PROBLEM SOLVING, KNOWLEDGE BASES, MOLGEN, PLANNING SYSTEMS, REPRESENTATION OF KNOWLEDGE.

This research was supported by the Defense Advanced Research Projects Agency under ARPA Order No. 2494, Contract No. DAHC 15-73-C-0435, and by The National Science Foundation under Contract Nos. MCS76-11649, and MCS76-11935, and by The National Institute of Health under Contract No. RR-00785

A Review of Knowledge Based Problem Solving As a Basis for A Genetics Experiment Designing System

STAN-CS-77-596
Heuristic Programming Project Memo 77-5

Mark J. Stefik and Nancy Martin

ABSTRACT -- It is generally accepted that problem solving systems require a wealth of domain specific knowledge for effective performance in complex domains. This report takes the view that all domain specific knowledge should be expressed in a knowledge base. With this in mind, the ideas and techniques from problem solving and knowledge base research are reviewed and outstanding problems are identified. Finally, a task domain is characterized in terms of objects, actions, and control/strategy knowledge and suggestions are made for creating a uniform knowledge base management system to be used for knowledge acquisition, problem solving, and explanation.

Key Words: Genetics, Heuristic Problem Solving, Knowledge Bases, MOLGEN, Planning Systems, Representation of Knowledge

Support:

MOLGEN Grants	National Science Foundation MCS76-11649, Stanford University MCS76-11935, University of New Mexico
SUMEX Grant	National Institutes of Health Biotechnology Resource Grant RR-00785
Heuristic Programming Project Contract	Advanced Research Projects Agency DAHC 15-73-C-0435

Table of Contents

Chapter — Page

Acknowledgments iv

I. Introduction 1

I.1 History and Organization of this Document 1

I.2 Philosophical Overview 2

II. General Scope of the MOLGEN Project 4

II.1 A Laboratory Assistant for Molecular Genetics 4

II.2 An Analogy 6

III. Problem Solving and Planning 8

III.1 Introduction 8

III.1.1 Problem Solving as Heuristic Search 8

III.1.2 Problem Solving as Theorem Proving 9

III.2 Fundamental Methods for Problem Solving 11

III.2.1 Means-ends Analysis 12

III.2.2 Problem Reduction 13

III.2.3 Backtracking 14

III.2.4 Hierarchical Planning 15

III.2.4.1 Well Spaced 'Planning Islands' 16

III.2.4.2 Abstraction: The 'Planning Method' of GPS 16

III.2.4.3 Hierarchy of Abstraction Spaces 17

III.2.4.4 Criticality Levels as Abstraction Levels 18

III.2.4.5 Overview of Hierarchical Planning . . . 19

III.2.5 Interacting Goals 19

III.2.6 Using Existing Plans 22

III.3 Summary of Planning Ideas 25

IV. Knowledge Based Systems 27

IV.1 Capabilities for a Knowledge based System 27

IV.2 Design Principles for Knowledge Aggregation 30

IV.2.1 Criteria for Weak and Strong Interactions . . . 32

IV.2.2 Demons and the Multiple Knowledge Sources Model 34

IV.2.2.1 BEINGS and ACTORS 35

IV.2.2.2 Lessons from HEARSAY 36

IV.2.3 Knowledge Access and Control by Description . . 37

IV.2.4 What We Have Learned 40

IV.3 Design Principles for Knowledge Acquisition 41

IV.3.1 Extensibility in Programming Systems 45

IV.3.2 Ideas from Data Base Systems 47

IV.3.2.1 SCHEMATA: Data Definitions 47

IV.3.2.2 Data Models and Accessibility 48

IV.3.2.3 Beyond Retrieval 50

IV.3.3 Knowledge Based Systems for Artificial Intelligence 51

IV.3.3.1 Object Centered Factorization of Knowledge 51

IV.3.3.2 Acquisition of Objects 53

IV.3.3.3 The SCHEMA-SCHEMA 54

IV.3.3.4 Acquisition of Actions 55

IV.3.4 Summary of Knowledge Acquisition Work 56

IV.4 Summary of Knowledge Base Research 57

V. Tentative Proposed Work 59

V.1 Perspectives and Observations about the Direction of this Research 59

V.2 MOLGEN System Sketch 62

V.3 Strategy and the Planning Network 64

V.4 A Toolbox for Artificial Intelligence 67

V.4.1 The Means-ends Tool 68

V.4.2 Means-ends Analysis in the Schemata Network . . 69

V.4.3 More From the Toolbox 71

V.4.4 Eliminating Special Cases 71

V.5 Concluding Remarks 75

Appendix I

Working Bibliography 76

Acknowledgments

Sincere thanks --

To Bruce Buchanan, Ed Feigenbaum, and Joshua Lederberg for their enthusiasm and insightful suggestions and for providing such a rewarding research environment.

To Harold Brown, Ray Carhart, Randy Davis, Jerry Feitelson, Peter Friedland, Jonathan King, Penny Nii, Nils Nilsson, and Terry Winograd, who all puzzled through earlier drafts of this manuscript and gave generously of their time and ideas.

Chapter I

Introduction

I.1 History and Organization of this Document

Since the early days of the DENDRAL project, Bruce Buchanan, Ed Feigenbaum, and Joshua Lederberg have wanted to collaborate on an artificial intelligence project in molecular genetics. Periodically they reviewed the potential for such a project considering the developments both in molecular genetics research and in artificial intelligence research. In the Spring of 1975, research was picking up momentum in molecular genetics with the development of a number of highly specific laboratory techniques based on restriction enzymes. At the same time, progress was evident in the development of software for management of knowledge time. A research group was formed at Stanford calling itself the MOLGEN project. Several geneticists have become involved in the project including Dusko Ehrlich, Douglas Wallace, Douglas Brutlag, and Jerry Feitelson. The computer science research is being done by researchers at the Heuristic Programming Project, which is directed by Ed Feigenbaum and Bruce Buchanan. The MOLGEN research effort is being directed by Nancy Martin. Many of the domain related questions have involved graph theoretical research which has been mostly done by Harold Brown. The MOLGEN project now includes three computer science graduate students - Peter Friedland, Jonathan King, and Mark Stefik. This report is a slightly revised version of Stefik's thesis proposal submitted in December 1976.

The report which follows is divided into four major sections. Chapter II is an overview of the task area in molecular genetics which is the domain of the MOLGEN system. It reviews the nature of some of the experiments in molecular genetics that are being done and introduces the problem solving task for the MOLGEN system as the interactive design of laboratory experiments. The design of experiments requires the facilities of a problem solving system and Chapter III is a review of fundamental ideas and recent research in general problem solving. One of the challenges of molecular genetics as a task area is the large amount of domain specific knowledge that seems to be required for effective problem solving. Chapter IV is an overview of the research that has been done in knowledge based systems with emphasis on techniques for the acquisition and use of knowledge. In this chapter, Section IV.2 offers a viewpoint on the aggregation of knowledge which may be seen as being either weakly or strongly interacting. Section IV.3.4 explores the contributions of research in several areas of computer science, including data base management. Finally, Chapter V re-examines both the problem solving and knowledge base work and proposes research and a design for the MOLGEN system.

I.2 Philosophical Overview

Since Newell and Simon introduced their program Logic Theorist in 1956, many workers in artificial intelligence have done research toward building computer systems capable of problem solving. For many researchers the ambitition has been to create a computer system with a general problem solving ability that could play a useful role in human affairs. The difficulties of producing a general and powerful system have lead researchers to limit their efforts in two possible ways. Some researchers have concentrated on very small test domains (sometimes termed "toy problems") in order to develop techniques applicable to larger domains. (See for example [Fikes72b] or [Green69]). Although this research has uncovered some basic and fundamental problems and solutions for some of them, the programs have not in fact developed into powerful and general problem solving systems. One of the reasons for this relates to the size of the knowledge base that is involved in practical problems. For example, the knowledge base for designing scientific experiments is of a different order of magnitude than that for stacking blocks. The relevance for this remark is based on an observation by Dijkstra, that any two things which differ in some respect by a factor of a hundred or more are utterly incomparable. As Dijkstra notes, one cannot design a jet airplane by taking the design for a child crawling across the floor and scaling up by the ratio of the relevant speeds. Entirely different design principles need to be invoked. Thus the methods of resolution theorem proving or Means-ends analysis cannot be carried directly from the small test domains into the large systems. Significant problems arise simply from the size of the knowledge base.

Other researchers have built performance programs in larger but judiciously chosen areas of human problem solving and have demonstrated the importance of using a large amount of domain specific knowledge to guide the problem solving process effectively. (See for example [Buchanan69] or [Nilsson74]). Although several of these systems have achieved impressive results within their chosen domains, the systems created have not illustrated a general problem solving power by subsequent extension to other domains. One reason for this is that the performance programs have typically used ad hoc approaches for knowledge representation, which haved proved too rigid to accommodate a variety of task domains. These systems have served to highlight what are now recognized to be some major stumbling blocks for large knowledge based systems -- acquiring the domain and strategy knowledge from a user and integrating it into a knowledge base so that it can be used effectively.

The difficulties in building and maintaining large computer systems is not unique to artificial intelligence. Ideas about the organization of such systems have come from several areas of computer science. Dahl, Dijkstra, and Hoare have made important steps toward creating a science of large program and system development. One of the tenets of building large systems is based on an observation of the limitations of the human mind. The observation is that precise thinking is possible in terms of only a small number of elements at one time. In programming terms, this suggests that a system should be designed hierarchically in smallish chunks. This design process mandates that the operation of the entire system can be comprehended in terms of the

subsystems which are mentioned in the top-level description of the system. Proof of the correct functioning of the overall system at the top level is based on the assumption that each of the lower level subsystems will function as specified. Each of the subsystems in turn is also written as one of these chunks so that it, too, is easily comprehended. This approach to designing a system in layers is an embodiment of the power of abstraction which keeps the size of the component pieces of the system manageable. Dijkstra has suggested in [Dahl72] that this is a critical design principle for large systems.

> Summarizing: as a slow-witted human being I have a very small head and I had better learn to live with it and to respect my limitations, rather than try to ignore them, for the latter vain effort will be punished by failure.
>
> Dijkstra in [Dahl72]

The same issues which arise in the development of a large program are present in the development of a large knowledge base for an artificial intelligence problem solving system. Both the extensive domain knowledge and general problem solving knowledge necessary in a large knowledge based system need to be organized into small comprehensible chunks which can be acquired and used. Recently techniques for knowledge acquisition and explanation for knowledge based systems have been reported in [Davis76c] or [Winograd73]. A basic theme throughout this work is that a system can be in some sense aware of what it knows when it has a knowledge of its own representations[1] It will be seen that models of knowledge (termed schemata) serve to structure knowledge into its component parts and provide a source of the system's awareness of what it knows. Schemata also provide a model for guiding the knowledge acquisition process and a means for integrating new knowledge into an existing knowledge base. The structuring of domain and planning knowledge, which facilitates human understanding of that knowledge, also makes feasible its acquisition by the system, its explanation to the user, and its effective use in problem solving by the system.

Finally, the motivation and thrust of the MOLGEN project is the solution of a broad class of problems from molecular genetics. These problems are all drawn from the task of designing laboratory experiments and will utilize the problem solving techniques that have been developed in artificial intelligence. Chapter II discusses the classes of genetic experiments to be considered and Chapter III delineates and examines the variety of problem solving techniques in artificial intelligence -- illustrating their differences, potentials, and some unsolved problems. Thus, this project will include a synthesis of ideas from the most recent problem solving systems, from knowledge based systems, and from structured programming.

---

[1] Chapter IV surveys this research and Chapter V proposes a number of extensions to it.

Chapter II

General Scope of the MOLGEN Project

II.1 A Laboratory Assistant for Molecular Genetics

MOLGEN is to be a computer-based system capable of reasoning about experiments in molecular genetics. For the purposes of MOLGEN, the world of molecular genetics consists of genetic objects (mostly DNA structures) and operations on these objects.

Observable Attributes:	Radioactivity, U.V. Absorption etc.
Theoretical Attributes:	Nucleotide Sequences Bonding Patterns etc.

Figure 1. Attributes of Structures in the world of MOLGEN

The structures may be viewed as having both observable and theoretical attributes. The observable attributes are the readings from actual laboratory measurements and correspond to those features of structures which can be measured. These includes such things as biological activity, radioactivity, ultraviolet absorption, or electron microscopy observations. The theoretical attributes are those molecular features hypothesized in the theory of molecular structures which are not directly viewable. This includes such things as DNA precise bonding patterns or known nucleotide sequences. The dichotomy between observable and theoretical attributes of genetic structures may appear at times to be academic since many of the theoretical objects are only one step away from being observable and it is natural to lump together an attribute with the physical observation of that attribute. For example, a bubble is a structural attribute which corresponds to a particular substructure of DNA defined in terms of a characteristic bonding pattern. If a structure containing a bubble is prepared for viewing under the electron microscope and photographed, then a characteristic picture is generally observed. It is tempting to use the term bubble to mean the hypothesized physical attribute or the observation interchangeably. In the system being proposed, the preparation of the structure and the viewing of the photograph constitute one of the MOLGEN transformations. This transformation contains information relating to the probability that the structural bubble will survive the preparation for viewing and the probabilities that other non-bubble structures will be misinterpreted as bubble observations due to unusual overlapping of structures. It is precisely

the practical information of this type which separates a hypothesized bubble from its observation.

The MOLGEN transformations are the available laboratory techniques which transform structures or which make no apparent physical change but cause a theoretical structural attribute to become visible. These transformations make up the "legal moves" in the laboratory and sequences of these transformations may be put together to form experiments. Since hierarchy in knowledge structuring has already been mentioned in the introduction as being important, it should be noted that biologists themselves describe genetic knowledge hierarchically. For example, Eco R1 [1] is the name of an enzyme which cleaves DNA inside the nucleotide [2] sequence "GAATTC". Use of this particular enzyme may be considered to be a legal move in MOLGEN. Eco R1 is a particular example of a restriction enzyme, a class of enzymes which can be characterized in terms of restriction sites. [3] A restriction enzyme is a particular type of endonuclease, that class of enzymes which cuts DNA at a non-terminal nucleotide, and endonucleases are a subset of the nucleases which cut polynucleotides. Reasoning about these enzymes may take place at any of these hierarchical levels of descriptions, so that knowledge about MOLGEN transformations may be seen to be hierarchical. Similarly it is well known that DNA can be organized into genes and punctuation, and that these are further organized as sequences of nucleotides. Thus, genetic structures are hierarchically organized.

Within the context of structural problems, there are two major goals in genetics experiments: (1) structural synthesis and (2) structural analysis. In the synthesis experiments, the program can be given a starting sample of DNA as well as a target sample. Designing a synthesis experiment involves finding a sequence of experimental steps (or legal moves) to transform the initial structure into the target structure. Synthesis may also be designed in a backward sense, seeking any suitable starting structures which can be transformed into the target structure. The general task of analysis is the structural elucidation of an unknown sample. Specifically, an analysis experiment seeks to discriminate between competing hypotheses of structure for a sample. A very basic form for an analysis experiment is the binary discrimination experiment. In this case we are given two competing sample hypotheses. Designing an analysis experiment means to find a sequence of experimental steps whose final outcome yields distinguishable sample characteristics in the observable world of genetics for the alternate sample hypotheses.

As an automated laboratory assistant, there are two major tasks which the program is expected to perform: (1) experiment checking and

---

[1] from Escherichia coli RY13

[2] DNA consists of nucleotides which form the letters of the genetic alphabet. Nucleotides have two parts -- a sugar backbone and a base and are distinguished by their bases. The four common bases are adenine, guanine, cytosine, and thymine. These are commonly abbreviated as A,G,C, and T respectively.

[3] Restriction sites are those places at which the enzyme will cleave the DNA molecule. These may be characterized in terms of nucleotide sequences characteristic for each enzyme.

(2) experiment designing. Experiment checking involves the computer simulation of previously designed experiments. This means that a set of input samples would be defined and a specific sequence of laboratory steps would be given. The computer system would then simulate the sequence of transformations on the representations of the samples terminating finally with a set of new samples. These new samples can be compared with actual laboratory results as a test of the initial hypotheses or of the accuracy of the transformations in the knowledge base. Such a system would be used by the system designers for debugging the transformation knowledge base and by geneticists for comparing the predicted results from the MOLGEN system against actual laboratory experiments. The checking facility would also be used to compare alternate experimental designs before investing any laboratory effort. A more sophisticated task for the program is the designing of experiments. This means that the program would need to know of the strategies involved in building sequences of transformations. This strategy knowledge would be in addition to the legal moves of genetics and encompasses a broad range of knowledge including such things as plan sketches for various contexts, design cost heuristics which predict the costs of considering certain design options, and mechanisms for evaluating the relevance and specificity of laboratory transformations to the current problem.

A substantial part of the effort in creating a system capable of designing experiments as a laboratory assistant centers around the creation and maintenance of an extensive genetics knowledge base. These imply a number of system capabilities to facilitate knowledge acquisition, integration, and debugging which are discussed in Chapter IV.

II.2 An Analogy

A knowledge based experiment designing program for molecular genetics may be viewed constructively in terms of an analogy involving an intelligent assistant (the design program) for using a very awkward text editor (lab techniques). The genetic structures being investigated form the "text" for the text editor. In a synthesis experiment, the geneticist is using the text editor to enter or modify some text; for an analysis experiment he is trying to read the text. The commands that the editing program can accept for manipulating the text, corresponding to the actual laboratory steps or legal moves of genetics, are quite awkward and at times ill suited to the task at hand. For example, some parts of the text are in invisible characters forcing the geneticist to issue commands to first change the text in specific ways; to modify the text he must first find ways to protect other regions of the text; to add new text he must limit himself to adding pieces from other text which he has around. The design program can be viewed as an intelligent assistant which has a good deal of experience with the ins and outs of the very awkward editting program. In addition to giving good advice based on its understanding of the text editor and the geneticist's intent, the assistant must be prepared to accept changes to its knowledge base since the manual for the text editor is continually updated as the user discovers the effects of the various commands. The assistant must also be prepared to accept new

strategies for using the editor and incorporate these strategies in a way which effectively improves the quality of his assistance.

Chapter III

Problem Solving and Planning

III.1 Introduction

Since LT, the Logic Theorist, was introduced by Newell, and Simon in [Newell56], problem solving research has been concerned with techniques of problem solving and methods for expressing the problems. Nilsson in [Nilsson74] gives an excellent survey and family tree of problem solving systems in artificial intelligence. The word planning in artificial intelligence connotes prior analysis involving perhaps a sense of abstraction or remoteness from the primitive details of problem solving. An intelligent problem solver may be expected to plan a strategy for solving a problem. In this terminology, the MOLGEN project wants to use planning in the design of experiments.[1] Chapter III discusses the fundamental ideas from artificial intelligence which can be used in the generation of plans; Chapter IV will deal generally with the issues and problems of managing of a large knowledge base. First some broad classical frameworks suitable for viewing experiment designing as problem solving will be presented. These frameworks will illustrate the task in a rather simplified form in preparation for Section III.2 which will discuss the more specific strategies for planning with insights into specific applicabilities and limitations. Finally Section III.3 will summarize some of these strategies and introduce some issues which show the impact of some of the ideas from knowledge based systems on the open questions in problem solving.

III.1.1 Problem Solving as Heuristic Search

The term heuristic search[2] has come into general usage in artificial intelligence to characterize problem solving methods which are represented as a large tree of subproblems. Solutions exist at unknown locations in unexplored areas of the tree. Judgmental rules, called heuristics, are applied to direct the search towards finding a satisficing[3] solution. The program begins its search along partial

---

[1] Section III.2.4.1 illustrates the numerical meaning of the word "planning".

[2] See for example [Sandewall71].

[3] Simon in [Simon69] coined the term "satisficing" methods to characterize those methods that look for good or satisfactory solutions instead of optimal ones. In many satisficing situations, the expected length of the search depends on how high the standards for the solution are set, but hardly at all on the size of the search space. Simon gives as an example the time required to search a haystack for a needle sufficiently sharp for sewing. The time required depends on the density

paths and stores a tree of the paths it has explored. Typically a number is attached to the end of each branch to express the estimate of further gain should that path be completely explored.

Since the notion of heuristic search is so general, the problem of designing a molecular genetics experiment fits within the paradigm of heuristic search at several different levels. Although many of the planning ideas that will be described below may be classified generally as heuristic search, they represent specialized insights into particular approaches which may be missed in the most general framework. For this reason, the formulation of the experiment designing problem which follows may be viewed as a rather simplified rendition for using heuristic search which will be expanded upon in the later sections. In this simplified formulation, the top node of the search tree represents the formal starting state of a genetics experiment, for example, the initial genetic structures in a synthesis experiment. The alternatives at each step in the plan are the various possible laboratory steps that could be applied to transform the current genetic sample toward the desired structure. Similarly, a binary analysis experiment can be represented as a heuristic search. In this case, the initial state is a pair of alternative hypotheses for the structure and the desired goal state is a new state where some difference between the hypotheses has become observable. As before, the alternatives are the various laboratory transformations. The heuristics, which guide the choice of transformations at each step in planning, reflect the expertise and judgement of the geneticist.

Several algorithms have been developed to assist in choosing a minimum cost path in a heuristic search tree [Nilsson71], where an estimating function is available to measure how close any intermediate state in the experiment is to a final state. In order to guarantee that the algorithms will find a minimal path, the estimating function must never overestimate the distance to the goal. For complex problems, a practical difficulty continues to appear in many contexts. Simply stated, it is sometimes best to retreat from a goal in order to get closer to it. In mathematical theorems, this arises in those cases where it is easier to solve a more general theorem than a specific one. In organic synthesis, it is sometimes better to build up a rather complicated structure which seems farther from a target compound than some current step in the synthesis, but from which an elegant reaction will transform the complicated structure almost directly to the desired product. These difficulties in designing are not limited to scientific problems, but arise almost immediately in the course of automatically designing a sequence of actions in quite restricted domains.

III.1.2 Problem Solving as Theorem Proving

Newell and Simon's Logic Theorist program, an early approach to automatic theorem proving mentioned in the introduction, was based on the approach of heuristic search. In this framework, the situations are viewed as theorems, the operators are the rules of inference, the initial situation is a set of theorems assumed to be true, and the goal situation is the theorem to be proved. Much of the activity of the

---

distribution of sharp needles but not on the total size of the haystack.

program centered around the problem of deciding which rule of inference to apply next. Since that time, logicians have been developing techniques for proving theorems in the first order predicate calculus. Since first order predicate calculus allows quantification, it appears to be rich enough to cover much of the mathematics in science and engineering.[4] J.A. Robinson in [Robinson65] introduced a procedure for proving theorems using a single rule of inference, resolution, which can easily be used in an automatic theorem proving program. Performance of resolution based theorem proving systems reached such impressive levels that it gave rise to the vision of expressing all problems in the predicate calculus and using a single powerful theorem proving engine to do the proofs. It seemed that an elegant solution to theorem proving, which had started as a problem solving application, could be used generally enough to treat problem solving itself as an application. A number of systems based on this idea have been reported in the literature. (See for example [Green69].)

In spite of some initial excitement for this idea, a number of practical difficulties have become apparent. The problem of consistency in a large knowledge base is at the heart of an inherent difficulty with the general use of predicate calculus to express problems. Bobrow in [Bobrow75b] gives the following three theorems as an example:

> All birds can fly.
> Ostriches cannot fly.
> An ostrich is a bird.

The difficulty derives from the fact that any set of inconsistent theorems can be used to prove anything at all, for example, that two equals three or that the moon is made of green cheese.

Another serious difficulty with the methodology was presented by McCarthy as a challenge in [McCarthy64]. In this memo, McCarthy presents the problem of covering with dominoes a checkerboard having two opposite corners deleted. It is well known that it is impossible to carry out this operation. The difficulty of using a theorem-proving engine in this problem lies in the fact that in some sense the real problem is in realizing that the problem is impossible. Newell in [Newell65] sketches an approach to this problem which demonstrates that the proof that the covering is impossible may be expedited if the program knows about mathematical induction and can find a suitable invariant, namely the number of uncovered black squares minus the number of white squares. Use of this new knowledge constitutes what has been termed a representational shift.[5]

Re-formulation of large problems into a form usable for theorem proving is a difficult task. Even such simple classic examples as the Tower of Hanoi or the Monkey and Bananas Problem typically require

---

[4] See [Meltzer68] or [Robinson68] for a very readable discussion on the use of higher level or full predicate logics for expressing a range of problems.

[5] See [Amarel68] for an example problem where a sequence of shifts of representation are used to make the Missionaries and Cannibals problem easier to solve.

several attempts by the user to represent them adequately. Much of the awkwardness derives from the bookkeeping that seems to be necessary to keep track of the changes in the world state as alternate paths are explored in search of a solution. Delegating the responsibility of this to a theorem prover often means that considerable theorem proving effort is necessary to carry both changed and unchanged facts through state transitions. A number of approaches to this problem, which has been termed the frame problem, are discussed in [Raphael71]. The difficulty of problem expression combined with the inherent sensitivity of the system to inconsistency has led to a belief among many researchers in artificial intelligence that a pure theorem proving approach will not be practical for large real world problem solving. (See for example [Feigenbaum71]).

The common wisdom in artificial intelligence regarding heuristic search and theorem proving for problem solving systems is that the heuristic search methods are more efficient at finding solutions because the philosophy of the approach stresses the importance of domain specific information to guide the system to a solution. Theorem proving systems, although they are more difficult to steer [6], are in some ways more capable of using what they know because of their reasoning abilities. This apparent dichotomy of abilities has lead some researchers to try to combine the best of both approaches. The STRIPS problem solver reported in [Fikes72] used a heuristic approach known as Means-ends analysis (See Section III.2.1 below) to guide the search for operators and a resolution theorem prover to check operator applicability. Another system reported in [Kling71a], ZORBA-I, used an approach to reasoning by analogy to guide a theorem proving system. Both of these systems are described in more detail in Section III.2.6 below.

III.2 Fundamental Methods for Problem Solving

Since the frameworks of simple heuristic search and theorem proving described above are inadequate for general problem solving, much work has gone into developing more powerful methods. A commonplace observation has been that much domain-specific knowledge is needed but it has been generally believed that much of what can be stated about strategy must be in some sense domain independent. For example, people who are good problem solvers in one area are often able to solve problems in another area. This belief has lead to a search for fundamental techniques. The following section describes the techniques which have been recognized by this research. It ranges from rather general notions like Means-ends analysis or abstraction to specific proposals for incorporating notions of hierarchy in a domain-independent way.

---

[6] Resolution strategies like the unit preference rule, which gives preferred status to resolutions which might lead to the null clause, or the set-of-support strategy give the system a sort of directionality. They do not amount to a full goal-driven or goal-seeking strategy in the sense of providing domain-specific guidance to the selection of subgoals.

III.2.1 Means-ends Analysis

Means-ends analysis is a technique for problem solving pioneered by Newell, Simon, and Shaw in their classic GPS system[7]. GPS was designed for use in experiments in problem solving by computer and much of the progress in this area has been inspired by this early effort. In one GPS formalism, a representation is given for a current and a desired goal state and a mechanism is given for detecting differences between the states. Actions, which change objects or situations, are also defined. The task for GPS is to select a sequence of actions to remove the differences. To do this, GPS requires a table of connections which associate each kind of detectable difference with the actions relevant to reducing that difference. Implicit in this technique is the reasoning that if there is a sequence of differences D1, D2, D3, ... , Dn and action A1 removes difference D1, A2 removes D2, etc., then the sequence A1, A2, ..., An will transform the current situation into the goal situation.

As Simon [Simon69] points out, one might say this reasoning is valid in worlds where actions are additive or factorable. However, the problems to which problem solvers must addresss themselves are seldom completely additive in this way. Actions have side effects. The order in which goals are achieved is important. (See Section III.2.5.)

In practice, the differences and their associated operators are ordered in terms of importance to direct the process to the most important differences first. Thus the system iterates a cycle of finding the most important differences between the current situation and the goal situation, and then finding an operator to act on that difference.

The one step at a time approach of this version of Means-ends analysis[8] is characteristic of a number of methods known as forwards reasoning. The operators in such systems are sometimes represented in terms of production rules and a set of such rules together with a mechanism for their application is termed a production system. For example, if A is an operation and B,C, and D are sufficient conditions[9] for its use then the following might be used to represent the operator:

Presuppositions: B and C and D
Operation: A

The productions are arranged in such a way that each application of a production rule during the problem solving process makes changes in the world to reflect progress toward the goal. These changes allow

---

[7] See [Newell59] and [Ernst69].

[8] A modified version of Means-ends analysis incorporating problem reduction, as reported in [Ernst69], will be discussed in the next section.

[9] For forward reasoning systems, these conditions have sometimes been termed "presuppositions". The implication is that they must be satisfied before the operation can be applied. In the problem reduction systems (See Section III.2.2.), the conditions can be used to set up subgoals.

other production rules to trigger and carry the solution another step forward.[10] In the Means-ends characterization above, each difference corresponds to the conditional part of a production rule and the associated operator corresponds to the action part or right hand side of a production rule. The primary feature of this approach is its flexibility. Although it is a relatively simple system, it affords rather complex goal-seeking behavior with flexible reasoning from states which may be close to or distant from the goal state. Forward reasoning has been termed goal seeking in [Nilsson76]. The basic simplicity of this method limits its ability to cope with large problems since the worst case time to approach a goal N steps away where there are K potential operators at each step grows as K^N.

III.2.2 Problem Reduction

One of the most basic techniques used to tackle large and complex problems is the idea of factoring them into independent subproblems. When the subproblems that are used correspond to simpler instances of the original problem so that the same technique is applicable, this process can be recursive. Because of the plans within plans nature of this process, Simon in [Simon69] has called this a formal hierarchy in contrast to a more general notion of hierarchy where the subproblems are not necessarily independent.

Such techniques are called reduction methods. In many cases work proceeds backwards from a goal state towards starting states and the subproblems are encountered in the process of satisfying necessary preconditions. For example, suppose that A is a goal state and B, C, and D are necessary preconditions. Then the following reduction rule may be used to represent the relationship.

```
Preconditions: B C D
Goal: A
```

Alternatively, if B, C, and D are actions, problem reduction could be similarly expressed as follows.

```
To Achieve:   A
Apply:        B C D
```

The term backwards reasoning should not be taken as referring merely to the direction that the problem solver uses on a problem, that is, from a goal situation to an input situation using inverse operators. The important point is that the problem is factored into independent subproblems by establishing subgoals. Nilsson terms this technique problem reduction or reasoning backwards. Some authors call it top down or goal driven planning in contrast with forwards reasoning systems which are termed bottom up or data driven.

The Means-ends analysis algorithm presented in the previous section may be modified slightly to carry out problem reduction. This

---

[10] See [Davis76a] for an overview of some ways of representing the memory aspects of a changing world state.

extension involves distinguishing between two basic criteria for selecting operators termed desirability and feasibility by Ernst and Newell. Desirability means that an operator should produce an object that is similar to the desired situation. Feasibility means that the operator should be applicable to the input situation. When GPS selects an operator according to its desirability, this amounts to establishing a subgoal. This version of Means-ends analysis was an important part of GPS. The desirability considerations for a problem viewed in one direction are equivalent to the feasibility considerations for the opposite direction.

Returning to the representation of domain actions as production rules, it is often useful to distinguish between conditions which are "presuppositions" and those which are "preconditions". Production rules whose presupositions are satisfied may be said to be feasible. For production rules which are desirable, it satisfaction of the preconditions may be set up as a subgoal. Thus the operational distinction between preconditions and presuppositions is whether any planning effort can be allocated to satisfy them. If the costs and potentials of satisfying conditions can change, it becomes a question for the knowledge base which way a given condition should be treated.

Many authors have demonstrated that a system can at times usefully employ a combination of forwards and backwards reasoning more effectively than either alone. Whether to reason forwards or backwards depends on the domain. If there are few goals and many rules, then reasoning backwards is likely to be more efficient. If there are few rules and many possible goals, reasoning forwards might be preferred.

III.2.3 Backtracking

When it is possible to sketch out the solution path to a problem as a single tree of fixed subproblems, then the technique of factoring big problems into subproblems is entirely sufficient. For many practical problems the component subproblems depend on the particulars of each situation. Alternative approaches may be given with the intention of picking the one that works best. This suggests that a problem solver must have a mechanism for trying some steps in a plan in a tentative fashion, leaving open the option of discarding them later for something else.

Considerable work on this idea has come from the development of the MICROPLANNER system, implemented at MIT by Sussman, Winograd, and Charniak. (See [Sussman72].) The first implementation contained an automatic backtracking strategy where the failure of any goal resulted automatically in the undoing of the computation back to the failure point where another alternative would be selected. If the alternatives at that point are exhausted, backtracking would continue back further to the next point. Experience showed that this strategy often resulted in much wasted computation. For example, if a goal was to achieve (A and B) and B failed after A succeeded, the failure would automatically cause both steps in the plan to be undone. A subsequent alternative might require A to be done over again resulting in an apparent computational waste. Even more serious is the possibility that the system will backtrack to another alternative which is doomed to perform exactly the same calculation and fail again. One example is that of a

robot building a block structure. His programming is such that his first alternative is to always try first to pick things up with his right hand. In the course of building, he picks up a block which is very hot and burns his right hand. He drops it at once, commences automatic backtracking and tries again with his left hand. Criticisms of automatic backtracking and suggestions for other mechanisms have been reported in detail in [Sussman72] and resulted in the development of a newer system known as CONNIVER.

An emerging consensus on the backtracking question is that the backtracking concept has been used to cover too broad a spectrum of situations. A variety of situations needs to be distinguished and specialized solutions need to be used. One example of a general problem formerly covered by backtracking is that of interactions between higher level goals giving rise to conflicts deep in the refinement process. Instead of simply backtracking and choosing new higher level subproblems, it is generally better to use techniques which analyze the nature of the interactions. (Techniques for handling interactions between goals are discussed in Section III.2.5).

The CONNIVER philosophy switched from that of PLANNER toward providing some lower level mechanisms from which a programmer could implement his own particular approaches to the backtracking problem. Following the ideas in [Bobrow72], the CONNIVER language included a construct known as a context tree, where each context or data frame was in effect a copy of the state of the world which could be passed to daughter nodes. For simple backtracking, any changes made by the daughter process during problem solving would simply be discarded when the daughter node returned. Alternatively using the ADIEU mechanism, contexts may be selectively returned so that computations made by the daughter node need not be repeated. Another CONNIVER construct, the AU-REVOIR mechanism which permits computation to be resumed at a given point in a daughter node, creating what Bobrow in [Bobrow74] called a co-routine regime. The advantages and uses of the various control mechanisms are not settled yet and more work will need to be done before the issues are thoroughly clarified.

III.2.4 Hierarchical Planning

The notion of hierarchical planning reflects an inherent aspect of planning - that planning, to be efficient, must take place in successive levels of abstraction. This means that the highest levels of planning must consider operations or legal moves that are in some sense removed from the numerous alternatives at the primitive level of the domain. Hierarchical planning reflects the wisdom that a program which spends all of its time worrying about the details in a subject area can achieve only the solutions to toy problems. The following sections discuss the historical development of the ideas of hierarchical planning and attempt to clarify exactly what the ideas are. They start with an mathematical elaboration of the problem, discuss some approaches to using this idea, and conclude with a framework for hierarchical planning which may be useful in a variety of domains.

III.2.4.1 Well Spaced 'Planning Islands'

Generally speaking, a well chosen division of a problem into subproblems can have enormous implications in the reduction of search time. In his excellent early survey article of artificial intelligence [Minsky61], Marvin Minsky demonstrated a reduction by what he termed a fractional exponent. In a search tree with 10_2 branches descending from each node, a 20 step search might involve 10^{20} trials, clearly out of the question for a real search. Suppose that four points or "planning islands" along the path can be found at levels 4, 8, 12, and 16 of the planning tree. This strategic placement divides the initial large search into five independent searches of four levels each requiring a total of only $5\ 10^4$ trials.

As Minsky concludes

> Thus it will be worth a relatively enormous effort to find such islands in the solution of complex problems. Note that even if one encountered, say, 10^6 failures before success, one would still have gained a factor of perhaps 10^{10} in overall trial reduction[11]. Thus, practically any ability at all to plan, or analyze a problem will be profitable if the problem is difficult.

This reduction is dramatic indeed although it depends heavily on the placement of the islands. For example, if the islands were placed at levels 16, 17, 18, and 19 in the planning tree, the search would still require 10^{16} trials. Thus we see that merely breaking a problem into subproblems is not nearly as powerful an idea as breaking it into well-spaced subproblems. Perhaps the most straightforward approach to finding planning islands is to use a simplified or abstracted model of the problem situation. The idea is to have an abstract model which preserves the character of the problem situation but with much of the detail suppressed. A solution to the abstract problem could then be used to provide planning islands in the more complex space of the original problem. These islands may be regarded as a sequence of subproblems in the original space. Even if the abstracted problem is not a perfect homomorphism of the original, its solution may prove useful as a guide. The next section introduces an approach to abstraction used to supplement the Means-ends analysis of GPS.

III.2.4.2 Abstraction: The 'Planning Method' of GPS

Newell, Shaw, and Simon reported an auxiliary technique for GPS beyond Means-ends analysis termed the Planning Method in [Newell59] and used it to find proofs in propositional logic. The main steps of the method are:

---

[11] Because of the central importance of this idea in pruning problems down to manageable size, Minsky and other writers have termed this activity planning to connote a high level of processing distinct from the actual searching of the problem space. In the MOLGEN context, this terminology would permit the use of the planning heuristic to do experiment designing.

a. Abstracting by omitting details of the original objects and operators to form an abstract problem space.

b. Forming the corresponding problem in the abstract space.

c. Solving the abstract version of the problem using Means-ends analysis.

d. Using the solution of the abstract problem to form planning islands for the original problem.

e. Solving the original problem.

The method actually contained failure points and loops between the steps shown above so that, for example, alternate solutions from the abstract space could be used for making planning islands in the original space. The particular abstraction scheme that was employed for both states and operators was to (1) ignore differences among logical connectives (AND and OR), (2) ignore negations, and (3) ignore the order of symbols.

This abstraction scheme may actually generate no plans or many plans, although it can be guaranteed that an abstract plan exists if a plan in the original space exists. Because of the abstraction process, some of the plans that it generates may have no counterpart in the original space. The method appeared to be very powerful in producing proofs.

III.2.4.3 Hierarchy of Abstraction Spaces

As Polya has noted in [Polya54], society's aphorisms contain great kernels of wisdom if we can but learn when to apply them.

> If a little bit helps some, try some more.

The abstraction scheme from GPS was only used at one level. Could the abstraction itself be abstracted? Although the scheme used for GPS would need a different approach to add more levels to it, Marvin Manheim described a hierarchical approach for the particular problem of highway route selection - and implemented a hierarchical strategy for design in a computer program [Manheim66]. Manheim's procedure incorporates two main notions:

a. The idea of refining a design progressively in steps from the level of very general plans down to the very precise level of actual construction.

b. The idea of assigning probabilistic values to plans at the high levels and particularizing those plans having the the greatest expected value.

Manheim's hierarchy consisted of the specification of several increasingly constrained areas for locating the highway with more elaborate estimations of cost as the route was more stringently constrained.

Since Manheim used a Bayesian decision theory model to guide the selection of paths, the costs of the actions even in the upper levels of the abstraction spaces had to be estimated before the program could decide which alternatives in the design to pursue. It was a weakness of the procedure that these distributions had to be estimated by the user, a highway engineer, although it is possible that other methods of estimation would have proved satisfactory.

III.2.4.4 Criticality Levels as Abstraction Levels

The technique of using a hierarchy of abstraction levels has been pursued in domains related to robot planning in the ABSTRIPS [Sacerdoti73] system developed at Stanford Research Institute. The process of abstraction used extends the methods described above in that it is domain independent.

In the robotics systems, the abstraction spaces differ from the original or ground level space only in the level of detail used to specify the preconditions of operators. At each stage of a developing plan, only those operators of sufficient significance need to be considered; operators which achieve only details are simply ignored. This approach makes the mapping of solutions from the higher abstraction spaces toward ground level very straightforward. In ABSTRIPS, the preconditions for the operators are assigned criticality levels.[12] By ranking some of the preconditions as details, ABSTRIPS is essentially capable of taking big steps in developing a "length first" plan. The planning process at each criticality level is completed all the way to an abstraction of the goal state before dropping to the next lower abstraction level.

The appropriateness of any assignment of predicates to criticality levels is reflected directly by program performance. In particular, a good assignment can be characterized by a minimum of backtracking during the refinement process. The importance of this assignment brings attention to the practical problem of determining these values. The ABSTRIPS system started with a user supplied partial ordering of predicates, but reserved the right to boost the criticality value for a literal if no short plan could be found to establish a goal value for it. In Sacerdoti's subsequent project, the NOAH system, planning is done in a hierarchical approach as in ABSTRIPS except that the hierarchy is determined by the calling structure in the SOUP code of the system, that is, it is expressed procedurally and is fixed in the system.

---

[12] It is worth comparing this idea to the notion of operator selection used in GPS where at any stage of plan formation, an ordering is used to determine which differences and operators to consider next. GPS remains limited to seeing only one step ahead, even if that step is a mere detail.

III.2.4.5 Overview of Hierarchical Planning

Section III.2.4 began with Minsky's numerical formulation of planning islands which demonstrated the combinatoric significance of abstraction but yielded no hints for practical application. Newell and Simon's abstract planning idea from GPS then gave us an example from theorem proving and illuminated the relation between planning islands and abstraction spaces. Finally, Sacerdoti's hierarchical planning idea provided an approach for establishing a hierarchy of abstract planning levels applicable in a broad class of domains.

There remain two unexplored aspects of planning with criticality levels in a complex domain. Firstly, interactions in a complex domain are likely to be subtle so that the assignment of predicates in the operators to criticality levels cannot be done simply by inspection. Any automatic approach for making this assignment could prove interesting.

Secondly, it may be possible to relax the notion of strictly additive refinement somewhat. In Sacerdotis approach, each successive refinement may add details to the existing solution. In analogy with the near misses of Winston[13], a more general approach to refinement might permit subtracting of part of a design proposed by an earlier more abstract guess. This can be illustrated by an example from molecular genetics. The restriction enzyme, Eco. R1, which cleaves DNA at a particular restriction site has been mentioned already in Section II.1. Suppose that the experiment under consideration requires as an initial step the isolation of two genes which are rather distant from each other on a bacterial chromosome. A hierarchical approach to this might permit reasoning about the use of an abstract restriction enzyme to cleave the DNA and postpone the choice of a particular restriction enzyme. The abstract restriction enzyme could be assumed to simply split the bacterial chromosome in the center between the two genes and designing could continue to later parts of the experiment. Later when this step is refined, it is unlikely that any particular restriction enzyme suitable for separating the two genes will be found to split the chromosome in precisely this way. It may cut a little closer to one gene than the other or even remove some of the material between the genes. Thus the refinement process must technically undo some of the state predicted by the abstraction. Alternatively, this example may be seen as involving a refinement of a positional specification. This notion of refinement involves a flexibility in the representation of a world state in addition to the assignment of criticality levels. A number of difficulties arising from this relaxed form of hierarchical abstraction will be discussed in Chapter V.

III.2.5 Interacting Goals

The powerful notion of hierarchical planning described above involves judicious factoring of a large problem into independent subproblems. The question arises whether it can still be applied if the subproblems are not quite independent. Even very casual observation of human problem solving behavior shows that people plan ahead without the ability to foresee that their subproblems may interact. A shopper in an

---

[13] See [Winston70]

unfamiliar supermarket who needs several items would certainly pick up item B if he unexpectedly passes it enroute to item A. Thus the shopper can re-order his activities and take advantage of any surprises while he shops.

The type of interaction in plans that has been studied the most is in the interactions between conjunctive goals. For example, if the goal of a plan is to achieve both A and B, achieving one of these goals may easily affect the achievement of the other. Problems in Blocks World have proved rich enough to explore these interactions and the following problem is probably the simplest instructive example.

Initial Situation Goal Situation

Figure 2. Interacting Goals

The interactive conjunction of goals is simply "A on B" and "B on C". The rule is that a block can be moved only if it has a clear top.

The problem of achieving interactive conjunctive goals appears in many types of problem solving. For example, any problem which is stated with initial and final states will have several distinct differences between these states. Reducing that set of differences may be viewed as the conjunction of reducing each of the individual differences. Even if the differences are viewed in a hierarchy, this merely postpones their inevitable appearance as the plan is refined unless the high level subproblems are strictly independent.

Returning to the sample problem above, let us see what a Means-ends analysis will do. Suppose it tries first to put A on B. After clearing A, it can place A on B. But now, in order to put B on C, B will have to be cleared - thus undoing the first goal that was achieved. The situation is even worse if the first goal tried is to place B on C.

A number of approaches to problems like this have appeared in the literature. Sussman, whose HACKER system is presented in [Sussman73], makes what he calls a linearity assumption which simply means that there is an order with which the goals can be achieved. His program then continues in a manner analogous to Sussman's own programming. It tries to create a plan; it discovers bugs; it modifies the program to fix the bug. The bug fixing knowledge is contained in a set of critics which can compare the bug with known types of problems caused by the linearity assumption, and suggest revisions to the plan. For example, HACKER has a mechanism called protection, which looks for actions that violate previously achieved goals. The fix in this case is to try to reverse the order of higher level goals. While HACKER will often

produce a correct plan eventually, it does so in some cases in a cycle of building a wrong plan, suggesting revisions with critics, and then building another possibly wrong plan. HACKER works effectively with problems which can be fixed by re-ordering the goals. The problem in Figure 2, unfortunately, is what Sussman terms an anomalous problem for which HACKER achieves a non-optimal solution.

Austin Tate has suggested that it is possible to abstract the nature of the interactions between goals and use this information to suggest new approaches to this problem. His system, called INTERPLAN, is described in [Tate74] and [Tate75] and makes use of the idea that abstractions of the interactions between goals are easier to work with than the original goals themselves. Tate finds it useful to abstract the assumed holding periods, or periods over which goals are assumed to be true. INTERPLAN analyzes the holding periods for both main goals and first level subgoals in the plan with a view toward moving them around to ease conflict situations. Moving a subgoal to an earlier part of the plan is what Tate terms PROMOTING the subgoal. INTERPLAN is capable of creating an optimal solution to the problem in figure 2 after moving subgoals around so that the holding periods of the higher level goals remain unbroken.

Probably the most satisfying approach to this problem is used in Earl Sacerdoti's NOAH system described in [Sacerdoti75a]. The key idea is that NOAH avoids the linearity assumption and considers the conjunctive goals in parallel as long as possible. Within the NOAH system, the parallel representation is achieved using Separate and Join nodes in a procedural network. Instead of using critics in Sussman's sense to fix bugs introduced by the linearity assumption, Sacerdoti uses constructive critics to create an ordering for the goals based on the interactions which are discovered. This is carried out by a resolve conflicts critic. If an action in one conjunct deletes an expression that is a precondition for a subgoal in another conjunct, then the endangered subgoal may be moved so that it is achieved before the action that would delete the subgoal. This synthesis of the best ideas from both Sussman and Tate is a very powerful mechanism for generating plans. Used in conjunction with other critics, this approach to resolving conflicts has enabled the NOAH system to tackle many problems that are quite beyond the capabilities of both HACKER and INTERPLAN.

Sacerdoti sums up the basic philosophy of NOAH in [Sacerdoti75a] as

NOAH makes no rash assumptions ...

Thus the linearity assumption in HACKER is rash because its effects must often be undone. The philosophy is continued in the way NOAH binds available objects in plans. For as long as possible, NOAH postpones binding the objects to particular places in the plans and uses formal variables. No guessing is done early to be undone later. In the end, other critics are invoked to simplify the plan and remove redundant preconditions. The basic idea of maintaining generality in planning appears to be a very important principle for the generation of plans. It is worth recalling at this point that the general problem of interactions between goals has been specialized to the problem of interactions between conjunctive goals. The algorithms described above

are not capable of dealing with interactions between disjunctive goals, for example, "A on B" or "A on C". Goals in complex environments are likely to contain complex expressions involving both conjunction and disjunction.

III.2.6 Using Existing Plans

One of the characteristics of human intelligence is the ability to use the solutions of old problems to aid in finding solutions for new problems. In the most elementary form, this involves recognizing an old problem and retrieving its solution. An approach known to make this effective is to generalize the solutions that have been found previously. For example, it is more effective to save a technique for solving a wide class of linear equations than it is to save the solution to just one equation. In many cases like this, the task of solving a particular problem from first principles is entirely equivalent to the task of finding a general solution. This idea of generalizing a solution is closely related to notions of reasoning by analogy which will be discussed below. Finally, one of the motivations for using old plans comes from the robotics research in planning and executing plans. In executing a plan, a robot may encounter situations in his world that were not anticipated or were not in its world model, eg. the path is blocked by an unexpected obstacle. The idea is to preserve as much as possible of the existing plan, to make local modifications to deal with the difficulty, and generally to avoid planning the entire problem from scratch over again with the new knowledge incorporated.

The first version of STRIPS [Fikes71] used a combination of theorem proving methods and Means-ends analysis. Within a given world model, resolution-based theorem proving was used to decide whether operators were applicable and whether goals had been satisfied. For the actual choosing of operators and searching through the world models, STRIPS used means-ends analysis. In 1972, the MACROP feature was added to STRIPS to increase its problem solving power (See [Fikes72b]) by enabling STRIPS to generalize and save solutions to problems. A saved solution or macro action could then be used as a single component of a new plan to solve a new and typically larger problem.

A major new feature of the MACROP addition to STRIPS was the capability to generalize plans. The following simple two step plan for achieving the goal of locating a box within a room will be used to illustrate the process.

Go through DOOR1 from ROOM1 into ROOM2.

Push BOX1 through DOOR1 from ROOM2 into ROOM1.

The immediate impression from a plan like this is that it could be generalized so that it does not mention specific objects. Unfortunately, the simple idea of replacing each unique constant by a parameter (eg. DOOR1 by anydoor1) is not sufficient. In the first place, this approach doesn't always produce the most general plan. For example, the basic plan above would still be valid if the robot started from a room distinct from the one into which he pushed the box. In the

second place, some operators have restrictions on their applicability to objects. The procedure that STRIPS uses for generalizing plans is a domain independent manipulation of the old plan. First, constants which are preconditions for any operator are replaced by distinct parameters every time they appear. Then STRIPS constructs proofs and resolves the clauses in the plan using the proofs of satisfiability in the original specific plan as a model. At the end of this process, constraints which appear while substituting parameters for constants in the new proof act as constraints in developing the more general plan. For example, the sample plan above would be constrained so that the GO operator takes the robot into the room where the box is. At this point, some excessively general steps may remain in the plan. For example, if the two clauses INROOM(R1) and INROOM(R2) were produced as preconditions for the plan, R1 would be bound to R2 to prevent the plan from appearing to accept situations where the robot was nonsensically in more than one location initially. These over generalizations correspond to those cases where two parameters are produced from a single occurrence of a constant from a single clause. Such parameters are bound together. Finally, steps whose outcome in the generalized plan now depend on a unique assignment of parameters are modified to check for this condition. For example, the plan

Push BOX1 to LOCATION1.

Push BOX2 to LOCATION2.

depends on BOX1 being distinct from BOX2. A check for this condition is added to appropriate steps from the original plan.

The second phase in the use of generalized plans by STRIPS is the monitoring of the execution of plans. Much of this work is contained in the PLANEX algorithm which makes use of a special data structure, the "triangle table", to keep track of the effects of each operator used in a macro plan on the changing world state. Considerable emphasis is placed on efficiently finding the longest applicable "tail" or final N steps of a plan. The motivation for this emphasis derives from problems encountered during actual execution of robot plans. Often, when execution fails and replanning is necessary, it is sufficient to introduce a short sequence of operators to fix the problem thus forming a plan by appending these operators to an appropriate tail. The two capabilities of the MACROP feature, generalizing plans to save as macro plans and then using these generalized plans or parts of them to solve bigger problems substantially increased the problem solving range of the STRIPS system.

The idea of generalizing a plan used in STRIPS may be cast as one form of reasoning by analogy. Generalizing involves finding a solution which can encompass as special cases more than one specific plan. Analogy covers a broader range of techniques in using a known solution to assist in finding another. The first computer-oriented research in analogical reasoning was reported by Tom Evans in [Evans68]. Evans created a system, termed ANALOGY, which successfully worked problems from the widely used Miller Analogies Test. This exam presents each examinee with a pair of figures, A and B, for which some relation holds, a third figure C which corresponds to A, and a set of five potential answer figures. The question is invariably phrased as "A is

to B as C is to ..." . The computational task may be seen as exploring a space of possible analogies and picking the one which is in some way the best. Much of the computational work in Evans' program is devoted to the pattern-recognition aspects of processing the line drawings to identify parts of the figures. Analogies are generated which consist of a number of operations used in the tests, for example adding or deleting objects, rotation, reflection, and such. Rule strengths, associated with each of the candidate operations in an analogy, are used to rank potential analogies generated by the program. The particular ranking used appears to be fairly specific to the analogy tests. Finally, after the best analogical relationship is found, the ANALOGY program is finished and does not use the relationship for any further problem solving.

In 1971, Robert Kling reported a system, ZORBA-I, which used a notion of analogy to improve the performance of a problem solving system. (See [Kling71b]). After recognizing a variety of techniques in problem solving which have gone under the general rubric of reasoning by analogy, Kling directed his efforts to adding one type of such reasoning to QA3, an existing resolution-based problem solving system. ZORBA-I accepted two theorems, T_1, a theorem with a known proof, and T_A, an allegedly analogous theorem whose proof was sought. Kling's approach was based on two fundamental ideas.

1. That the proof for T_A could be expedited if the data base was limited to those datums most likely to be relevant in the proof. This limitation was to prevent excessive floundering among irrelevant inferences from irrelevant axioms.

2. That the subgoals or lemmas used in the proof of T_1 could be used to provide planning islands in the proof of T_A.

To carry out the second idea, ZORBA-I creates an analogy which consists of a one-to-one mapping of predicates appearing in the proof of T_1 to those appearing in T_A and a one-many mapping between the axioms used in the proof of T_1 and the limited data base for proving T_A. ZORBA-I permits a user to supply a semantic template for each predicate which is used to help constrain the predicate mappings to more meaningful ones. Kling distinguishes between a complete analogy which includes all the predicates and axioms appearing in the proof of T_1 from a partial analogy which contains only some of them. ZORBA-I develops a sequence of partial analogies that terminate in a complete analogy by successive extensions and a heuristically guided clause matching process. Kling's particular approach to analogy was heavily influenced by the kinds of information that can be incorporated by a resolution-based theorem proving system. Although ZORBA-I communicated with its theorem prover strictly via a modified data base, this proved to be a powerful enough approach to allow the system to tackle a variety of problems, particularly from algebra, which had previously been beyond the capabilities of QA3.

At the present time there remain two rather fundamental problems that are left concerning the use of generalization and reasoning by analogy. The first problem is as follows.

When is a plan worth saving?

If every plan is saved, the system must face a continuously increasing repertoire of stored plans. The second problem is related.

Given a new problem to solve, how can the system find a previously solved analogous problem ?

Answers for both questions may depend on mechanisms for classifying plans and problems. The two questions require the inverse operations of saving and retrieving for plans which have been classified by applicabilities and costs. Effective classification systems are generally based on hierarchy. In addition, plans may well be saved with hierarchical suppression of particular plan details. A plan for a robot solving problems in a number of interconnected rooms may well suppress any details about opening doors if opening a door can be viewed as a trivial subproblem. These ideas are among those to be explored the next section.

III.3 Summary of Planning Ideas

To conclude Chapter III on planning, it is worth looking back over the fundamental ideas and reviewing what is known and where further research is needed. We started with two classic views of problem solving - heuristic search and theorem proving. The frame problem highlights much of the awkwardness of a purely theorem proving approach and in fact the combinatorics that come into play when theorem proving is insufficiently directed can leave a system floundering about proving irrelevant consequences. Heuristic search, whose philosophy is based on the use of domain specific knowledge for guiding this process, is too general a notion to provide any deep insights. Means-ends analysis was the first example of a forward planning formalism which offered a great deal of flexibility in problem solving behavior at various distances from the goal. It and other one step at a time systems, however, suffer the combinatoric consequences of exponential worst case behavior. This leads us to problem reduction with the conventional wisdom of divide and conquer. The logical extension of this, hierarchical planning, can cut down search by a fractional exponent, but leaves us with the technical problem of determining the appropriate abstraction spaces for a domain and for finding the mappings from the abstraction spaces into the original problem space. The criticality level idea of making the abstraction spaces correspond to successive levels of detail in the domain was a first attempt at defining a domain independent notion of abstraction. This transforms the designer's search for suitable abstraction spaces into a search for appropriate criticality level assignments to the predicates describing the ground level transformations in the problem domain. When we ask how knowledge of the problem domain can be used to determine these assignments we have left planning questions behind and have entered the area of knowledge base questions.

The other planning topics from above also lead directly into questions in the area of knowledge based systems. We considered the problems of interacting goals and discussed the known solution to the problem of ordering interacting conjunctive goals. Disjunctive goals

apparently require new mechanisms. We also discussed the STRIPS technique for generalizing and saving plans and the ZORBA-I technique for reasoning by analogy. This left us with two fundamental questions:

When is a plan worth saving ?

How can a system find a known solution to use for analogical reasoning?

More generally, we are left asking how a problem solving system should know what use to make of any transformation in the problem domain - be it a previously solved plan or an elementary transformation. This leads to more knowledge base questions:

How should plans be represented?

How can plans be classified?

How should the applicability or feasibility of transformations be represented? How can this be made flexible to accommodate changes in the knowledge base?

How can a system acquire strategy knowledge to guide the problem solving process?

How should the knowledge be structured so that it can be explained?

How can the system assist a domain expert in structuring knowledge?

The practical application of the planning ideas that we have discussed requires answers about the knowledge base itself and leads us directly to Chapter IV.

Chapter IV

Knowledge Based Systems

Several artificial intelligence research projects have been developed to meet the needs of an application as well as to satisfy theoretical computer science interests. Researchers with such projects have often tended to view computer science as an empirical science. The design and theoretical work in such systems is motivated and enriched from the encounter with the practical difficulties of creating computer systems having capabilities for managing a knowledge base which is large enough and flexible enough to meet the needs of the application. Thus, a discussion of these capabilities is the starting point for this chapter. Section IV.1 proposes some capabilities for MOLGEN which have been achieved separately in various other systems. Attention is focused on two of the underlying aspects of the capabilities which are discussed in detail in Section IV.2 and Section IV.3. Chapter V continues the discussion by proposing an architecture for a system based on these principles.

IV.1 Capabilities for a Knowledge based System

In setting forth a set of capabilities for a knowledge based system, one is reminded of the story of a family approaching an architect to design a house for them. The architect may start out by asking them what they have in mind, and the answer, if the story could be abbreviated, is everything. The house should be small, and yet have rooms for many purposes. The front room should be large for parties, yet cozy for a small group. And of course, as the architect discovers (and what is in practice the first question) the house should not be too expensive. Even if the architect can work out compromises and stay within the budget, his design may be obliterated and his estimates thrown off if the family is allowed to suggest too many minor modifications during the actual construction.

Since the design of a system depends directly on its desired capabilities, and resources are too limited to try to achieve everything, it would be useful to outline at this time some proposed capabilities for the MOLGEN system.

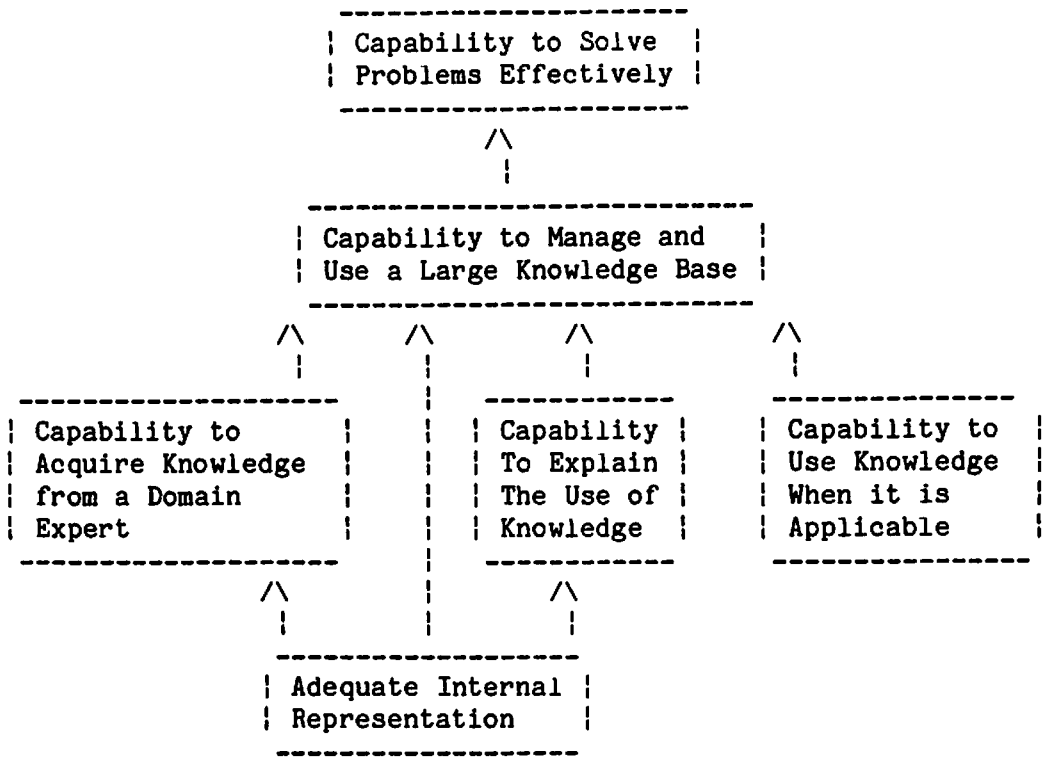

Figure 3

Chapter II has already explained the scope of the problem solving aspect of this project and Chapter III reviewed the state of the art in problem solving. It is well known that effective problem solving in a sophisticated domain rests on a large knowledge base[1]. In the MOLGEN case, this knowledge base will include the information about the objects and actions of molecular genetics and strategies for designing experiments. Management of the knowledge base includes the ability to acquire and integrate new knowledge from an expert, to modify existing knowledge, and to provide an organization of the knowledge that facilitates competent use.

The development of a large knowledge base for MOLGEN will require some expeditious means for incorporating the knowledge of molecular genetics into the computer. The transfer of knowledge from a domain expert may be seen as a pair of translations as follows:

```
[Mental Knowledge] -> [External Form] -> [Internal Form]
Human Memory          What is written    Computer Memory
```

Although many early systems included a programmer in this chain, this is an incumbrance we are seeking to avoid. The first translation in the diagram above is carried out by the domain expert. The difficulty of

---

[1] It may be noted that the knowledge base we have in mind is large when compared to some early problem solving programs, but small when compared to the size of some current data bases. Section IV.3 discusses these relative sizes.

the translation depends on what has been described as the conceptual distance between his mental form and the external form. An important objective of the design of the external form is to provide a structure which parallels the conceptual model of the domain expert. In the MOLGEN case, we believe that the characterization of the knowledge as objects, actions, and strategies reflects this design objective.

The state of the art in knowledge acquisition techniques is still a long way from being able to reduce the conceptual distance substantially in a general way. A complete reduction might involve entering directly the text, graphs, charts, and photographs from a technical journal in the domain - for example, The Journal of Molecular Biology. Other than the obvious technical problems of providing for multiple media and the inadequate state of natural language and visual processing techniques, there are two fundamental issues. The first is that these journals assume a reader has a level of technical competence and an inference capability. A novice in molecular genetics may miss an important point simply because he fails to deduce some unwritten result implied by the information in the journal. Brevity requires that the "obvious" things be left unsaid. Other assumptions are unwritten because the author does not realise that he is making them. The second fundamental problem is the integration of the new information with the rest of what is known about the domain. It is not enough to just know a set of formulated facts. In order to integrate the new knowledge, one needs to know how information is to be used and when it is important. The task of organizing knowledge automatically [2] is far beyond current capabilities. The creation of a system where a user can specify information in a flexible way -- expecting the system to use it effectively -- is at the state of the art.

Recent work has made tentative steps in addressing these fundamental problems. Part of the minimal technical competence problem is alleviated by having a model for the knowledge that is expected. In Schank's work reported in [Schank76], a story understanding program uses prefabricated scripts to fill in the unspecified elements of a story. Within the context of knowledge acquisition as reported in [Davis76c], the TEIRESIAS system builds models of domain knowledge from its current knowledge base to create expectations about new instances of knowledge. This work is described in Section IV.3.

Explanation systems need the ability to reverse the translations of knowledge acquisition systems. Explanation systems may be used for several different purposes such as:

1. Maintaining the trust and credibility of the user when the system acts in the role of a consultant. [3]

2. Providing user/system interaction during the problem solving process.

3. Informing a novice of the relevant domain knowledge for solving a particular problem.

4. Providing part of a knowledge base debugging tool.

---

[2] See [McDermott74] for a discussion of some capabilities.

[3] See [Shortliffe76], [Sacerdoti75b], or [Deutsch75].

Explanation should be geared to the expertise and purpose of the user. The TEIRESIAS system mentioned above has a measure of the difficulty of steps in a deduction so that it can accommodate either an expert or a beginner with its explanations. Since we are including strategies in the knowledge base, we must be able to explain not only the genetics objects and actions but also the system's plans and intentions. Many of the the genetic processes and strategies which the MOLGEN system will be asked to explain will require significant innovations in knowledge explanation systems.

Several representational methods are being investigated by artificial intelligence researchers at the present time. Davis and King presented a good overview of the power and applications of production systems in [Davis76a]. Woods discussed some of the foundations for semantic networks and common misconceptions involving their use in [Woods75]. Hendrix suggested some means for coping with fundamental difficulties for expressing quantification in semantic networks in [Hendrix75]. An ambitious proposal for a general system for representing knowledge is being developed by Bobrow and Winograd as reported in [Bobrow77a].[4]

Knowledge acquisition and explanation capabilities, as discussed above are necessary components of a knowledge base management system which is able to:

(1) Provide knowledge aggregation mechanisms so that the right knowledge can be applied at the right time. (Knowledge is aggregated in that it is found and brought together.) [Section IV.2]

(2) Provide for extensibility and addition of knowledge so that new knowledge and new types of knowledge can be integrated into the system. [Section IV.3]

These two issues are discussed in detail in the following two sections as noted. Then Chapter V will outline a proposed overall design for the MOLGEN knowledge base.

IV.2 Design Principles for Knowledge Aggregation

Historically the knowledge used by artificial intelligence programs has been embedded within the procedures which used them. Practically every large program was divided into a few large sections and the organization of the knowledge base followed the same divisions. Knowledge which was used together was aggregated into the data structures available to the procedure which used it. Since procedures had access to fixed sets of knowledge the aggregation was permanent. Deciding what kinds of knowledge were potentially relevant was not part of the computational task. For example, an early version of the DENDRAL program was divided into a preliminary inference maker, a data adjuster, a structure generator, a predictor, and an evaluation

---

[4] See Section IV.3.3.1 and Section IV.2.3.

function. Each of these made use of its own sets of knowledge. When programming modules contain the domain knowledge base, the modules are sometimes termed Knowledge Sources (KSs). In this organization the modularity of the knowledge base follows the modularity of the procedures which use it.

In contrast with this is the organization of several question answering systems already mentioned in Section III.1.2 which used a uniform organization of knowledge as theorems. These efforts were aimed at the creation of systems which could accept an arbitrary new body of knowledge about a domain and use a standard set of reasoning methods to do problem solving. These systems seem to have suffered from the opposite extreme of the rigid procedurally modular systems in that they have lacked adequate means for focusing on subsets of the knowledge base.

Both views of organization have established important principles of design. A static division of a knowledge base into clusters of strongly interacting knowledge, reminiscent of Simon's nearly decomposable systems, is in accord with the common wisdom that facts which are used together should be grouped together. Such systems may be realized within different representational methodologies. For example, the modules may be the top few branches in a hierarchically organized system or they may form a set of permanent clusters or partitions within a network of knowledge associations. The conflicting common wisdom from theorem proving is that a system should consist of a large number of smaller facts which can be utilized in some uniform fashion. The argument for the second view is that facts need to be used in different contexts and that a system with fixed prior groupings of facts will be unable to use what it knows when faced with a new context. Winograd summarized this conflict in [Winograd75b] as follows:

> ... we must keep an eye on both sides of the duality -- we must worry about finding the right decomposition to reduce the apparent complexity, but we must also remember that interactions among subsystems are weak but not negligible. In representational terms, this forces us to have representations which facilitate the "weak interactions".

While weak and strong interactions have been discussed in [Simon69] and [Winograd75b] and it is clear that knowledge which interacts must be aggregated in some manner, two questions remain to be clarified:

1. What are the criteria for distinguishing weak and strong interactions?

2. What mechanisms can be used for finding knowledge which is weakly interacting, that is, how can weakly interacting knowledge be aggregated?

IV.2.1 Criteria for Weak and Strong Interactions

We offer the following definitions of strong and weak interactions:

1. Knowledge is strongly interacting if it should be used together in all problem solving contexts.

2. Knowledge is weakly interacting if it should be aggregated differently depending on context.

Thus, a partitioning is inadequate for facilitating the weak interactions when it is fixed and completely independent of context. Strongly interacting knowledge should be aggregated in a permanent context independent manner and weakly interacting knowledge should be aggregated in a temporary context dependent manner. As is explained in detail below, we suggest that permanent links be established between units of strongly interacting knowledge, while temporary links for weak interactions should be established by pattern matching.

Methods of aggregating strongly interacting knowledge will be explored in detail in Section IV.2.3 and Section IV.3.3.1. In this section, we will be concerned with how temporary links are formed to facilitate the temporary interactions between knowledge sources.

In a system with many units of knowledge, most of the interactions will be context dependent. We contend that many of the knowledge sources must be activated according to the problem solving context of the system. With undecomposable knowledge sources, the only means to express weak interactions is by controlling access to the individual knowledge sources. Thus we must pay attention to the alternatives for the creation of the links for context dependent interactions. The temporary links for context dependent interactions can be established in two ways. (1) By using knowledge sources as the primary index one can establish links to relevant problem solving contexts. Demons embody this approach by using pattern matching to recognize the context. The link is made when the knowledge source becomes active. Alternatively, (2) problem solving contexts can be used as the primary index with links being established to the relevant knowledge sources. An example of this approach is given from the TEIRESIAS system in Section IV.2.3. Again pattern matching can be used to establish the temporary link. The difference lies in the location and nature of the pattern. In (1) knowledge sources are activated when they recognize a context. This idea is discussed in the next section. Alternatively in (2), contexts can have mechanisms for selecting or aggregating knowledge sources according to the patterns in the knowledge sources. This alternate approach is discussed in Section IV.2.3. Between these two sections, Section IV.2.2.1 presents some of the methodology and ideas which have evolved in representing domains as communities of experts. In such frameworks, the knowledge within the experts is strongly interacting and communication between the experts facilitates the weak interactions. Some of the research in this area started with the viewpoint in (1) above but has evolved to motivate the ideas of (2).

To avoid losing the main ideas while exploring the side issues and history of the ideas in what follows, the main points together with the sections in which they are discussed are listed here.

1. The early designs for artificial intelligence programs involved embedding the domain knowledge inside procedures for using it. Modularity of domain knowledge followed the modularity of the procedures. This methodology introduced the idea of knowledge sources in programs. [Section IV.2.1]

2. Interactions between knowledge have been characterized in the literature as being weak or strong. Weak interactions are those which are temporary and context dependent; strong interactions are permanent and context independent. [Section IV.2.1]

3. The mechanisms for facilitating weak interactions play an important role in the integration of new knowledge in a system. Such facilities are expected to find and apply knowledge sources in a system in those problem solving contexts where they are relevant. [Section IV.2.1]

4. The basic mechanism for facilitating weak interactions is pattern matching. Knowledge sources can use patterns to recognize contexts (as with demons) or contexts can use patterns to recognize knowledge sources. [Section IV.2.1]

5. Demons are a useful approach to organizing knowledge sources when the contexts in which they can be applied are diverse but easy to recognize. Such knowledge sources are said to be event driven. [Section IV.2.2]

6. Demons should not be used as the sole mechanism for implementing weak interactions since they do not provide <u>coordination</u> for those weak interactions involving <u>multiple</u> knowledge sources. These interactions can be facilitated by attaching a pattern of the knowledge sources and coordination information to an agent of the problem solving context. [Section IV.2.2]

7. In the TEIRESIAS system, meta-rules act as agents for the problem solving context. These strategy knowledge sources contain patterns which can be matched against the domain knowledge to find knowledge relevant to the current problem solving context. The object level rules in this system are the right decomposition of domain knowledge into permanent context independent chunks and the meta-rules express and coordinate the temporary context dependent interactions between them. [Section IV.2.3]

8. Systems based on this methodology have developed in

the direction of using smaller and simpler knowledge sources. Production rules can be used as knowledge sources for both strategy and domain knowledge. Content reference has been used as a mechanism for pattern matching by strategy rules. [Section IV.2.2.2 and Section IV.2.3]

9. Strategy knowledge sources can in theory cover a hierarchy of types of knowledge and provide a powerful and flexible representation for this knowledge. A system which actually offers this power has yet to be built and some extensions to the ideas above seem to be needed. [Section IV.2.3]

10. Pattern matching facilities based on content reference depend on the decomposability of the knowledge source. The content reference ability in existing systems match strictly according to the presence of certain tokens in the rules. For more complicated rules where the way these tokens are used has a bearing on the classification, more powerful mechanisms are required. Proposals are made to increase the expressive power of rules while providing powerful methods for classifying them. [Chapter V]

IV.2.2 Demons and the Multiple Knowledge Sources Model

Demons are procedures which are activated when some activation condition is satisfied in a data base. They are useful when a knowledge source needs to be used in a diversity of contexts which are easy to recognize. In the PLANNER language reported in [Hewitt71], these are the antecedent theorems. Whenever anything is asserted (ie. added to the data base), all antecedent theorems are checked against the new assertion. In production systems as described in [Newell73], each production can be considered to be a demon waiting for a condition so that it can fire. Bobrow and Raphael give a good overview of pattern directed invocation in programming languages in [Bobrow74].

One outgrowth of the early work in demons and pattern directed invocation was the attempt to extend this idea as far as possible. This lead to the development of a computer system composed entirely of demons acting as expert knowledge sources.

In the sections which follow, the BEINGS of Lenat and the expert KSs of the HEARSAY system will be discussed. These sections will not discuss or evaluate these systems in their entirety, but will concentrate on the approaches these systems have followed in their treatment of context dependent knowledge. The source of strength in Lenat's system, that each expert recognizes his own relevance and makes his own contribution to the problem solving without being aware of the nature of the other experts, is ultimately a source of weakness. That the experts know how to organize themselves individually is no guarantee that they can work effectively as a group. There is no specific mechanism for coordinating the activities which may compete for processor time. In the HEARSAY terminology, this is part of the focus of attention problem.

IV.2.2.1 BEINGS and ACTORS

Douglas Lenat used pattern directed invocation between procedures as knowledge sources in his concept of "Beings" reported in [Lenat75]. He has suggested that problem solving knowledge can be organized as a community of interacting modules where each module, termed a Being, implements a particular expert in a small part of the domain. As in the case of the Actors described by Hewitt, Bishop, and Steiger in [Hewitt73], this approach to organizing knowledge promotes the following design methodology:

> 1. Decide on the kinds of experts to have in the domain. Each expert corresponds to one aggregation of strongly interacting knowledge.
>
> 2. Decide for each expert what messages it should send and receive. These messages are used to form the links for weak interactions between knowledge sources.

Lenat's Beings differ from Actors in that they do not mention the name of the expert to receive a message, but rather broadcast their messages to the entire community. Each Being is responsible for recognizing and answering messages within its domain of expertise. Within its special part of the domain, each Being has a set of strategies for recognizing its relevance to any proposed question. Lenat developed the PUP6 system using Beings as a representational form as reported in [Lenat75]. PUP6 was an automatic programming system which wrote a concept formation program [5].

Most of the Beings in PUP6 were rather complicated modules. Lenat has suggested that this is appropriate since the behavior expected from Beings is complex. They required the capability to send and receive messages to achieve both the triggering and the coordination of the Beings. Communication was constrained to a set of 29 standardized questions which one expert could ask another. The vocabulary, syntax, and semantics of these questions was contained as part of the code for the Beings themselves. As new experts requiring extensions to the vocabulary were added to the system, changes were required in existing Beings.

In the terminology of the previous section, the Beings are the modules of strongly interacting knowledge around which the problem domain has been organized. The context dependent interactions are facilitated by the message communication between the Beings. It is conceivable that more than one Being would be activated by a given message. For such situations, the Beings and their messages must be carefully designed to provide a mechanism for arbitration. Putting these arbitration mechanisms in the messages between Beings is in conflict with the design goal that experts should not need to know of each other's existence. The HEARSAY system, which is discussed in the next section, offers some special mechanisms for this control problem which is part of what has been termed the focus of attention problem.

---

[5] The concept formation program which was synthesized was based on work by Winston reported in [Winston70].

IV.2.2.2 Lessons from HEARSAY

The HEARSAY speech understanding system [6] also follows the discipline of dividing the knowledge base into a set of procedural Knowledge Sources (KSs) activated by pattern directed invocation. In contrast to the systems mentioned above, HEARSAY has been under development over a period of ten years and has undergone a design review in light of this experience. The evolution of HEARSAYII from HEARSAYI illustrates some important directions in the design of knowledge bases.

HEARSAYI was designed to make use of the following diverse sources of knowledge: acoustics-phonetics, phonology, syntax, semantics, and pragmatics. As with PUP6, one of the design goals of HEARSAY has been that the experts would not need to know of each other's existence or structure. The motivation here was to provide a system where new KSs could be simply added or deleted for experiments in measuring their impact on the effectiveness of the total system and for modularity in developing the system. Because of the variable nature of the speech signal and an inadequate theory of the production of speech, the KSs are error prone and must work together cooperatively to correct each other's mistakes. Communication between KSs takes place in a dynamic global data structure, the blackboard, which contains the current state of the world. This consists of a set of hypotheses or partial hypotheses at the word level of recognized speech. Each KS may access the blackboard to create, delete, or modify hypotheses. In HEARSAYI the KSs are activated in a lockstep sequence of poll, hypothesize, and test. The poll phase determines which KSs have something to contribute, the hypothesize phase activates the KS showing the greatest confidence about its contribution, and the test phase consists in having all the KSs evaluate the hypothesis.

Many of the design decisions in HEARSAYI which have come under review are of general interest in the design of knowledge bases. First, the limitation of the blackboard and hypothesize and test paradigm to hypotheses at the word level in HEARSAYI has proved too restricting. HEARSAYII uses a blackboard partitioned into seven distinct information levels. The decomposition of the blackboard and problem space into discrete levels makes it possible to decompose the KSs more finely. In the terminology of Section IV.2.1, we would say that too much information had been aggregated in the KSs and that in HEARSAYII they were decomposed into smaller modules which could interact in a context dependent manner. Experience has shown that most KSs need to work with only one or two levels so that they can be as simple in structure as their knowledge permits. Secondly, the lockstep control sequence of HEARSAYI for the hypothesize and test paradigm inhibits interaction between processes resulting in repeated computations and blocked parallelism. HEARSAYII replaced the sequential control sequence with pattern directed invocation so that a KS could be activated when the blackboard contained information satisfying a precondition of a KS. In this framework the KSs may be viewed as production systems where the precondition corresponds to the condition on the left hand side and the KS corresponds to the action on the right hand side.

---

[6] See [Erman76] and [Hayes-Roth76] for some recent articles about this system.

In summary, the evolution of HEARSAYII from HEARSAYI involved the following important changes in design.

(1) A decomposition of the KSs of strongly interacting knowledge into smaller, simpler units which can interact in a context dependent manner.

(2) A decomposition of the blackboard into more levels. This facilitates (1) above.

(3) The blackboard was extended to show relationships between hypotheses including support and structural relationships. This made it possible to express the contexts for the weak interactions between knowledge sources. In HEARSAY terminology, this allowed the sharing of partial hypotheses between KSs.

What remains to be discussed about HEARSAYII is the mechanism for coordination of the KSs. The coordination problem in HEARSAYII is termed the focus of attention problem and has two components:

1. Choice of a partial hypothesis (HEARSAY's meaning for context) in the problem space for attention.

2. Choice of a Knowledge Source to use within this context.

Associated with each hypothesis are indicators telling how much computational effort has been expended so far as well as combined estimates from the KSs of the desirability of allocating more. These indicators are used to direct the first part of the focus of attention problem -- the selection of context in the problem space. For the second aspect of focusing, the selection of a KS, HEARSAYII takes advantage of the production rule view of the KSs. Each plausible KS is asked to evaluate its preconditions and to estimate its applicability. Frederick Hayes-Roth and Victor Lesser have suggested several fundamental principles for rating KSs in [Hayes-Roth76]. For example, KSs may be favored which promise a best outcome, or which have the most valid data to work from, or which are the least expensive or most reliable.

We have seen that HEARSAYII has provided a focus of attention module with the ability to choose among competing knowledge sources for allocation of computational resources. The next section generalizes this idea by (1) using a number of context dependent strategy KSs instead of just one focus of attention module, and (2) by applying a pattern matching facility to the KS itself instead of to an abstraction of it. We will see that the success of this approach depends on continuing the trend toward small and simple KSs.

IV.2.3 Knowledge Access and Control by Description

TEIRESIAS [7], is a system which contains some interesting innovations in the use of context information for structuring

---

[7] See [Davis76c].

knowledge. In this section we will be concerned with TEIRESIAS's treatment of context dependent interactions. In Section IV.3.3 we will return to this system in our discussion of knowledge acquisition. TEIRESIAS was developed in collaboration with and integrated into the MYCIN system for medical consultation. The MYCIN system includes a knowledge base of approximately four hundred production rules. These production rules are the Knowledge Sources (KSs) within the MYCIN/TEIRESIAS system. An example of a production rule follows:

If 1) the morphology of ORGANISM-1 is rod
   2) the gram stain of ORGANISM-1 is gramnegative
   3) the aerobicity of ORGANISM-1 is facultative
   4) the infection with ORGANISM-1 was acquired
      while the patient was hospitalized

Then there is suggestive evidence (.7) that the
     category of ORGANISM-1 is enterobacteriaceae.

The MYCIN system conducts a medical consultation by evaluating in depth first order an AND/OR tree formed by these production rules. As of June 1976, the largest number of rules relevant to any one goal was forty. At that stage exhaustive invocation was still computationally feasible. In response to an expected continuing growth of the knowledge base, a mechanism for guiding the selection process using meta-rules was developed.

The meta-rule approach developed by Davis involved augmenting the rule syntax above with new meta-level (strategy) primitives to provide a language for strategy. The following is an example of a meta-rule in the TEIRESIAS system.

If 1) the infection is pelvic-abscess and
   2) there are rules which mention in their premise
      enterobacteriaceae and
   3) there are rules which mention in their premise grampositive rods,

Then there is suggestive evidence (.4) that the former should be
     done before the latter.

In this example, the first clause about pelvic-abscess defines the context. The second and third clauses contain patterns which are matched against the domain rules in the knowledge base. A domain rule will match if it mentions enterobacteriaceae or grampositive rods in its premise. The current implementation of meta-rules in TEIRESIAS supports two kinds of statements. Meta-rules can make statements about the likely utility of other rules and they can also impose a partial ordering on the evaluation of other rules. This partial ordering is in the same spirit as the allocation of processor power in HEARSAY. The same principles for choosing between KSs discussed in [Hayes-Roth76] can be implemented within production rules. It is interesting to return to Winograd's suggestion as quoted in Section IV.2.1. The object level rules in this case are the right decomposition of knowledge in the domain into permanent context independent chunks and the meta-level rules express and coordinate the temporary context dependent weak interactions between them. Thus the various premises and

actions of the object rules are permanently wired together while a meta-rule indicates interactions between groups of rules in order to coordinate their use. Thus, weak interaction is keyed by the context described in the meta-rule.

In the strategy rules above, we see that the pattern, instead of being associated with the object rules, can be contained within the strategy rules. The pattern is the argument to the "mentions" function. Much of the motivation or the use of pattern matching in TEIRESIAS to establish the context dependent links is the same as that in HEARSAY or PUP6. Use of pattern matching to find knowledge sources is the mechanism that guarantees that as new knowledge sources are added to the system, they will be automatically applied in those contexts in which they are relevant. Davis refers to this matching process as reference by description and distinguishes between two broad approaches: (1) External Descriptors and (2) Content Reference. The external descriptors approach consists of a methodology where a number of different characteristics are chosen and each KS is described in terms of them. For a procedure this could include such things as the procedure's main effect or its preconditions. The second approach is by direct examination of KS content. The meta-rules above have the ability to examine the characteristics of object level rules. The advantages of the second approach derive from the ease with which new knowledge and strategies may be incorporated into a system.

TEIRESIAS decomposes the process of applying object level rules in their corresponding contexts into two steps. First, pattern matching is used to create sets of rules for each of MYCIN/TEIRESIAS's contexts. In this system, there is a separate possible context for each object that a rule may conclude about. These sets correspond to permanent aggregations of knowledge discounting changes to the knowledge base. Then MYCIN accesses these sets of rules as it traverses its context tree. The meta-rules express temporary interactions between these sets. This approach mixes the two types of referencing mentioned above. It allows the prior computation of external descriptors while preserving the flexible strategies and ease of adding new rules to the system characteristic of the content reference approach. It should be noted that in many systems, the number of problem solving contexts would be too numerous to make this complete grouping of rules feasible.

Important design considerations for KSs to permit reference by content are

1. The contents of KSs should be accessible (addressable).

2. KSs should be simple (or at least regular) in structure.

A precise meaning for the notion of structural simplicity has not yet been worked out nor has much work been done to clarify the trade-off between simplicity and expressive power. It is known, for example, that expression of any form of iteration is awkward and generally difficult to recognize in typical production systems. Another data point on the simplicity vs expressive power scale follows from Sacerdoti's work in the NOAH system. The add and delete lists associated with each action are used to represent the effects of an action for purposes of global

selection of actions. The system depended on the programmer to pick the right actions to represent in these lists while various smaller subactions were represented only in the QLISP code. Sacerdoti suggested that the QLISP modules were not simple enough for inspection by the system. In any case, the system did not know enough to carry out a meaningful inspection. In Chapter V a technique will be suggested for acquisition and management of extended rules which are more powerful than these production rules and more restricted than QLISP.

A second observation about the structuring of knowledge sources, strategy knowledge sources or meta-rules in particular, is that they probably need more powerful mechanisms for pattern matching than those that were used in TEIRESIAS. Davis suggested that meta-rules can be extended through an arbitrary number of levels. Thus the first level strategies expressed in meta-rules direct the use of object level knowledge, second order strategies (meta-meta-rules) direct the use of strategies and so on. Although TEIRESIAS was programmed to accept meta-rules of arbitrary order, the medical domain in which the system was tested offered no instance of a rule of greater than first order. Davis gives a mathematical treatment of meta-rules suggesting that they can reduce evaluation work by an exponential factor. There is, however, a sleeper in the argument. Recall that the main content referencing mechanism in TEIRESIAS' current meta-rule implementation is the "mentions" function. This function examines premises and actions of rules for the existence of particular tokens. Unless there are particular tokens used in meta-rules distinct from those in object rules, meta-meta-rules can only ask about the same tokens again. One can imagine expressions about mentionings becoming awkwardly large and complex. Davis hints at a fix for this problem in the context of a poker playing example.

> To win at Poker,
> first try bluffing,
> then try drawing three cards,
> finally try cheating.

A rewritten version of this might be "First use any psychological ploy to discourage the competition, then try something to improve your hand, and finally do anything that will make sure you win." Each clause has been written as a more general description of the actions. This suggests that we need more powerful methods to describe rules than is currently provided by reference by content. A proposal for doing this is discussed in Chapter V.

IV.2.4 What We Have Learned

Having completed our survey of knowledge base interactions, let us summarize it. We began with a proposed classification of interactions between chunks of the knowledge base as either weak or strong. Weak interactions were characterized as temporary and context dependent; strong interactions were characterized as being permanent and context independent. Strongly interacting knowledge should be grouped as a unit or knowledge source. Temporary links for context dependent interactions between knowledge sources can be established by pattern matching.

One approach to establishing the links for the context dependent interactions is to provide a pattern of the relevant context to the knowledge source. In this framework the knowledge sources themselves are sometimes called demons. This approach has been discussed with examples from the PUP6 system of Lenat as well as the HEARSAY system. The HEARSAY experience lead to a formalization of the focus of attention problem, which includes the coordination of multiple interacting knowledge sources which may compete for processor time in a given problem solving context.

The MYCIN/TEIRESIAS example extends this aspect of the focus of attention problem by providing multiple strategy knowledge sources termed meta-rules. A meta-rule acts as an agent of the problem solving context to coordinate the weak interactions between object level rules. The object rules in this case are the right decomposition of the domain knowledge into context independent chunks and the meta-rules express and coordinate the temporary context dependent interactions between them.

Both demons and meta-rules use a form of pattern matching for controlling the use of knowledge in different contexts. In the case of demons, the knowledge sources carry a pattern of the context in which they should be applicable. In the case of meta-rules, the strategy knowledge source associated with the context carries a pattern of the plausible domain level knowledge sources. In both cases, simplicity in the structure being matched, problem solving context or knowledge source, is thought to be an important design consideration although a definition of simplicity has not been given precise meaning.

Whenever new knowledge is entered into a system, its logical relationships to the existing knowledge must be established. We will see in the next section that a number of the ideas about descriptors which have been discussed with regard to their use in controlling the way knowledge is accessed also play a role in the way it can be acquired by a system and integrated into a knowledge base.

IV.3 Design Principles for Knowledge Acquisition

Knowledge acquisition research has taken place on three rather distinct fronts - in the area of programming languages, in database management, and in the knowledge based systems of artificial intelligence. This section examines them with three purposes in mind. First, the simple ideas have been around for quite a while and it is worth discussing them clearly so that they need not seem to be re-discovered in later contexts. Secondly, the simple ideas have rather limited power and it is important to delineate this power. Thirdly, the powerful ideas are rather subtle and involve mechanisms which may seem a bit complicated. The power and significance of these ideas is best understood by comparison to the simpler approaches.

Although the main topic for this section is knowledge acquisition, many of the ideas for organizing knowledge to facilitate acquisition are important for broader purposes in the management of a knowledge base. These points will be presented along with the the main ideas of this section.

Knowledge base and data base researchers are currently attempting to define the differences between their respective fields. There are certain obvious differences. Earlier when we stated that MOLGEN would have a large knowledge base, we pointed out that the base would still be small by data base standards. According to [Fry76], it is not unusual to find government or commercial data bases of over one billion characters. This is roughly a thousand times larger than any knowledge base used in artificial intelligence. Many other differences result from this size difference. With a huge data base, researchers must be concerned with efficient retrieval of information. The information retrieved is generally used as input to separate programs performing specific tasks such as report generation, payroll, or a display of the information for a human user. The data base contains limited knowledge about itself and its uses. In early artificial intelligence systems, the knowledge necessary to direct the problem solving was often part of the control or problem solving program. As knowledge base researchers have moved to separate data from code, they have tried to create systems which reflected the dense interconnections necessary for problem solving. Thus, knowledge bases must contain the rules of inference, corresponding to the actions and strategies discussed earlier, which provide the control information to the system. The direction of this report [8] is to include even more strategy information in the knowledge base so that the knowledge base contains the information to direct the use of knowledge in problem solving.

The differences in research orientation are tending to converge somewhat as progress is made. Some researchers have built systems integrating both knowledge bases and data bases. An example of this, the Gus system, was reported in [Bobrow77b]. GUS converses in a mixed initiative English dialog with a user about travel arrangements.

> In the travel domain, the Official Airline Guide is a data base which GUS treats as a large external formatted file. GUS can use an extract of this data base but the information in the file does not form part of its active working memory for the same reason the Official Airline Guide does not have to be memorized by a travel agent. Only that portion of the data base relevant to a particular conversation need be brought into the working memory of the system.

In GUS, the frames which drive the dialog constitute part of the knowledge base and the travel guides are part of the data base.

Research about knowledge acquisition began with the efforts in the late sixties to make programming languages more powerful by making them extensible. The idea was that a programmer could modify the language by defining entities within it that were conceptually similar to the mental structures he had for his problem. This corresponds to the later work in knowledge based systems to facilitate effective communication with an expert. Effective communication should take place in terms and concepts close to those which are in general use in the technical jargon of the domain. Much of the need for natural

---

[8] See Chapter V.

terminology derives from a desire to make use of the tried and true classifications of knowledge that have evolved in a technical area. As Thomas Cheatham remarked in [Cheatham69]

> Discussion of the motivation for extensible language rests on a basic premise, namely that there exist diverse programming language requirements which are becoming more diverse, and that it is of critical importance that each user ... be provided with a language facility appropriate to his problem area. ... A part of this premise is that it is not enough to have a language which is formally sufficient to host the particular data and unit transactions some user has in mind. Rather it is of critical importance that the kinds of data and unit transactions which he wants to think of as primitive be available, effectively as primitives, in his language facility.

Again, we list the main points together with the sections in which they are discussed.

1. Effective communication mandates the use of tried and true or natural classifications from a domain in order to reduce the conceptual distance for a person expressing domain knowledge. This motivated the development of extensible languages. [Section IV.3]

2. The first work in extensibility was done in the context of programming languages. The three components of these languages - data, operations, and control - correspond naturally to the three classes of knowledge we have discussed earlier - objects, actions, and strategy. [Section IV.3.1]

3. The main mechanism used to provide extensibility was the ability to define new (larger) entities in terms of a set of basic primitives. [Section IV.3.1]

4. Workers like Dahl or Liskov and Zilles have suggested that the new data types and the allowable operations on them be defined at one place in a cluster in order to promote abstraction for structured programming. [Section IV.3.1]

5. It was generally thought that extensibility in the programming language would result in clear and efficient programs and that these programs would be much easier to write. [Section IV.3.1]

6. The important lesson from this work was that the amount of knowledge necessary for a user to mold the nature of the system for his requirements had been seriously underestimated. The systems themselves remained too ignorant to provide much help. [Section IV.3.1]

7. Faced with the requirements of enormous data bases, data base management researchers have concentrated on increasing a system's knowledge about its data. [Section IV.3.2]

8. The main idea was to have schemata associated with the data itself to describe the logical relationships, field names, formats, and physical layout. [Section IV.3.2.1]

9. The idea of procedural attachment has appeared in the data base literature but it has not been implemented very extensively. [Section IV.3.2.1]

10. Much of the research has been in comparing three models for data organization - hierarchical, relational, and network - for their relative efficiencies and flexibilities for retrieval. [Section IV.3.2.2]

11. Some workers have suggested that type-checking assertions for operations on data can be entered as part of the data definitions. This is a step closer to the object centered factorization of knowledge ideas for knowledge based systems. [Section IV.3.2.3]

12. The schemata for data base systems were used to provide data definition capabilities for systems using a uniform mechanism for storage of values. Knowledge based systems have extended the power of schemata to organize groups of values and procedures into "conceptual objects". [Section IV.3.3.1]

13. Schemata for conceptual objects are used in knowledge based systems to guide the acquisition of new instances of objects. Schemata can be used to ensure the completeness of information about objects by guiding the acquisition process. They also can guide any necessary bookkeeping as new objects are added to the system. Procedural attachment is helpful for providing flexibility in filling out and checking the values for instances of objects. [Section IV.3.3.2]

14. Just as an object schema may guide the acquisition of a conceptual object, a "schema-schema" may be used guide the acquisition of a new schema. Using this idea a system can acquire information about new <u>kinds</u> of objects as well as new instances of objects. Thus schemata can provide a mechanism for extensibility. The essential knowledge that programming systems lacked for providing assistance in extensibility is contained in these schemata. Realization of this is one of the important contributions of knowledge base research. [Section IV.3.3.2]

15. Procedural attachment in schemata is also important for assisting the management of changes in a knowledge base so that when a change is made in one definition, other dependent definitions can be located and changed at the same time. [Section IV.3.3.2]

16. Knowledge based systems have also provided examples of the acquisition of actions. This has involved the use of rule models which correspond to schemata for actions except that the models contain information derived from examples and they facilitate only very limited structures for rules. In the MYCIN/TEIRESIAS system, rule models are derived from rules in the knowledge base. [Section IV.3.3.4]

17. Rule acquisition has used the problem solving context as well as a rule model to guide the acquisition of a new instance of an action. [Section IV.3.3.4]

18. There is room for more research on the use of schemata to support the classification and acquisition of new kinds of actions and strategies. Proposals for this work are presented. [Chapter V]

With these high points in mind, we begin with the development of extensible programming languages.

IV.3.1 Extensibility in Programming Systems

As Perlis remarked during an opening address for a SIGPLAN symposium on extensible languages[9], three things define a language: data, operations, and control. Not surprisingly, these correspond to the three classes of knowledge mentioned in Section IV.1 - objects, actions, and strategies. These three plus syntax form the axes at which development in extensible languages has taken place. One of the major efforts in extensible systems indicative of the the scope of these efforts is the ECL system reported by Ben Wegbreit in [Wegbreit71]. This system was developed to assist programmers working on projects where there is considerable interplay between design and development. ECL allowed extension of syntax for specification of new linguistic forms in terms of existing forms. It supported data type extension allowing a programmer to define new data types and information structures needed to model the task at hand. In this regard ECL supplied a number of built in types - Boolean, integer, floating point, character, symbols, and pointers - and provided mechanisms for efficient access and storage of the structures. Much of this corresponds to the record structures now available in Algol-like languages. Operator extension allowed a user to define new operations on the new data types and to extend old operations to cover the new data types. Control extension allowed the creation, deletion, and coordination of independent asynchronous processes. These extension mechanisms were sufficiently broad to cover co-routines, Dijkstra's P and V operations, multiple parallel returns, and process scheduling. The basic methodology behind all of these extensions was to provide a

---

[9] See [Perlis69].

set of primitive entities in the language. A user could then define his own higher level entities as special combinations of the basic primitives. Some extensible language facilities involved the creation of compiler-compilers and constrictors as mechanisms for keeping the flexible user-defined language economic.

Barbara Liskov and Stephen Zilles were among the proponents of extensible languages as an aid to structured programming. In [Liskov74], they emphasized the nature of user defined constructs as abstractions, that is, mechanisms for the suppression of irrelevant detail. They advocated a very restricted procedure for definition of abstract data types where the representation (for example, record structure) and operations on it (defined as unique procedures having access to the representations) were defined together in one unit termed a cluster. These user defined primitives, analogous to the familiar primitives of the base language, would be abstract entities for manipulation by the program only through the defined operations. Their internal structure would be unknown (in fact unknowable) outside of the cluster. This was thought to encourage a formulation of abstract data types that was independent of representation and was in contrast to those extensible systems where a user learns one mechanism to define the representation and another to define the operations on it. Perhaps the most widely known language which incorporates this philosophy is SIMULA with its class definitions. Although the motivations are somewhat different, the monitors discussed by Hoare[10] in operating system design reflect many of the same considerations.

It is interesting to view the changes in the ways people viewed extensible programming systems after a period of trial and experimentation. Thomas Standish, reviewing his own PPL system in [Standish71] which was one of the most successful of all the extensible language systems, termed PPL a language that failed. This was in spite of the fact that it was fully implemented, was the language of choice in Harvard's introduction to programming course, and was tested over a diversity of application areas by over 450 users. It seems to have been a case of expecting too much. As Standish remarked

> It was thought that just as programmers decree the organization of processes (by defining and calling subroutines), they should also decree appropriate organization for data and for notation, in order to attain clarity and efficiency. ... What we did not fully grasp was the amount of effort and knowledge required of a user to deform a language in significant ways.

Finally Standish summarized it all again, the frustration of expecting too much from the simple mechanisms.

> You can't state something simple to an unknowledgeable mechanical recipient and expect it to alter its behavior in major ways.

---

[10] See [Hansen73].

Perhaps the key word in this quotation is "unknowledgeable" which leads us to the efforts by researchers in artificial intelligence to make a system know what it knows.

IV.3.2 Ideas from Data Base Systems

Data base technology can be traced back to the late fifties [11] when several workers discussed the use of general routines capable of sorting files of different formats and arbitrary contents. The technology developed in response to the typical data processing operation in the late sixties where every new need for data involved writing a new program. Using existing data files for a new program generally meant that somebody had to understand the program that wrote the files because the format of the data was locked up in some combination of programs and control cards. Fry [Fry76] references a scenario where a business manager knew that data bearing on a business decision existed, but some of it had been produced on a different machine, some had incompatible formats, and the description of the logical organization of some of the data was unavailable. The manager was unable to obtain answers in a reasonable amount of time even though the data was in some sense in the system. This type of situation gave rise to the vision of a system with all of the data integrated with data definitions stored with the data and general purpose software to access and manage the data files. This type of system has been termed data base management as opposed to data management.

The rest of this section discusses the ideas from data base management most relevant to knowledge base research. We will begin with a discussion of data definitions to explore the limits of the capabilities that have been provided. We will see that different logical arrangements of the data have an impact on the accessibility of the data. Some data models are thought to result in lower sensitivity of programs to changes in the data and its definitions. Finally we will look at the work in an area on the border between knowledge bases and data bases where some additional capabilities for consistency checking in data bases have been explored.

IV.3.2.1 SCHEMATA: Data Definitions

Crucial to the capability of integrating data into a data system for uniform manipulation and centralized management is the idea of a data definition, usually termed a schema. Programming languages have traditionally provided facilities for naming and characterizing data elements within records. What is new with data bases is the idea that these schemata are outside the code of the programs and stored with the data. This creates the potential for allowing the use of generalized data base management software to manipulate the data.

Schemata are used to specify structure and interrelationships of data elements. Some of the structure specified in schemata is very similar to the information associated with the RECORD structures in ALGOL-like languages. The names of the various fields and their types (eg. integer, floating point, character) as well as length information

---

[11] See [Fry76] for early references.

may be specified. Similarly such things as hierarchies of elements (eg. BIRTHDAY as Month Day Year) as in PL/1 and variable format in terms of conditional elements or repeated groups may be specified. References (pointers or symbolic) to other elements in the data base were permitted. In addition to these application independent specifications, schemata may contain information about units of measurement or data domain classifications of the elements. Section IV.3.2.3 discusses the use of this additional information for maintaining data base integrity.

Gio Wiederhold discusses the use of procedural attachment in schemata for data bases in [Wiederhold77]. These procedures may be used to derive data when references are made to particular data. Wiederhold distinguishes two kinds of procedure activation - actual and potential. These correspond to the demons and servants respectively in [Bobrow77a].[12] Actual results are those changes to the data base which are propagated when a data element is updated. This means that the data base administrator has attached a procedure in the schema which is executed whenever a particular data element is changed. Potential results are those which are computed on request. Wiederhold discusses an example where the effect on company revenue of changing an employee's salary is computed using both approaches and makes some implementation suggestions for a practical system. An example which is less demanding computationally is one where a procedure is invoked to convert an internal binary form for a date to a symbolic form suitable for external presentation. In both of these examples, the procedure is implemented by the data base administrator and is not considered to be part of the data base. These ideas for procedural attachment have not been extensively implemented within the data base systems although they represent an important part of the research for artificial intelligence applications.

Part of the reason for using general and uniform data base management software to access the data has been the desire to create programs which are insensitive to changes in the data layout. This has been successful for the following kinds of changes: size of fields, the addition of new fields in schemata, or modifications in the physical (but not logical) layout of the data. This means that the schemata for the data are changed and the data itself is changed correspondingly but the program does not have to be changed. This facility is described as creating a measure of "independence" of data layout. Marginal independence of the logical structure of the data has been achieved but it is not yet clear how much more independence can be achieved while retaining sufficient efficiency.

Much of the debate in the choice of designs for data base centers around the choice of different data models. It is believed that the various models offer differing degrees of efficiency, flexibility, and program sensitivity to changes in the structure of the data. This choice is the subject of the next section.

IV.3.2.2 Data Models and Accessibility

Three major models for data base systems have evolved and been

---

[12] See Section IV.3.3.1.

discussed in the literature - hierarchical, network, and relational. Since the details of the different data models are not of great interest for the rest of this report, the reader is referred to either the recent book by Date ([Date75]) or the March 1976 issue of ACM Computing Surveys for a review of the different models. Each model casts the entire data base into a uniform formalism - either trees (hierarchical), networks (eg. the CODASYL system), or tables (relational system). The argument is basically that the relational approach offers considerable flexibility, but that it would require an associative memory to be efficient about its accesses. A theory of normal forms has been worked out which can optimize some updating and retrieval characteristics. The hierarchical approach is the simplest, but is awkward when the data does not fit into a simple hierarchy. The network approach is more general than the hierarchical approach and there is considerable debate about the relative merits of it and the relational approach.

Since these models accommodate differing degrees of efficiency and flexibility -- both important considerations -- the choice depends on the application. Some models have been recommended as offering greater degrees of logical data independence, that is, the capability to make logical changes to the data base without significantly affecting the programs which access it through the data management software. In data base terminology, logical changes means something on the order of changing the record structure of the files. In relational data bases, the logical structure may be changed by changing the configuration of the tables. Capabilities for this sort of flexibility are typically in conflict with requirements for efficient access or report generation along the lines of traditional data processing. For example, a programmer may organize the access requests for efficiency by following the actual physical layout of information in a file. The relational approach offers in principle the kind of flexibility that would preclude the necessity for re-organizing a program, but such systems have not been implemented with the kind of associative memory that would keep the programming efficient. In practice, the kinds of capabilities for data independence are as follows:

1. The ability to support a variety of user views of the logical structure of the data.

2. The ability to support retrieval after modest changes to the schemata.

3. The ability to tune the data structure to optimize performance for certain access patterns with diminished performance for other access patterns.

Because of the ambiguity of the phrase data independence and the great interest in representation systems which are in some sense sufficient to represent a variety of kinds of knowledge, it is worth looking briefly at what would be an ultimate form of data independence. Full data independence would mean that a data base could continue to retrieve information independent of any changes in format or computations that are needed. For example, an entry could be deleted if it were logically possible to compute it from other entries in the data base. This would require that a system must know all of the interrelationships in the data base. Expression and management of

these interactions was the purpose of the techniques in Section IV.2 and the data models mentioned above are by no means powerful enough to subsume that work.

In Section IV.3.3 we will suggest that the important consideration for knowledge based systems is the grouping of information into conceptual objects. The much smaller size of knowledge base systems as well as an emphasis on a different set of capabilities for applying knowledge at the right time have resulted in the knowledge base research concentrating on a different set of issues.

IV.3.2.3 Beyond Retrieval

Most of the research in data base management has viewed the computer system as neutral to the meaning of the data. Major emphasis has been on the trade-off of flexibility versus fast access. Experience has shown that users make mistakes when entering, transforming, or retrieving data and some tentative work has been done to help protect the integrity of a data base from certain errors due to carelessness or lack of knowledge on the part of users. These sources of error are distinct from those caused by unauthorized access (security violations), mechanical failure (reliability), or errors caused by inadequate interlocks for controlling simultaneous access by multiple users. These other errors, while important in the practical operation of large data bases, require techniques and mechanisms in addition to what will be discussed here.

Eswaran and Chamberlain [Eswaran75] and Hammer and McLeod [Hammer75] have suggested an approach for maintaining integrity which is based on (1) the specification of assertions about the data base to define the meaning of correctness and (2) the actions to be taken in event of violations. These assertions may take the form of limits on transitions in the data base (eg. The age of an employee is non-decreasing) or limits on the values for specific items (eg. salary ranges). These assertions can be checked whenever a change is made to the data base. Eswaran has suggested that the appropriate place to make many of these assertions is in the schema or data definition. Associating such checks with the schemata is a little closer to the object centered factorization of knowledge discussed in Section IV.3.3.1.

In a complicated set of operations, some assertions may not be satisfied during an intermediate state. For example, in the course of transferring of funds from one account to another by first withdrawing some funds from one account and then depositing them in another, the books would not balance momentarily. This has lead to the idea of a transaction or set of operations presented to the data base management software as a unit. Checking can also be useful during accesses which do not modify the data base. A user may specify a form of a retrieval which involves the nonsense comparison of unrelated data. When an operation involves comparison or arithmetic operations between elements of data, a form of rudimentary type checking based on data definitions can be used to detect user errors like adding dollars to doughnuts. Eswaran suggests partitioning the data base into compatibility sets within which these operations are permitted. Roussopoulos and Mylopoulos have suggested in [Roussopoulos75] that such type checking

can be facilitated by augmenting the data base with a semantic network. This work is by no means complete and the network that they proposed has some theoretical difficulties with quantification, but the idea is to augment the data base operations with network operations and checks for consistency.

This area of research on data bases is at the boundary of the work on knowledge based systems which is the topic of the next section. We will see that the extensions of these ideas lead to increased capabilities for knowledge acquisition. No doubt as these ideas get refined and developed, they will appear more regularly in data base systems.

IV.3.3 Knowledge Based Systems for Artificial Intelligence

Knowledge acquisition generally involves the acquisition of new instances of knowledge as well as of new types of knowledge. For example in the MYCIN/TEIRESIAS system, the notion of organism is viewed as a type of knowledge and the knowledge about the particular organism E. coli is acquired as an instance of an organism. Acquisition of new types of knowledge involves the most recent work on what might be termed extensibility. The work on extensibility for programming languages included capabilities for data, operations, and control, corresponding to the three classes of knowledge which we have discussed - objects, actions, and strategies. We will see that knowledge base research in extensibility rests on many of the same ideas that were used in data base systems, notably the notions of schemata, as well as the programming language work and some new ideas. The work on data base systems ignored extensibility for actions and control and concentrated on the representation of objects. The work on extensibility in the knowledge based systems of artificial intelligence has also concentrated on objects but considerable work in the acquisition of new instances of actions in the form of rules has also been done.

IV.3.3.1 Object Centered Factorization of Knowledge

One of the powerful ideas developed by researchers in knowledge based systems is the representation of knowledge as conceptual objects. This idea has been rigorously pursued by Davis with the MYCIN/TEIRESIAS system reported in [Davis76c] and by Bobrow and Winograd with the KRL language and GUS system reported in [Bobrow77a] and [Bobrow77b]. The use of conceptual objects involves a synthesis of several of the ideas from extensible language research and data base research as well as some new ideas.

The idea of organizing knowledge into conceptual objects has the same motivation as extensible language work, that is, minimizing the conceptual distance for a user. Thus, conceptual objects in the computer are expected to have many of the attributes of their counterparts in our minds. For example, a conceptual door could be opened or closed and would require its knob to be turned before it could be opened. Furthermore, the idea of specifying the structure of a conceptual object in terms of its components follows directly from the work in defining and manipulating record structures in programming language work. Continuing our example, a door may have components such

as a knob, hinges or panels. These components are in turn defined in terms of simpler entities until we reach system primitive objects like integers or strings. The conceptual objects idea also includes the clusters of Liskov or classes from SIMULA so that the procedures for the operations on an object are included as part of the object's definition. In our example, the operations open, close, lock, and unlock would be procedures in the definition of the conceptual door. In KRL, these operations take the form of attached procedures. The knowledge base is said to be viewed as object-centered in that that the objects are the primary index for accessing and they contain procedures for the operations. This is contrasted with a procedure-centered approach which uses operations as the primary index so that each procedure has special cases for the various kinds of objects. Finally, the conceptual objects idea includes the schemata from data base work. The schemata constitute external descriptions of the objects. This permits standard access methods to use the schemata as templates so that all objects can be manipulated by uniform methods.

The idea of conceptual objects relates to the discussion in Section IV.2 about context dependent (weak) and context independent (strong) interactions in that the components of a conceptual object are seen as strongly interacting. When this is the case, the object-centered factorization is an appropriate approach to reducing the complexity of the knowledge in a domain.

In addition to being a synthesis of established ideas, the conceptual objects idea includes some new ideas. In the first place, conceptual objects in the knowledge base are linked together by various kinds of relationships. Two important relationships are generalization and specialization. In our example, a fancy carved door, which might contain such components as a large gargoyle, would be a specialization of the conceptual object for a door. Specializations may inherit properties (eg. open and close procedures) from their generalizations. Another relationship might express default information about objects. Bobrow and Winograd discusses several additional kinds of relationships in [Bobrow77a]. Much of this work on linking objects together seems to derive from work on semantic networks. In particular, the ideas for inheritance of properties and for expressing relations in a network have been expressed by several researchers. This research has not emphasized the conceptual object ideas, eg., it has not involved the use of schemata. A good overview of the semantic network research research is [Woods75].

A second new idea for conceptual objects is that their schemata can be used to guide the acquisition process. This idea is an important facet of the MYCIN/TEIRESIAS research and is discussed in the next section.

Since the conceptual object idea is a synthesis of many previous techniques, it derives power from those approaches. In addition it offers an approach to solving some additional problems important in the research of knowledge base systems. One such problem is the multiple representation problem discussed by Moore and Newell in their report about the MERLIN system.[13] Multiple representations can be useful in simplifying many computations if the consistency of the various

---

[13] See [Moore73].

representations can be maintained. The importance of the matching process in working with multiple representations is discussed in [Bobrow77a]. It could also be noted that the procedural attachment mechanism gives a simple approach to maintaining consistency among multiple representations. Bobrow and Winograd distinguish between procedures that are activated when some component is modified and those which are activated in order to fill a component. (These are termed demons and servants respectively.) Servants can provide a mechanism for maintaining consistency between objects viewed as multiple representations. In the door example above, a servant could be used to update a connection table for conceptual rooms whenever a door was opened or closed. In this example, there are multiple representations for the state of the door. Presumably the connection table representation is convenient for calculating paths between rooms.

Another major benefit of the grouping of the knowledge into conceptual objects having schemata is the potential for system assistance in the acquisition of instances of the objects based on their descriptions. This is the subject of the next section.

IV.3.3.2 Acquisition of Objects

The idea of using schemata to group facts together into large entities has appeared in several places in artificial intelligence. In KRL these are called units. In the GUS system[14] they are called frames. In Schank's work, they might be called scripts. The MYCIN/TEIRESIAS system already discussed in Section IV.2.3 is unique in its use of schemata to guide the process of acquiring knowledge. The use of schemata to guide this acquisition process is an essential advance over the extensibility techniques in the programming language research. The schemata provide the knowledge about knowledge that the system needs in order to provide assistance during the acquisition process.

In June 1976, the MYCIN/TEIRESIAS knowledge base contained information about 125 different organisms. A single organism schema is used to describe and guide the acquisition of the fairly complex information structure required for each organism. In the early versions of MYCIN adding a new organism to the system meant doing it manually with little machine assistance and it was a common mistake to forget some part of the substructure. In addition it was necessary to appropriately update several other representations in the form of lists and tables in the system. These two problems - maintaining completeness of the substructures in representations and maintaining interrelationships between them have provided a focus for the use of schemata to guide knowledge acquisition.

The acquisition of a new organism in MYCIN/TEIRESIAS uses a schema in creating a dialog with a user. The schema provides the framework for knowing what information will be required, how to ask for it, how to check it, and how to update various internal lists automatically without concerning the user. In addition, the schema ensures that the creation of the new instance of an organism will be documented as to author and date.

---

[14] See [Bobrow77]

The information in the domain is organized into conceptual objects, each of which has a schema. These schemata have access to information relevant to filling in new instances of each conceptual object. They guide the interactive dialog with a user for entering new instances of the objects. The schemata have two features which are especially important for this. The first is a limited form of procedural attachment of the slot experts, which functions on the most primitive representations in the system. Associated with each slot expert are English phrases for prompting or displaying information to the user and a procedure capable of filling in instances of the data according to advice passed from the schemata. (This is the only example of procedural attachment in the MYCIN/TEIRESIAS system with the limited function of facilitating the filling of slots.) The second important feature of the schemata is the specification of updates to lists and tables used whenever a new instance is created. Thus MYCIN/TEIRESIAS has two levels of data typing (1) the complicated domain level structures configured by their schemata and used as components in the rules and (2) lower level slots which have associated slot experts and prompting information. Within the system these lower level structures correspond to those entities which are in a sense too small to be decomposed - so that their schemata are of an almost trivial form.

The discussion above centered on the question of adding a new instance of a knowledge type to the system. One of the important ideas from the knowledge base research in [Davis76c] is the idea that schemata can themselves be considered to be a type. This means that many of the same mechanisms which are called into play to create a new instance can be used to create a new type. This approach to extensibility is the subject of the next section.

IV.3.3.3 The SCHEMA-SCHEMA

In the MYCIN/TEIRESIAS system in June 1976, there were 125 organisms but only one organism schema. There were 25 schemata for the various data types in the system. Describing the format of every schema in the entire system is the single schema-schema. These numbers reflect a very high utility for each schema in the system and emphasize the important role each schema can play in the acquisition process.

The process of acquiring a new type of conceptual object in TEIRESIAS proceeds by first acquiring a schema for that object and then acquiring an instance of that schema. In [Davis76c] Davis gives an example of the creation of a new schema for nutrients. This example starts in the context of entering a new rule in the system when the phrase "nutrient of the culture medium is blood-agar" is mentioned in the premise. This initiates a dialog where the system uses the schema-schema to guide the acquisition of a schema for nutrients. Creating the schema involves acquiring English phrases for prompting as well as establishing the relationship of a nutrient schema to other schemata in the system. The schemata in MYCIN/TEIRESIAS are connected in a network by the FATHER and OFFSPRING links in each schema and by the <datatype>-INST links in the slots. The FATHER and OFFSPRING links determine a network of schemata which is used to make possible the inheritance of properties. In particular, a father schema may be viewed as a generalized schema which contains all of the information that its

offspring have in common. At the end of this example, a new schema has been created which points to its instances and which has been integrated into the schema network. Any subsequent operations on the network will involve the new nutrient schema as well as the other schemata.

The acquisition process is skillfully guided by the schemata network. Acquisition is broken down into small, easily understandable steps. There are also two simplifying assumptions made which limit the schemata which can be acquired by this dialog. The following information is not acquired in this way.

1. New slots in the schema which are not inherited from ancestors in the network.

2. Updating specifications for internal multiple representations.

This information, while important in many cases, was considered to be beyond the expertise of the user, a domain expert. The MYCIN/TEIRESIAS philosophy has been to isolate the user from programming details with the small possibility that the knowledge base may be compromised if the new data type is in fact related to an existing internal structure. It should be noted that the program was capable of acquiring this information from a user, but that it was inhibitted from doing so for the reasons stated. These special kinds of information were acquired by the use of a special network editting program.

Section IV.3.1 discussed work that was done to provide extensibility for three classes of knowledge. In the work described above, we have focussed on extensibility for the objects of a system and seen that the SCHEMA-SCHEMA provides the essential knowledge about knowledge for acquiring new kinds of conceptual objects. We have not discussed extensibility for the actions of the domain. The reason for this derives from the task of the deductive consultation program. Although there are about 400 rules in the MYCIN/TEIRESIAS system, the right hand sides for all of them (except for a few meta-rules) are uniformly "CONCLUDE". Similarly, the 24 predicate functions (for example, SAME, KNOWN, DEFINITE) have been static over the life of the project and it has not proved necessary to provide for extensibility in these functions. Davis makes no claim of having solved the problems for extensibility for either the predicate functions or new classes of rules. Although this system has not included research into the acquisition of new types of rules, it has provided some noteworthy examples of the acquisition of new instances of rules. This is the subject of the next section.

IV.3.3.4 Acquisition of Actions

Sometimes in the course of a diagnostic session a user may decide that MYCIN/TEIRESIAS has drawn an unsatisfactory conclusion. This is generally an indication that some change in the knowledge base is required. In this event, he has the option of telling the system which conclusion should or should not have been made and having the system assist him in tracking down the problem. If this option is chosen,

TEIRESIAS will access its history list of the consultation and answer questions about why certain rules were or were not invoked at any stage of the consultation. When the user believes that he has found the knowledge bug, he can modify a rule or add a new rule to the system. If it is a new rule, the system will attempt to classify it and compare it to similar rules in the system and may suggest some modifications to the rule. For example, if almost all of the other rules which conclude about the same organism mention portal of entry in their premise, the system may ask the user if he wishes to add such a clause to the premise. These rule models differ from the object schemata in the previous section in that (1) they are derived from the rules in the knowledge base and (2) They are not used to guide the acquisition process for rules as completely as schemata guide the process for objects. For example, they do not have the ability to fill in parts of a rule and they do not correspond to types of rules. More about this will be discussed in Chapter V.

Finally, the system can use the context in which a rule was entered to check its suitability. When the user has completed his fix, the system remembers the context in which the problem was discovered and checks whether the fix actually remedies the situation. These capabilities for acquiring knowledge from the user about actions are one of the distinguishing features of knowledge based systems. Since strategies in the MYCIN/TEIRESIAS system are also expressed in rules, this gives the system the ability to acquire new instances of strategy as well as domain level knowledge.

IV.3.4 Summary of Knowledge Acquisition Work

Having completed this section on knowledge acquisition, it is worth reviewing the highlights briefly. The first work discussed was in the area of programming languages. Extensibility for data, operations, or control meant the ability to define new entities in terms of existing ones. It was discovered that this idea was not in itself powerful enough to significantly reduce the errors in constructing a system or to reduce the conceptual distance. Considerable knowledge about the system was needed to successfully introduce new types.

The next area of work was in the area of data base management. This work introduced the idea of a data definition or schema as well as some tentative work for data type checking for operations in the data base. The schema provided an external definition of the structure of the data and made possible the manipulation and access of the data base by standard routines.

It remained for the knowledge base research to use these ideas to provide powerful techniques for knowledge acquisition based on an object centered view of the knowledge base. In Section IV.3.3.1, the notion of conceptual object was defined. The conceptual object idea was seen partly to be a synthesis of ideas from programming language and data base research. It used existing ideas for defining objects in terms of their components and defining the procedures for operations on objects with the objects themselves. This viewpoint has been termed an object-centered viewpoint and may be traced back to Liskov in the

extensible language research. [15] A new aspect of the conceptual objects idea included a network of relations which makes possible the inheritance of properties between related objects. A large portion of the work on conceptual objects has been done by Bobrow and Winograd in their development of the KRL language.

In terms of the knowledge base interactions discussed in Section IV.2, the conceptual objects may be seen to consist of components which should be used together in all contexts.

One of the most important contributions of the knowledge base research in [Davis76c] was the realization that the schemata for conceptual objects could be used to guide the acquisition process. These schemata provide the essential knowledge about knowledge that was lacking in the extensible language research and make possible considerable assistance from the system in acquiring objects. Extensibility is achieved by having a schema for schemata so that the system can acquire new types of knowledge by first acquiring schemata for them. The schemata have access to procedures for filling values of new instances and for maintaining consistency between multiple representations of objects. The acquisition process can help insure knowledge base consistency. More work is needed in this area to handle (1) the acquisition of new slots in a schema other than those which are inherited from ancestors and (2) the updating of multiple representations. In particular it is worth exploring ways that the system can assist the user in finding such representations and in establishing procedures for updating.

Most of the research on extensibility has concentrated on the objects of the knowledge base. Knowledge base research on the acquisition of rules has concentrated on the acquisition of new instances of rules making good use of rule models and context information. More work needs to be done on the use of schemata for rules and possibly on the acquisition of functions, eg. predicate functions. The next chapter suggests continuing this line of research by creating schemata to guide the acquisition of rules including relatively complex strategy rules.

IV.4 Summary of Knowledge Base Research

We began this chapter on knowledge base research by observing its importance to problem solving. Effective problem solvers in complex domains require significant amounts of domain specific knowledge. Some questions for problem solving systems become significant for knowledge bases of this size.

How can the knowledge be acquired?

How can changes in the knowledge base be accommodated?

---

[15] It has also been an important element in Hewitt's ACTORS, and in the SMALLTALK system. We have not attempted an exhaustive survey of this idea.

How should knowledge be managed so that it can be used in several different problem solving situations?

The first question was the concern of the previous section. Ideas for the second and third questions lead us to Section IV.2 about knowledge aggregation. Summaries of both sections have appeared above.

One of the unifying themes of knowledge base research is the idea that meta-knowledge, that is, knowledge about knowledge can be used to facilitate capabilities for the multiple uses of the knowledge base. In Section IV.1, we discussed some desired capabilities for a knowledge based system. We identified the needs for knowledge acquisition, problem solving, and explanation. Three classes of domain knowledge as the objects, actions, and strategy/control knowledge of the domain were distinguished. Strategy knowledge may be viewed as a form of meta-knowledge about actions which facilitates problem solving. Schemata may be viewed as a form of meta-knowledge which facilitates acquisition. Procedures attached to the schemata can be a form of meta-knowledge which facilitates automatic updating in the knowledge base. Statistical knowledge derived from the knowledge base, for example the rule-models of TEIRESIAS, may be viewed as the meta-knowledge for checking new instances or for suggesting possible defaults. Information about the problem solving performance of rules in different situations could be used as meta-knowledge for debugging the knowledge base or guiding the selection of strategy methods during knowledge acquisition.

In the next chapter, we will propose extensions to these ideas to extend the capabilities of the knowledge base for the following additional requirements:

1. How can a variety of types of domain actions be accommodated in the knowledge base?

2. How can a variety of types of strategy and control knowledge, (such as those mentioned in Chapter III) be incorporated in a knowledge base?

3. How can a variety of types of problem solving states be expressed and manipulated by the system?

4. How can the problem statements for a variety of types of problems be acquired?

5. How does the expression and representation of problem solving states relate to the expression of the domain and strategy knowledge?

In the next chapter, we will propose extending many of the kinds of meta-knowledge mentioned above to cover these additional requirements of a knowledge-based problem solving system.

Chapter V

Tentative Proposed Work

The applications goal of the MOLGEN project is the crafting of a computer system which will perform as an informed assistant for the design of experiments in molecular genetics. The artificial intelligence goal is to test some ideas for the representation of knowledge and the management of a complex knowledge base. In what follows, we will be examining the knowledge and planning processes involved in the design of a limited class of scientific experiments.

Of special interest will be the management of strategy knowledge, that is, the knowledge which directs the control structure for the creation of experimental plans. Thus, strategy knowledge is not limited to some set of useful heuristics which are invoked occasionally during planning. Rather, the term strategy is being used in its broadest sense to mean the knowledge which directs the entire problem solving process. In this broad framework, the planning process is carried out entirely under the control of strategy knowledge from the very beginning of a MOLGEN problem when a top-level strategy rule is invoked.

V.1 Perspectives and Observations about the Direction of this Research

Chapter IV traced the development of many of the ideas for representing knowledge in a computer. The earliest work we examined was the work on extensible programming languages. Perlis was quoted as observing that three things define a language - data, operations, and control. We observed that these correspond directly to three kinds of knowledge for a knowledge base - objects, actions, and strategy. It was generally thought that extensibility in a programming language would result in clear and efficient programs and that these programs would be easy to write. The important lesson from this work was that the amount of knowledge necessary for a user to mold the nature of a system for his requirements had been seriously underestimated. The systems themselves remained too ignorant to provide much help.

Several of the ideas have been developed further in data base research and knowledge base research. We saw that knowledge base research was making headway on the extensibility issue in its efforts to create problem solving systems that could use a large base of domain knowledge. Most of the progress in extensibility has taken place on the definitions of objects and much less has been done for the actions and strategies. The work on objects introduced the notion of conceptual objects[1] and the use of schemata to guide the acquisition process. Schemata provide the essential knowledge about knowledge that was lacking in the extensible language effort. A schema for schemata (the

---

[1] See Section IV.3.3.1.

SCHEMA-SCHEMA) made it possible to acquire a new type of object by first guiding the acquisition of a schema for it and then using that schema to acquire the instance of the object itself.

Some good work has been done on the acquisition of rules. In the MYCIN/TEIRESIAS system, rule models, derived from rules in the knowledge base, were used to create expectations about new rules acquired by the system[2]. Differing in function from schemata, these rule models were not used to fill in parts of a rule but rather were used to create reminders for the user based on the assumption that new rules would follow the patterns of rules already in the knowledge base. Since the MYCIN/TEIRESIAS has only one kind of rule[3], the description of rule components is built into the program for rule acquisition.

We propose to extend this line of research into the acquisition of more types of action and strategy knowledge. The belief in the feasibility of this proposal is based on a number of observations and assumptions which are listed here.

1. Many important logical constructs are difficult to express in the simple production rule format. For example, iteration is awkward to express or recognize in typical production rule systems. Examples of control strategies will be presented in Section V.3 and Section V.4 which could not be expressed in a single MYCIN-like production rule. Clarity requires that these strategies be expressible in a single coherent module. (It is not satisfactory to create a complicated structure involving several rules and dummy linking variables in order to force the expression into a restrictive production rule style.)

2. In addition to a requirement for the ability to express strategy and control information, it is important to maintain the visibility of the components of the rule so that a rule can be analyzed by the system. Thus, the idea of using production rules and hiding the important part of the algorithm in a non-decomposable procedure named by the right hand side of the rule defeats this purpose. Existing systems, such as NOAH with its QLISP procedures, have required abbreviated descriptions of the actions, supplied by the programmer (in this case ADD/DELETE lists), to enable the system to reason about the actions. Such systems do not have the capability to abstract this information from the rules directly.

3. Parallel to the desire to make the components of a rule available to the system for analysis is the desire to make structure information available for guiding knowledge acquisition.

---

[2] See Section IV.3.3.4.

[3] Every rule in TEIRESIAS could be viewed as an instance of a schema with an "If" component and a "then" component.

4. Just as an object may be decomposed into its component objects, an action (or strategy) may be decomposed into its smaller component actions. It is proposed that the schemata idea be extended from objects to cover knowledge about actions and strategy as well. Thus, a schema for a type of domain action would be used to guide the acquisition of an instance of that action.

5. The reference by content mechanism used in the TEIRESIAS system for accessing rules by description is inadequate for dealing with complex rules. The reference by description mechanism, as implemented, could distinguish the use of a token only by its position as being either in the premise or action part of a rule. Use of schemata for rules provides a description of the rule substructure and facilitates more sophisticated pattern matching facilities.

6. Schemata for actions and strategy, like schemata for objects, can be used to fill in default or computed values for components, to insure that no necessary components are left unspecified, and to tend to updates in the knowledge base. For example, a schema for Separation Technique actions would require information about the basis and resolution of separation. It would automatically fill in the parts of the action rule which direct the system to loop through all the structures in the current sample. Furthermore it would update the knowledge base by insuring that the new instance of a separation technique was included on the appropriate lists so that it would be used when necessary by the problem solving process.

The sections which follow will fill in some of the details of this proposal. We will see that the schemata for strategy knowledge create a powerful approach for providing a toolbox of problem solving techniques. These techniques can be instantiated to create strategy and control rules for the knowledge base. A sophisticated type of procedural attachment, termed inspectors, will be introduced which will make it possible to express strategies without some of their complicating special cases (because the system will already know about them). Finally, meticulous adherence to the principle that everything should have a schema will lead us to creating schemata for such things as world-states and even the current state of the problem solving process (termed the planning network). This approach enables us to represent a spectrum of complex entities with a uniform and consistent mechanism. This will greatly simplify the programming of the system and make a great deal of information, which is typically represented in an ad hoc manner, a visible part of the knowledge base.

V.2 MOLGEN System Sketch

To provide reference points for the rest of this proposal, this section will begin with a sketch of the proposed MOLGEN system. The MOLGEN system will be very large and will be built and designed by several researchers. Much of it will consist of programs but most of it will be the knowledge base. The following diagram shows the major components of the system.

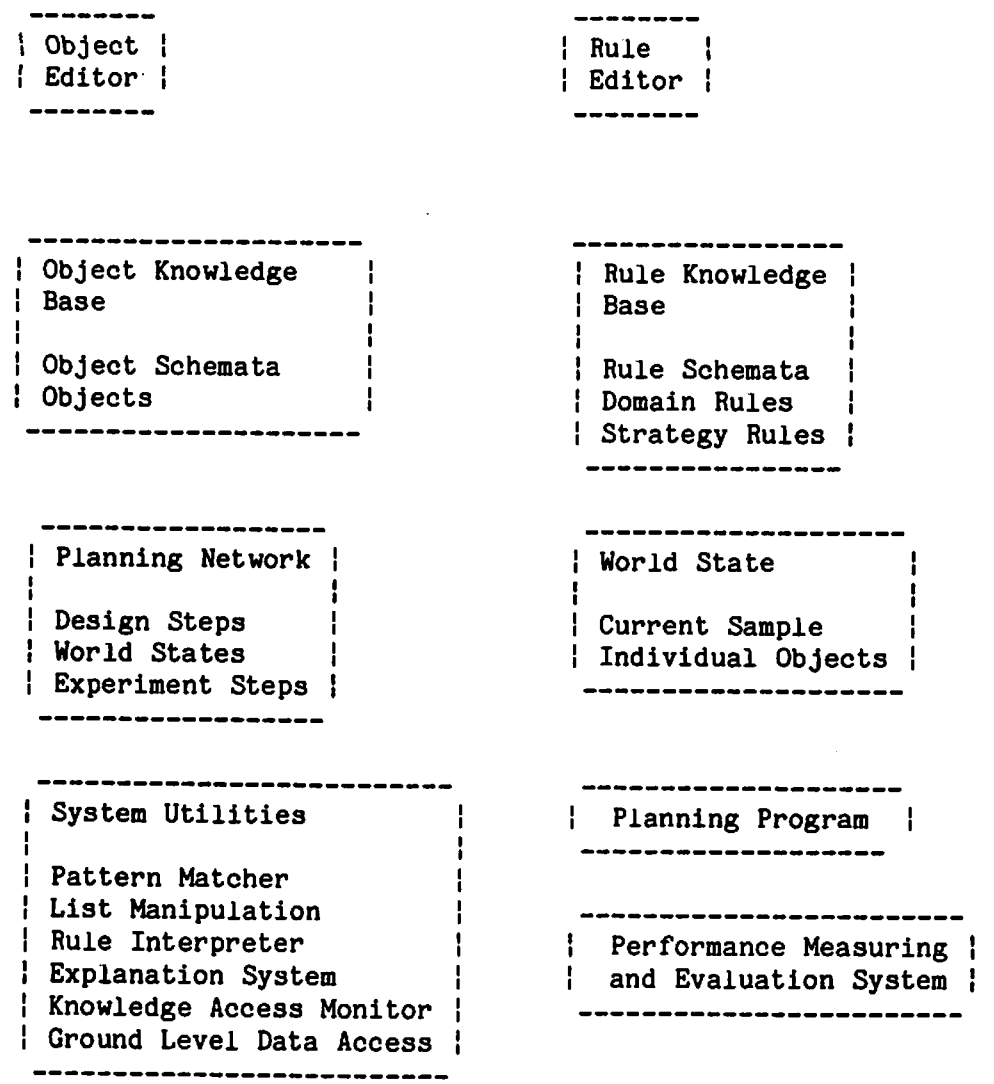

Figure 4. MOLGEN System Components

The object editor and rule editor are programs for knowledge acquisition. The object editor will be a system for entering schemata and objects; the rule editor will specialize in the acquisition of action and strategy knowledge. Both editors will use schemata to drive the acquisition process. The term rule (as contrasted with procedure) is meant to connote something which is structurally simple enough for

the system to examine and analyze. As will be discussed in the next section, the rules of MOLGEN will be extended to cover much more complex processes than were needed, for example, in the MYCIN/TEIRESIAS system. The schemata for the strategy rules will embody a collection of problem solving techniques termed the artificial intelligence toolbox.

All of the dynamic information, that is, information which changes in the course of problem solving, will be contained in either the planning network or the world state. The planning network is the representation of the problem solving state and is discussed below in Section V.3. The world state is the current sample (or samples) containing all the information about substances and other entities (eg. temperature) which are present in the simulated genetics environment at the current moment in planning. Previous or predicted future world states are contained within the planning network.[4] Both the planning network and world state have prototype schemata in the object knowledge base. All of the actions in the rule knowledge base -- both domain and strategy -- are defined in terms of the changes they induce in the state information.

Finally we come to the planning program which is in many systems the heart of the system. In MOLGEN, however, the structure of this program will be very simple -- since most of the work is driven by information in the knowledge base. The operation of this program would start with the acquisition from the geneticist user of a problem statement. Since this involves acquisition of knowledge, it would be guided by schemata and the work would actually be done by the object editor. The acquisition process would include initializing the world state and the planning network. The next step for the planning program is to start the problem solving process with a top-level strategy rule. This rule may be selected through the schema for the problem statement. Given the name of this rule, the planning program invokes the rule interpreter to start the problem solving process.

Further tasks for the planning program would be to field interrupts from the user which re-direct the planning process. The program would also manage a display of the evolving world state and planning network. Attention of the experiment design process could be manually re-directed when the expert interrupts the planning program and invokes a strategy rule on a different aspect of the planning network. He may elect to save the current planning network on a file so that he can return to it later. Finally, the MOLGEN explanation system could be invoked to explain the events of the problem solving process. As the system becomes polished, smooth interfaces to the object editor and rule editor to allow changing the knowledge base during planning will be developed.

From this description of the planning program, we can see that much of the programming work has been transferred into the general

---

[4] The phrase "world state" is being used in the same manner as in robot planning work. In this work, the planning and control information is not considered to be part of the world state. In a problem solver capable of considering alternate beliefs about the world or many views of the world at different times, it is appropriate that there are several world states. In such a system, one of them may be designated as the current world state.

system utilities. The problem solving process is rule based so that the knowledge which directs the process is contained in the knowledge base and the programming effort is limited to building the routines for creating the knowledge base and a rule interpreter.

The last component in the figure, the performance measuring and evaluation system, will be integrated into the planning program and knowledge base routines. When the system is designing experiments, we may ask on what basis its effectiveness can be judged. Similarly, when a user is entering strategy knowledge or the system is choosing between strategies, on what basis can a selection be made? The idea is to build mechanisms into the system which facilitate the gathering of information on which to base these decisions. The creation of measuring tools and evaluation procedures will be a central theme for one of the MOLGEN researchers.

The next section suggests that this design offers tremendous flexibility for trying out new strategies and planning paradigms. What would traditionally have required a new planning program can be done in this design by acquiring a new strategy rule. All of the power of schemata-driven knowledge acquisition is available to make the acquisition of and experimentation with new strategies as painless as possible. We believe that this flexible design will result in a powerful laboratory tool, so that MOLGEN can make some real contributions to the practical design of interesting laboratory experiments.

V.3 Strategy and the Planning Network

There are several sources of information which strategy processes need to access and manipulate in order to create plans for experiments. One source of information is the knowledge about the objects in the domain. When a domain rule has a condition relating to an object, knowledge must be brought to bear for deciding whether to treat that condition as a presupposition or a precondition. Similarly, the desirablility of a given rule is determined in part by the effects it has on objects in the domain. Thus strategy information must deal with the knowledge of the objects and actions in the knowledge base. In addition to this, strategy must deal with a knowledge of the current world state. Determination of which domain actions are feasible is possible only with a context provided by the current world state. The nature of the knowledge in the world state may change, for example, early in the experiment design process, the world state knowledge could be of an abstract nature. Finally, strategy must be concerned with the current problem solving state as indicated by the planning network mentiond in the previous section. As will be discussed below, the planning network provides for the expression of the orderings or partial orderings of the steps of developing plans and the entire history of world states and tentative planning steps that have been sketched out by the planning process. Focus of attention directives can be expressed in terms of the planning network, which provides a language for directing problem solving effort to different facets of the problem.

Since the planning state knowledge is important for the expression of strategy in MOLGEN, it is worthwhile exploring briefly the nature of this knowledge. It is useful to consider the planning network in MOLGEN as being composed of three planes -- the experiment plane, the planning plane, and the focus plane. These planes contain (1) the experimental steps and world states, (2) the planning and design steps and (3) the focus of attention knowledge respectively. All three planes of the network are built dynamically during the problem solving process. Different types of nodes in the network correspond to the different components of the problem solving process.

It is natural to begin with a brief description of the kinds of nodes in the experiment plane. These nodes are express a solution to the design problem. In the simplest case, this corresponds to a sequence of laboratory steps that transforms the initial laboratory conditions to a set of final conditions. These final conditions may reflect modified structures or simply an increased state of knowledge. More generally, there will exist branch points in the experiment plan. These correspond to those places where design proceeds along alternate paths depending on a laboratory measurement in the sequence, the results of which cannot be known until an actual experiment is performed. In terms of nodes in the network, three kinds of nodes are suggested. The first kind of node corresponds to the world states along the way. These nodes would express the initial, final, and intermediate states of the laboratory conditions in the experiment. World state nodes carry the dynamic knowledge which can be changed in the course of an experiment. Between world state nodes are the action nodes which describe the genetic actions used to transform the states. These point to corresponding rules in the rule knowledge base which describe the appropriate state changes for the experimental step. The action nodes would also contain the values of the experimental parameters (eg. gel voltage gradient) for each of the transformations. Finally, a third kind of node expresses the conditions at the branch points in the experiment plans.

As will become clear from later examples, the experiment plane may be inhabitted by nodes which represent world states or laboratory steps expressed at different levels of abstraction. Early in the design process, nodes may be formed which deal with models of DNA that are quite abstract and with very generalized laboratory steps. Typically, these general steps will be refined to more specific ones as the design process continues, for example, cutting may become an exonuclease or separation may become electrophoresis. It is not too surprising that the generalized steps and actions will appear in the experiment plane. For some purposes, the design process may be stopped if the general plan is already a complete enough answer for the user. Even the most specific plans the program will produce will contain a certain amount of abstraction.

In the planning plane above the experiment plane, is a representation of most of the problem solving activity which creates the design of the experiment. The nodes in this plane correspond to the basic problem solving operations described in Chapter III. Just as action nodes in the experiment plane point to domain rules which express laboratory transformations, each kind of node in the planning plane points to an appropriate type of strategy rule. These rules express such operations as generating an alternative, refining a step,

testing the suitability of world states which have been created, or ordering some partially ordered steps[5]. It is our contention that a few types of these operations cover all of the problem solving operations. In terms of schemata, this means that a small number of schemata are needed to represent the many problem solving nodes in the planning plane. For example, a refinement node schemata would have a slot for a rule which maps a general world state state to a specific one and a slot which maps a general action in the experiment plane to a more specific one. The basic role of a refinement rule is the proposing of subproblems. The refinement node keeps track of the mapping rules which are active and the correspondence between states and actions in the experiment plane. During a design process which used hierarchical planning, many levels of refinement nodes (pointing to other refinement nodes) would exist in the planning plane.

A basic question which dominates much of the design process is the question of the allocation of resources. A growing network may contain several approaches to a problem and several incomplete subproblems. The focus of attention problem, discussed in Section IV.2.2.2, is the problem of deciding where to allocate resources such as processor time to the various competing places in the partially completed design process. We propose the use of a number of focus of attention rules which manage this process. In terms of the network elaborated above, focus nodes located in the focus plane above the planning plane will be responsible for allocating processor space and time to the activities represented below in the planning and experiment planes. Since these planes express the complete problem solving state, they provide a language for expressing the control necessary for the focus of attention process.[6] A focus of attention rule would base its decision about resource allocation to areas of the problem on information available in the slots for that part of the problem. For example, nodes which generate alternatives in the planning process could contain estimates of the cost of generating the next alternative. The suitability of an alternative could be estimated by activating the rule in a test node. Each focus node would contain a measure of the resources it had to spend, the name of its focus of attention rule, and a pointer to that part of the planning network which was its particular domain. One capability for a focus node is to insert another focus node over a subset of its domain. This corresponds to a delegation of authority which allows for specialized approaches to the allocation of resources for different parts of the problem. This also promotes the practice of describing the focusing process in terms of small modules.

It is interesting to compare the planning network to some related structures in artificial intelligence. The planning network is like the blackboard of HEARSAY in that it expresses the state of the problem solving process. It differs in that the blackboard of HEARSAYII contains a fixed decomposition of the speech understanding problem. The structure of the MOLGEN planning network may be modified by changing the schemata in the knowledge base for the nodes in the network. Unlike the knowledge sources of HEARSAY, those of the MOLGEN planning network

---

[5] (As in the resolve conflicts critics of NOAH).

[6] Each competing planning process is addressable through a node in the network. A process could be initiated by activating the slot which names the rule.

correspond to actions and strategies in an acquired rule knowledge base. The planning network is also related to the procedural network of NOAH. Both networks contain nodes for the actions, world states, and abstractions of both of these. The planning network differs in that it contains nodes for the strategy information. It also differs in that the state information is retrieved from object schemata instead of the ADD/DELETE lists. The idea for expressing focus of attention in nodes distributed throughout the network is unique to the MOLGEN planning network as is the idea of expressing these nodes uniformly using schemata.

Before leaving the subject of focus of attention, it is interesting to recall from Section IV.2.2.2 that this process has two components: (1) selection of a problem solving context in the problem for further work and (2) selection among competing knowledge sources to apply in that context. In the MOLGEN framework, these two components are handled separately. The focus of attention rules are responsible for the selection of a problem context, that is, for the appropriate sites for further refinement or allocation of resources. The second component, selection among competing knowledge sources, is a process which will utilize pattern matching. In this case, the search is among competing domain or strategy rules to apply within the current planning context. This search process is expressed in the generator and refinement rules used in the network. The use of schemata to create rules creates a description of substructure of the rules being searched and facilitates this pattern matching process.

Corresponding to the classification of nodes in the planning network into a few types is the potential for classifying their associated rules into a few types. This refers to the central idea of having schemata for each kind of action and strategy. Thus, focus of attention rules are concerned only with the allocation of resources, refinement rules are concerned with the generation of subproblems, action rules are concerned with the transformation of state information, and so forth. The schemata for these kinds of rules are specialized so that that the acquisition process for these rules can be based on a set of specific expectations. The next section pursues this specialization process further by suggesting that the algorithms internal to these rules may be discussed in terms of a set of standard artificial intelligence tools.

V.4 A Toolbox for Artificial Intelligence

At the end of Chapter III, a number of issues were raised about the management of strategy knowledge.

1. How should strategies be expressed?

2. How can strategy information be assimilated so that the system will use it appropriately when designing or explaining experiments?

3. How can a knowledge based system assist a domain expert in structuring and expressing his ideas about strategy?

In this section we will propose an approach to the acquisition and management of strategy knowledge -- beginning with a familiar example.

Means-ends analysis is one of the simplest ideas in the current stock of methods for problem solving. As such, it should exist as a tool in a toolbox of artificial intelligence techniques to be used as needed. The current state of artificial intelligence, where a researcher must re-code Means-ends analysis any time he wishes to use it is akin to a carpenter forging a new hammer for each job. In the next few paragraphs we will explore this Means-ends analysis example and examine the system capabilities that are necessary to create such a tool kit. Many of the techniques for creating these capabilities are natural extensions to those presented in the previous chapter. This example will also provide a framework for introducing some terminology for the sections which follow.

V.4.1 The Means-ends Tool

Initial World State	Difference Table		Goal State or Goal Test
	D1	A1	
	D2	A2	
Difference Function	D3	A3	
or	...	...	
Feature Function	Dn	An	

Figure 5. Components of Means-ends Analysis

The elements of Means-ends analysis are presented in figure 5 above. Typically there is an initial world state and either a goal state or a goal test. For a synthesis experiment in molecular genetics, the initial state might be the initial molecule and the goal state might be the molecule to be synthesized. Existing chemical synthesis programs generally work synthesis in the reverse direction working backward from the desired molecule to any acceptable precursor. This alternate approach fits within the goal test paradigm, where the testing function decides whether a candidate molecule is acceptable as the starting precursor. An analysis experiment, such as the binary discrimination experiment described in Section II.1, could be expressed within a goal test paradigm.

The classical Means-ends analysis calculation proceeds a step at a time from the initial state to a goal state. At each iteration, a difference function is invoked to find the differences between the current state and the goal state and an ordered difference table maps the differences to their associated actions. In the goal test paradigm, the difference function is replaced by a feature extraction function and the table is used in much the same way.

One approach for making an instance of Means-ends analysis available as a tool would be to provide a packaged program which accepts arguments for the various components of Means-ends analysis (eg. a difference table, difference function, etc.). The alternative

being proposed here is a system which uses schemata to drive the strategy acquisition process and which can guide a user through the details. The goal is to create a supportive environment for the painless testing of fairly high level strategies. Such a system should be able to draw on its knowledge base to provide assistance in casting a problem into a Means-ends framework.

V.4.2 Means-ends Analysis in the Schemata Network

In terms of the planning network discussed previously, Means-ends analysis corresponds to a specialization of one of the planning operations, eg. a refinement process.

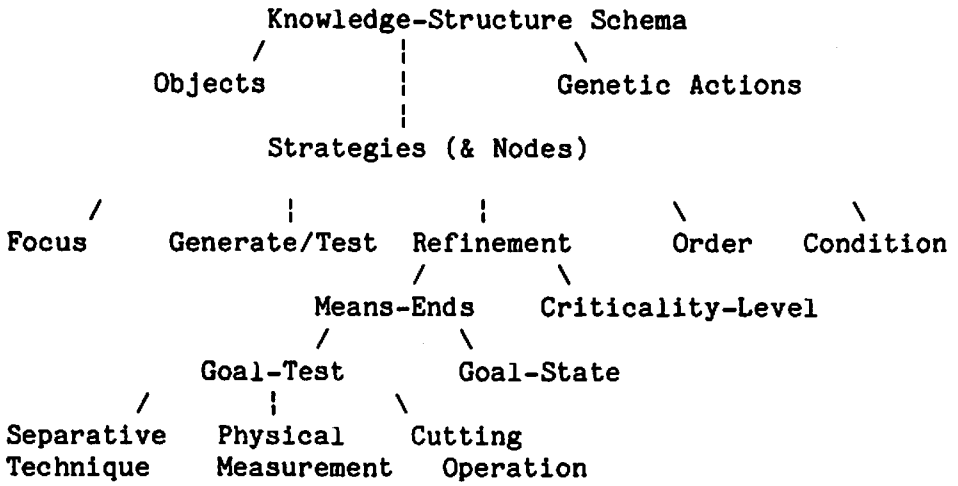

Figure 6. Fragment of the Schemata Network

The figure above illustrates the set of relationships between some schemata which could exist in the MOLGEN knowledge base. At the top of this hierarchy is a schema for knowledge structure.[7] Specializations of this are the schemata for the three classes of knowledge for MOLGEN - objects, actions, and strategies. For the purposes of this section, only the specializations of strategy are expanded here. The schemata at this level correspond to the kinds of nodes in the planning network described in the previous section. Thus we have schemata for rules for focus nodes, generate/test nodes, refinement nodes, order nodes, and condition nodes. The schemata for the remaining kinds of nodes in the planning network are not shown above. For example, the schema for world states would appear under object nodes, and the schemata for genetic actions and their abstractions would be specializations of the action schema. Continuing with the network above, specializations of the schema for refinement rules include those refinements which are based on Means-ends analysis and those based on criticality levels. The network above shows two versions of Means-ends analysis - the goal state and goal test versions. Under the goal test schema,

---

[7] (This corresponds to the KSTRUC Schema in the MYCIN/TEIRESIAS system.)

specializations for separative techniques and for physical measurements are shown.

One purpose of this schemata network is to express the inheritance relationships between the schemata. It is worth examining the inheritance implied for the node labeled separative technique in the diagram above. In the first place, it is a knowledge structure and in particular a strategy. More particularly, it is a schema for a refinement strategy which means that an instance of it in a planning network will involve a refinement node and a corresponding abstract action rule. (The abstract action rule, in this case, would be one that carries out the separation on an abstraction of the world state.) The refinement rule is concerned with proposing subproblems. The network also indicates that this rule is based on a Means-ends algorithm which means that the subproblems will be proposed by Means-ends analysis of the input world state to the refinement process. The particular type of Means-ends analysis is the goal-test paradigm. The rule will use the goal-test paradigm to propose a refinement for a separative technique. At this point we should note that the separative technique schema is not itself a rule but rather it is the schema for guiding the acquisition of such a rule.

Much of the information about the rule in our example is inherited from the Means-ends analysis schema. For example, this schema would indicate that a difference (or feature) table is required as well as a difference function. The particulars of these must be acquired from the user when he enters a rule. The schema may point to tests to be performed at knowledge acquisition time which check the tables that are entered. The differences are to be expressed in terms of the properties of objects that are in the knowledge base, for example, particular DNA structural features. To provide assistance, the system must scan its object knowledge base and suggest features which should be in the table. These differences must map to actions which are also drawn from the knowledge base - the genetic actions or legal moves. In this example, we see that at least one of the actions must correspond to a separation technique. One can imagine tests in the schema which have the capability to check that the actions chosen have an appropriate relationship to the differences which are set to trigger them. Finally, the loop inherent in Means-ends analysis would be filled in automatically by the Means-ends analysis schema. Then the goal-test schema would guide the acquisition of the feature extraction function and the goal testing criteria.

In summary, the acquisition process for a strategy rule is broken down into a number of small and manageable steps. The schema used to guide the acquisition process inherits many of its specifications for creating the rule from its ancestors in the schemata hierarchy. It is suggested that this process can be used to help prevent required entries from being forgotten when a new rule is acquired. Much of the structure of a rule can be filled in automatically - for example, the iterative loop in the Means-ends analysis example. Tests on the sets of acceptable values for the components of instances can be built into the schemata as a further check on the correctness of what a user enters. The goal of this process of assisted acquisition is to make the acquisition of domain specific strategy rules as painless and bug-free as possible.

V.4.3 More From the Toolbox

Although the previous section emphasized the example of a Means-ends analysis tool, the schema based approach would be used for any problem solving technique that the system could apply. For example, another technique would be the criticality level approach to hierarchical planning. The interactions between schemata would be somewhat more complex in this technique but the methodology of acquiring the knowledge in small pieces using schemata would be used again. For example, one aspect of using a criticality level approach is the assignment of criticality level numbers to objects in the domain. Another aspect of it is the length first expansion of the design. During the process of hierarchical planning, the planning network might appear as follows:

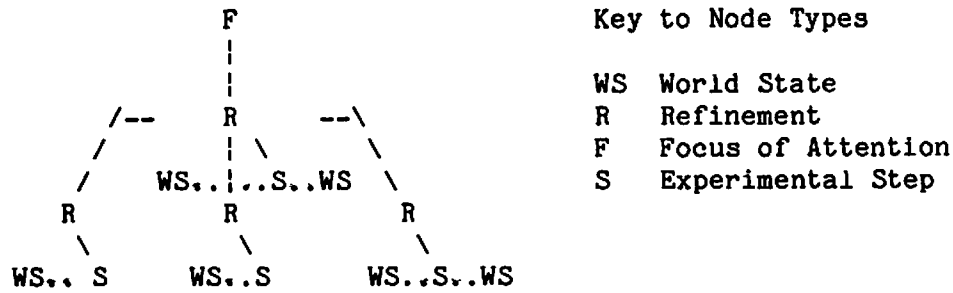

Figure 7. Fragment of Planning Network During Hierarchical Planning

In this figure, we see two levels of the refinement process. The focus node is in the focus plane, all of the refinement nodes are in the planning plane, and the other nodes are in the experiment plane. The top refinement node corrresponds to a general expression of the plan and the other refinement nodes are the next level of refinement to the design. Each refinement node corresponds to an experimental step with associated world states in the experiment plane. The entire operation of activating refinement rules and abstract action rules is under the control of a focus of attention rule in the focus node shown.

For hierarchical planning or any of the complex types of strategy that the system may be aware of, it is clear that the the schemata will be fairly complex. The interesting aspect of this is that the complexity is associated with the schema for the strategy. The schema itself may be used over and over again for each instance of that strategy when the domain specific information is added during acquisition of a rule. Much of the power of this approach is that when the schema is bug-free, a large number of instances of that strategy may be acquired and added to the knowledge base with confidence.

A factor that can complicate the structure of schemata and their rules is the handling of exceptional cases. The next section proposes some mechanisms for dealing with this.

V.4.4 Eliminating Special Cases

One of the motivations for using schemata to guide the acquisition

of rules is to simplify the acquisition of rules. The set of exceptional cases can potentially plague the statement of strategy rules in a system. This section gives an example of such an exceptional case and some mechanisms for stating the schemata for rules separately from their exceptions. It should be mentioned that the mechanisms mentioned in this section are somewhat tentative. Their purpose here is to illustrate some of the knowledge base management issues that have an impact on problem solving.

The MOLGEN knowledge base will contain a large number of refinement rules for different planning situations. Suppose that some of these refinement rules propose as subproblems the satisfaction of the preconditions of a given target rule. Let us presume further that this strategy for proposing subgoals is adequate for almost all situations with the following exception. When (1) the target rule is a domain rule for a restriction enzyme having a precondition for a somewhat basic pH, and (2) the DNA structures in the current world state are double stranded with a high percentage of adenine and thymine [8] (or if they are quite short), then these pH conditions will cause the structures to denature (become single stranded) and prevent the later successful application of the enzyme. In such cases the enzyme will fail to cleave the structures even though its preconditions are satisfied. Thus, some means for choosing an alternate restriction enzyme (or other cutting technique) needs to be employed.

There are several alternative places for the special case information in this example. Each alternative has its own impact on the amount of backtracking that has to be done, the complexity of the rules, the expense of their evaluation, and the management of the knowledge base. The following diagram illustrates the planning network for this example and will be referenced in the comparison of computational work and backtracking.

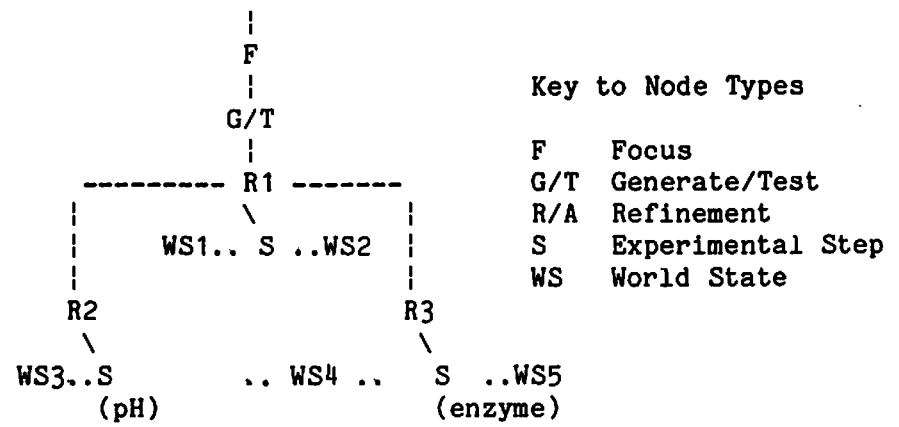

Figure 8. Planning Network for Enzyme/pH Example

The numbers after the world state nodes (WS) will distinguish them in the following discussion. We presume that the focus node, the generate/test node, WS1 and WS2 have been given as the problem

---

[8] Such structures are termed A-T rich. Their hydrogen bonding is weaker and they dissociate more readily in a low pH than A-T poor structures.

statement. WS1 expresses the initial state before application of the restriction enzyme and contains the description of the A-T rich DNA structures and other properties of the sample. WS2 contains a description of the desired output of this part of the plan. The top refinement node, R1, was created by the generator and points to the refinement rule in the discussion above. The experimental step associated with this refinement rule represents the abstract version of the experiment. The mapping rules in the top refinement rule have created the other refinement nodes R2 and R3. These nodes suggest no further refinements but are associated with their corresponding actions in the experiment plane -- for the pH step and the enzyme step respectively. When the state mapping in R1 is run, it will create WS3 (a particularized version of WS1). The action mapping will create the enzyme step. The refinement rule can then detect the unsatisfied precondition in the enzyme rule and create the pH step to satisfy it. When the pH step and enzyme steps are simulated, WS4 and WS5 are created.

There are four places where the special case information about the use of this enzyme with A-T rich structure might be incorporated.

1. In a pre-condition associated with the enzyme action rule.

2. In the Test rule.

3. In the Refinement rules (action mapping) for R1.

4. In a pH inspector.

The first option would prevent the selection of the enzyme by making it appear inapplicable (in the current world state -- WS1) to the refinement rule. The precondition could state that the enzyme was inapplicable to structures having AT-rich regions. However, this would also negate the possible use of this enzyme on a sample containing AT-poor structures with the AT-rich structures. Other experiments which might take advantage of the selective operation of this enzyme on the AT-poor structures would never be proposed. This option illustrates the motivation for the philosophy of stating the description of the action of a genetic tool separately from the criteria for its use.

The next option for placing the exception knowledge is in a higher level testing function for this part of the plan (ie. a test rule in a generate/test node.) The test rule would not have information specific to this special case, but would be able to detect the failure of the refinement by examining the results of the simulated steps. This approach maximizes the amount of backtracking required for this example and would proceed as follows. First, the pH precondition for the enzyme would be noticed by the refinement rule and proposed as a subproblem as though nothing was wrong. When the subgoal to establish a value for pH_9 is expressed to the pH schema, a procedure attached to the pH schema [9] would be activated and would carry out the denaturation process creating WS4. (It is important to note that this attached procedure would be just another rule in the rule knowledge base that happens to be activated by the subgoal mode of access to the schema for

---

[9] (This is a servant in Bobrow and Winograd's terminology.)

pH.) Finally, after denaturation and after the restriction enzyme was applied (ie. its abstract action was carried out), the test rule would be invoked. It would discover the discrepancy between WS5 and WS2 and report a failure to the focus node. This would initiate backtracking resulting eventually in the selection of an alternate choice for the enzyme.

Another option is to put this information in the refinement rule. This approach would minimize backtracking since the use of the particular enzyme would not be proposed in the network. To do this, the refinement rule must avoid proposing the use of the enzyme when it detected the AT-rich DNA in WS1. This mechanism suggests that the special cases for any of the enzymes that this refinement rule may propose as refinements must be incorporated into the rule. The repetition and dispersal of special case information requires some complications in the management of the knowledge base since one enzyme may appear in several different refinement rules and each refinement rule probably can utilize several different enzymes. Whenever the specifics of an enzyme are modified, it will be necessary to check for changes to all of the refinement rules which might reference it.

The fourth approach would be to associate the exception with pH itself. This approach extends the responsibility for a procedure attached to the pH schema. Instead of blindly carrying out the denaturation process, procedure attached to pH could inspect the current plan. It would find that the current structures would become denatured by this value for pH and also discover from the abstract world state (WS2) that this denaturation was not a desired or expected goal. We have called such attached procedures inspectors because of their role in inspecting global aspects of a developing plan. The inspector in this case would initiate backtracking immediately after the pH subgoal was proposed.

Although the last approach may seem more difficult, it has the advantage of associating special cases with the objects that cause them. In this case, the knowledge is not specific to the special case of our example, but is about rules which propose setting values for pH in plans when the denaturation effect was not anticipated. Thus any rule in the system which mentions pH invokes automatically this kind of checking. This includes all of the strategy rules and all of the domain rules. In this framework, rules which are based on simple schemata, for example our Means-ends analysis example, may actually invoke rather complicated behavior because of the inspectors associated with the objects that are mentioned. The factorization of the exceptions out to their associated objects follows the philosophy of the object-centered factorization of knowledge described in Section IV.3.3.1.

The MOLGEN knowledge will be able to accommodate each of the approaches to representing the special case knowledge in the example above. Different approaches will be best for different situations - depending on such things as the cost of backtracking, and the probability of certain situations.

Before leaving this subject of special cases, it is worth emphasizing some important points about attached procedures. In the first place, the inspectors and servants as discussed above are not

LISP procedures like the TEIRESIAS slot experts.[10] They are rules from the MOLGEN rule base. Since they can be strategy rules, they can perform any of the types of strategy operations, ie., they are not limited to initiating backtracking as in the example above. Finally, the concept of an inspector would be infeasible were it not for the fact the the planning network has been designed to be visible to strategy rules. As such, inspectors can work within the confines and types of communication available to other rules of their type that are invoked in the planning network.

V.5 Concluding Remarks

The thrust of this proposal is based on the contention that many of the ideas which have proved important for the acquisition and management of object knowledge may be extended to cover action and strategy knowledge as well.

Parallel to the schemata based rule knowledge base is the concept of expressing the dynamic knowledge of the problem solving process through schemata. This leads to the development of the concept of a planning network. This network provides a mechanism for expressing the problem solving state in terms of a small number of node types corresponding to basic problem solving steps used at all levels. The planning network idea, described in Section V.3, combines and extends the best elements of HEARSAY's blackboard, NOAH's procedural network, and schemata based representations.

The synergistic effect of these design elements creates the potential for a very exciting system. The same description of substructure which is used to decompose the acquisition process into small manageable steps makes possible the implementation of a sophisticated pattern matcher for choosing between actions or strategies. The schemata provide a framework where strategy knowledge can be expressed in terms of available standard strategy algorithms. This creates an available toolbox of problem solving techniques which can be instantiated with the particulars of domain specific knowledge. The planning network, which provides the language for strategy and focus of attention, also motivates the classification of strategy rules according to their basic steps. Perpendicular to this classification is one which is associated with the genetic knowledge.

As with any problem solving system, the success of the system will depend on the knowledge that it has available. The system's performance will depend on the strategy rules, domain rules, and object descriptions in the knowledge base. What is interesting about this proposed design for a system is the array of techniques proposed managing the knowledge base, acquiring the knowledge, and accessing it during problem solving. It is hoped that this flexible design will result in a powerful laboratory tool, so that MOLGEN can make important contributions to the practical design of interesting laboratory experiments.

---

[10] (Perhaps they should be termed "attached rules".)

Appendix I

Working Bibliography

Abbreviations

IJCAI Proceedings of the International Joint Conference on Artificial Intelligence held May 7-9 1969 in Washington D.C.

2IJCAI Proceedings of the Second International Joint Conference on Artificial Intelligence held at Imperial College, London September 1-3 1971. [Copies available from the British Computer Society, 29 Portland Place, London W1N 4AP England]

3IJCAI Proceedings of the Third International Joint Conference on Artificial Intelligence held at Stanford University, Stanford California, August 20-23, 1973 [Copies available from Stanford Research Institute Publications, 330 Ravenswood Ave, Menlo Park, California 94025]

4IJCAI Proceedings of the Fourth International Joint Conference on Artificial Intelligence held at Tbilisi, Georgia USSR, September 3-8, 1975. [Copies available from Publications Department, MIT A.I. Lab, 545 Technology Square, Cambridge, Massachusetts 02138]

AISB76 Proceedings of the AISB Summer Conference held at the University of Edinburgh July 12-14 1976. [Copies available from Department of Artificial Intelligence, University of Edinburgh, Forrest Hill, Edinburgh U.K.]

VLDB75 Proceedings of the International Conference on Very Large Data Bases held at Framingham, Massachusetts September 22-24 1975. [Copies available from ACM for $15 1133 Avenue of the Americas, New York, N.Y. 10036]

AIM A.I. Memo, Computer Science Department, Stanford, California

SRI Stanford Research Institute, Menlo Park, California

MIT Masschusetts Institute of Technology, Cambridge, M.A.

CMU Carnegie Mellon University, Pittsburgh, Pennsylvania.

[Aiello74] Aiello J.M., An Investigation of Current Language Support for Data Requirements of Structured Programming, MAC Technical Memo 51 (1974)

[Amarel69] Amarel S., Problem Solving and Decision Making by Computer: An Overview, in Garoin P.L. (ed.), Cognition: A Multiple View New York: Spartan Books (1969)

[Amarel68] Amarel S., On Representations of Reasoning About Actions, in Michie (ed.), Machine Intelligence 3, Edinburgh: Edinburgh University Press, pp 131-171 (1971)

[Anderson74] Anderson J.A., and Bower G.H., Human Associative Memory, John Wiley and Sons, New York (1974)

[Balzer73] Balzer R.M., "A Global View of Automatic Programming", 3IJCAI, pp 494-499 (1973)

[Becker70] Becker J.D., An Information-Processing Model of Intermediate-level Cognition, AIM 119 (1970)

[Belady75] Belady L.A., Lehman M. M., The Evolution and Dynamics of Large Programs, Report RC5615, IBM Research, Yorktown Heights

[Bobrow77a] Bobrow D.G., Winograd T., An Overview of KRL, a Knowledge Representation Language, to appear in Cognitive Science Vol. 1 No. 1 (Jan 1977)

[Bobrow77b] Bobrow D.G., Kaplan R.M., Kay M., Norman D.A., Thompson H., Winograd T., GUS, a Frame-Driven Dialog System, to appear in Artificial Intelligence (Spring 1977)

[Bobrow75a] Bobrow D.G., Collins A., Representation and Understanding: Studies in Cognitive Science, New York: Academic Press (1975)

[Bobrow75b] Bobrow D.G., Dimensions of Representation in [Bobrow75a] pp 1-34

[Bobrow75c] Bobrow D.G., Norman D.A., Some Principles of Memory Schemata in [Bobrow75a] pp 151-184

[Bobrow74] Bobrow D.G., Raphael B., "New Programming Languages for Artificial Intelligence," Computer Surverys, Vol. 6., No. 3 (September 1974)

[Bobrow73] Bobrow D.G., Wegbreit B., A Model and Stack Implementation of Multiple Environments, CACM Vol 16 No 10 (1973)

[Bruce72] Bruce B.C., A Model for Temporal References and its Application in a Question-Answering Program, Artificial Intelligence Vol. 3, pp 1-25 (1972)

[Buchanan69] Buchanan B.G., Sutherland G.L., Feigenbaum E.A., Heuristic DENDRAL: A Program for Generating Exploratory Hypotheses in Organic Chemistry, in Meltzer B. and Michie D. (eds.), Machine Intelligence 4, New York: American Elsevier Publishing Company, pp 121-157 (1969)

[Chamberlain76] Chamberlin D.D., Relational Data-Base Management Systems, ACM Computing Surveys Vol 8 No 1, pp 43-66 (March 76)

[Cheatham69] Cheatham T.E., Motivation for Extensible Languages, ACM SIGPLAN Notices Vol. 4 No. 8 pp 45-48 (August 1969)

[Codd70] Codd E.F., A Relational Model of Data for Large Shared Data Banks, CACM Vol 13 No 6 pp 377-387 (June 1970)

[Corey69] Corey E.J., Wipke W.T., Computer-assisted Design of Complex Organic Synthesis, Science, Volume 166 (178) [1969]

[Craik52] Craik K.J.W., The Nature of Explanation, Cambridge University Press (1952)

[Dahl72] Dahl O.J., Dijkstra E., Hoare C.A.R., Structured Programming, New York: Academic Press (1972)

[Date75] Date C.J., An Introduction to DataBase Systems, Addison-Wesley, Reading Massachusetts (1975)

[Davis76a] Davis R., King, J. "An Overview of Productions Systems", Machine Representations of Knowledge, (Proceedins of the 1975 Advanced Study Institute, Santa Cruz, CA). Also AIM-271

[Davis76b] Davis R., Buchanan B., Shortliffe E., "Production Rules as a Representation for a Knowledge based Consultation Program," Artificial Intelligence (to appear), also AIM-266

[Davis76c] Davis R., Applications of Meta Level Knowledge to the Construction, Maintenance and Use of Large Knowledge Bases, PhD Thesis Computer Science Department Stanford University (July 1976) Also AIM-283

[Deutsch75] Deutsch B.G., Establishing Context in Task-Oriented Dialogs, SRI Technical Note 114 (Sept 1975)

[Erman76] Erman L.D., Overview of HEARSAY Speech Understanding Research, ACM SIGART Newsletter, No. 6 pp 9-16 (Feb 1976)

[Erman73] Erman L.D., Fenell R.D., Lesser V.R., Reddy D.R., System Organizations for Speech Understanding: Implications of Network and Multiprocessor Computer Architecures for AI, in 3IJCAI pp 194-199 (1973)

[Ernst69] Ernst G.W., Newell A., GPS: A Case Study in Generality and Problem Solving, New York: Academic Press (1969)

[Eswaran75] Eswaran K.P., Chamberlain D.D., Functional Specification of a Subsystem for Data Base Integrity, in VLDB75 pp 48-67 (Sept 1975)

[Evans68] Evans T.G., A Program for the Solution of a Class of Geometric-Analogy Intelligence-Test Questions in [Minsky68] pp 271-351 (1968)

[Fahlman75] Fahlman S.E., A System for Representing and Using Real-World Knowledge, MIT AI MEMO 331 (May 1975)

[Feigenbaum63] Feigenbaum E.A., Feldman J. (eds.), Computers and Thought, New York: McGraw-Hill (1963)

[Feigenbaum68] Feigenbaum E.A., Artificial Intelligence: Themes in the Second Decade, Morrell A.H. (ed.), Information Processing 68, Amsterdam: New Holland Publishing Company, pp 1008-1022. Also AIM-67 (1968)

[Feigenbaum71] Feigenbaum E.A., et.al., "On Generality and Problem Solving", Machine Intelligence 6, pp 165-190, Edinburgh University Press, (1971)

[Fikes76a] Fikes R.E., "Knowledge Representation in Automatic Planning Systems", SRI Artificial Intelligence Center Technical Note 119, (January 1976)

[Fikes76b] Fikes R.E., "Deductive Retrieval Mechanisms for State Descripton Models," 4IJCAI, Vol 1, pp 99-106

[Fikes72a] Fikes R.E., Hart P.E., and Nilsson N.J., "Some New Directions in Robot Problem Solving," B. Meltzer and D. Michie (eds.), Machine Intelligence, Vol. 7, Edinburgh University Press, Edinburgh (1972)

[Fikes72b] Fikes R.E., Hart P.E., Nilsson N.J., "Learning and Executing Generalized Robot Plans," Artificial Intelligence, Vol. 3, No. 4, pp 251-288 (Winter 1972)

[Fikes71] Fikes R.E., and Nilsson N.J., "STRIPS: A New Approach to the Applications of Theorem Proving to Problem Solving," Artificial Intelligence, Vol. 2, pp 189-208 (1971)

[Fikes70] Fikes R.E., REF-ARF: A System for Solving Problems Stated as Procedures, Artificial Intelligence Vol. 1, pp 27-120 (1970)

[Findler71] Findler N.V., Meltzer R. (eds.), Artificial Intelligence and Heuristic Programming, New York: Elsevier Publishing Company, (1971)

[Flon74] Flon L., A Survey of Some Issues Concerning Abstract Data Types, CMU (Sept 1974)

[Floyd67] Floyd R., Nondeterministic Algorithms, JACM Vol 14 No 4 pp 636-644 (1967)

[Fry76] Fry J.P., Sibley E.H., Evolution of Data-Base Management Systems, ACM Computing Surveys Vol 8 No 1 pp 7-42 (March 1976)

[Galler74] Galler B., Extensible Languages, in Information Processing 74 published by Amsterdam: North Holland pp 313-316 (1974)

[Goldstein75] Goldstein I.P., Bargaining Between Goals, 4IJCAI, pp181-188 (1975)

[Green69] Green C., Theorem-Proving by Resolution as a Basis for Question-Answering Systems, in Meltzer B. and Michie D. (eds.), Machine Intelligence 4, New York: American Elsevier Publishing Company (1969)

[Green74] Green C.C., Waldinger R.J., Barstow D.R., Elschlager R., Lenat D.B., McCune B.P., Shaw D.E., Steinberg L.I., Progress Report on Program-Understanding Systems, AIM 240, August (1974)

[Hammer75] Hammer M.M., McLeod D.J., Semantic Integrity in a Relational Data Base System, in VLDB75 pp 25-68 (Sep 75)

[Minsky67] Minsky M., Computation: Finite and Infinite Machines, Englewood Cliffs: Prentice Hall (1967)

[Minsky61] Minsky M., Steps toward Artificial Intelligence, Proceedings of the Institute of Radio Engineers, Vol 4 Number 1 (Jan 1961) Also in [Feigenbaum63].

[Minsky74] Minsky M.A., A Framework for Representing Knowledge, in Winston P. (ed) The Psychology of Computer Vision, New York: McGraw-Hill (1975) (Also in MIT AI Memo 306 (June 1974))

[Model77] Model M.L., The Orthogonal Perspective Problem: A Problem for Representation Theory, personal communication (5 January 1977)

[Moore73] Moore J., Newell A., How can MERLIN understand, in Gregg L. (ed.), Knowledge and Cognition, Potomac, Maryland: Lawrence Erlbaum Associates (1973)

[Newell73] Newell A., Production Systems: Models of Control Structures, in Chase W.C. (Ed.), Visual Information Processing, pp 463-526, Academic Press: New York (1973)

[Newell72] Newell A., Simon H.A., Human Problem Solving, Prentice Hall (1972)

[Newell65] Newell A., Limitations of the Current Stock of Ideas about Problem Solving, Proceedings of a Conference on Electronic Information Handling, Kent A. and Taulbee O. (eds.), New York: Spartan, pp 195-208 (1965)

[Newell62] Newell A., Some Problems of Basic Organization in Problem-Solving Systems, in Yovitts M., Jacobi G.T., Goldstein G.D. (eds.) Self-Organizing Systems, New York: Spartan (1962)

[Newell59] Newell A., Shaw J.C., and Simon H.A., Report on a General Problem-Solving Program, in Proceedings of the International Conference on Information Processing (ICIP), pp 256-264, Paris: UNESCO House (June 1959)

[Newell56] Newell A., Simon H.A., The Logic Theory Machine: A Complex Information Processing System, IRE transactions on Information Theory, Vol IT-2, No 3, pp 61-79 (1956)

[Nilsson76] Nilsson N.J., "Some Examples of AI Mechanisms for Goal Seeking, Planning, and Reasoning", SRI Artificial Intelligence Center Technical Note 130 (May 1976)

[Nilsson74] Nilsson N.J., Artificial Intelligence, IFIP Congress held in Stockholm, Sweden August 5-10 1974. Also in SRI Artificial Intelligence Center Technical Note 89 (1974)

[Nilsson71] Nilsson N.J., Problem Solving Methods in Artificial Intelligence, McGraw-Hill (1971)

[Oyen76] Oyen R.A., Mechanical Discovery of Invariances for Problem Solving, Computer Engineering Department of Case Western Reserve, Cleveland, Ohio 44106

[Perlis69] Perlis A.J., Introduction to Extensible Languages, in ACM SIGPLAN Notices Vol 4 No 8 pp 3-5 (August 1969)

[Polya54] Polya G. How to Solve It, McGraw Hill, Princeton N.J. (1954)

[Pople75a] Pople H., Myers J.D., Miller R.A., DIALOG: A Model of Diagnostic Logic for Internal Medicine, 4IJCAI pp 848-855 (1975)

[Pople75b] Pople H., Artificial-Intelligence Approaches to Computer-based Medical Consultation, IEEE Intercon Conference (1975)

[Quinlan69] Quinlan J.R., A Task-Independent Experience-gathering Scheme for a Problem Solver, IJCAI pp 193-197 (1969)

[Raphael71] Raphael B., The Frame Problem in Problem-solving Systems in [Findler71], pp 101-124 (1971)

[Raphael68] Raphael B., SIR: Semantic Information Retrieval, in [Minsky68] pp 33-134 (1968)

[Reboh73] Reboh R., Sacerdoti E.D., "A Preliminary Qlisp Manual", SRI Artificial Intelligence Center Technical Note 81 (August 1973)

[Reddy73] Reddy D.R., Erman L.D., Fennell R.D., Eely R.B.N, "The HEARSAY Speech Understanding System: An Example of the Recognition Process", 3IJCAI, p185-193 (1973)

[Robinson68] Robinson J.A., New Directions in Mechanical Theorem Proving, in Morell A.J.H. (ed.), Information Processing 68, Amsterdam: North Holland Publishing Company, pp 63-67 (1968)

[Robinson65] Robinson J.A., A Machine-Oriented Logic Based on the Resolution Principle, J. ACM Vol 12 No 1, pp 23-41 (Jan 1965)

[Roussopoulos75] Roussopoulos N., Mylopoulos J., Using Semantic Networks for Data Base Management in VLDB75 pp 144-172 (Sept 1975)

[Rubin75] Rubin A.D., Hypothesis Formation and Evaluation in Medical Diagnosis, MIT AI-TR-316 (Jan 1975)

[Sacerdoti75a] Sacerdoti E.D., "The Nonlinear Nature of Plans", 4IJCAI, pp206-214, (also SRI Artificial Intelligence Center Technical Note 101) (1975)

[Sacerdoti75b] Sacerdoti E.D., A Structure for Plans as Behavior, Stanford Computer Science Department phD thesis. Also SRI Artificial Intelligence Center Technical Note 109 (August 1975)

[Sacerdoti73] Sacerdoti E.D., Planning in a Hierarchy of Abstraction Spaces, 3IJCAI, pp 412-422 (1973)

[Sandewall75] Sandewall E., Ideas about Management of LISP Data Bases, MIT AI Memo 332 (May 1975)

[Sandewall73] Sandewall E., Conversion of Predicate-Calculus Axioms, Viewed as Programs, to Corresponding Deterministic Programs, 3IJCAI pp230-234 (1973)

[Sandewall71] Sandewall E., Heuristic Search: Concepts and Methods, in [Findler71], pp 81-100 (1971)

[Schank76] Schank R.C., Abelson R.P., Scripts, plans, and Knowledge, 4IJCAI pp. 151-157, (1976)

[Schmidt76] Schmidt D.F., Sridharan N.S., Goodson J.L., Recognizing Plans and Summarizing Actions, in AISB76 pgs 291-306 (1976)

[Shortliffe76] Shortliffe E., MYCIN: Computer-based Medical Consultations, Ney York: American Elsevier (1976)

[Sibley76] Sibley E.H., The developement of Data-Base Technology, in ACM Computing Surveys, Vol 8 No 1 pp 1-5 (March 1976)

[Siklossy73] Siklossy L., Dreussi J., An Efficient Robot Planner Which Generates its Own Procedures, 3IJCAI, pp423-430 (1973)

[Simon73] Simon H.A., The Structure of Ill Structured Problems, Artificial Intelligence Journal, Vol 4, 1973, pp181-201

[Simon69] Simon H.A., The Science of Design and The Architecture of Complexity, in Sciences of the Artificial, MIT press (1969)

[Simon66] Simon H.A., On Reasoning about Actions, CMU Complex Information Processing Paper No. 87

[Simon63] Simon H.A., "Experiment with a Heuristic Compiler", JACM 10:4 pp 493-503 (October 1963)

[Sridharan76] Sridharan N.S., An Artificial Intelligence System to Model and Guide Chemical Synthesis Planning by Computer: A Proposal, Technical Report DCS-TR 43, Department of Computer Science Rutgers University, New Brunswick N.J. (1976)

[Sridharan76] Sridharan N.S., The Architecture of BELIEVER: A System for Interpreting Human Actions., Technical Report RUCBM-TR-46, Department of Computer Science, Rutgers University, New Brunswick N.J. (1975)

[Sridharan76] Sridharan N.S., The Architecture of BELIEVER - Part II. The Frame and Focus Problems in AI., Technical Report RUCBM-TR-47, Department of Computer Science, Rutgers University, New Brunswick N.J. (1976)

[Sridharan74] Sridharan N.S., A Heuristic Program to Discover Syntheses for Complex Organic Molecules, Proceedings of the IFIP74, (August 1974)

[Sridharan73] Sridharan N.S., Search Strategies for the task of Organic Chemical Synthesis, 3IJCAI, pp 95-104 (1973)

[Standish69] Standish T.A., Some Features of PPL, A Polymorphic Programming Language, ACM SIGPLAN Notices Vol 4 No 8 pp 20-26 (August 1969)

[Standish71] Standish T.A., PPL - An Extensible Language that Failed, ACM SIGPLAN Notices Vol 6 No 12 pp 144-145 (Dec 1971)

[Sussman74] Sussman G.J., The Virtuous Nature of Bugs, Proceedings of the AISB Summer Conference (July 1974)

[Sussman73] Sussman G.J., "A Computational Model of Skill Acquisition", MIT Technical Note AI TR-297 (August 1973)

[Sussman72] Sussman G.J. and McDermott D.V., "Why CONNIVing is Better than PLANNing", MIT 255A (April 1972)

[Tate75] Tate A., Interacting Goals and Their Use, 4IJCAI, p215-218 (1975)

[Tate74] Tate A., "INTERPLAN: A Plan Generation System which can deal with Interactions between Goals," Memorandum MIP-R-109, Machine Intelligence Research Unit, University of Edinburgh (December 1974)

[Taylor76] Taylor R.W., Frank R.L., CODASYL Data-Base Management Systems, ACM Computing Surveys Vol 8 No 1 pp 67-103 (March 1976)

[Trigoboff76] Trigoboff M., Propagation of Information in a Semantic Net in AISB76, pp 334-343 (1976)

[Tsichritzis76] Tsichritzis D.C., Lochovsky F.H., Hierarchical Data-Base Management: A Survey, in ACM Computing Surveys Vol 8 No 1 pp 105-123 (March 1976)

[Waldinger75] Waldinger R., Achieving Several Goals Simultaneously, SRI Technical Note 107 (July 1975)

[Warren76] Warren D.H.D., Generating Conditional Plans and Programs, in AISB76, pp 344-354 (1976)

[Warren74] Warren D.H.D., "WARPLAN: A System for Generating Plans," Memorandum No. 76, Department of Computational Logic, University of Edinburgh (June 1974)

[Waterman70] Waterman D., Generalization Learning Techniques for Automating the Learning of Heuristics, Artificial Intelligence Vol 1, pp 121-170 (1970)

[Wegbreit71] Wegbreit B., An Overview of the ECL Programming System, ACM SIGPLAN Notices Vol 6 No 12 pp 26-28 (Dec 1971)

[Wickelgren74] Wickelgren W.A., How to Solve Problems, W.H. Freeman, San Francisco (1974)

[Wiederhold77] Wiederhold G., Database Design, to be published by McGraw-Hill (1977)

[Wiederhold] Wiederhold, G.,Data Base Structure and Schemas, in preparation, partially available as class notes for MIS290, USCF

[Winograd75a] Winograd T., "Breaking the Complexity Barrier, Again,", SIGPLAN notices, (Jan 1975)

[Winograd75b] Winograd T., Frame Representations and the Procedural/Declarative Controversy" in [Bobrow75a], pp 185-210

[Winograd74] Winograd T., Five Lectures on Artificial Intelligence, AIM-246 (Sept 1974)

[Winograd72] Winograd T., Understanding Natural Language, Academic Press (1972)

[Winston70] Winston P.H., Learning Structural Descriptions from Examples, MAC TR-76, MIT (September 1970)

[Wipke76] Wipke W. T., SECS -- Simulation and Evaluation of Chemical Synthesis: Strategy and Planning, in Proceedings of the Symposium on Computer-Assisted Organic Synthesis Planning" held by the American Chemical Society (April 6-9 1976)

[Wipke73] Wipke W.T., Computer-Assisted Three Dimensional Synthetic Analysis, in Computer Representation and Manipulation of Chemial Information, W.T. Wipke et. al. (eds.), John Wiley (1974)

[Wirth71] Wirth N., Program developement by Stepwise Refinement, CACM Vol. 14 pp 221-227 (1971)

[Woods75] Woods W.A., What's in a Link: Foundations for Semantic Networks, in [Bobrow75a] pp 35-82 (1975)

[Zloof75] Zloof, M.M., Query by Example, in AFIPS National Computer Conference Proceedings, Vol 44 pp 431-437 (1975)