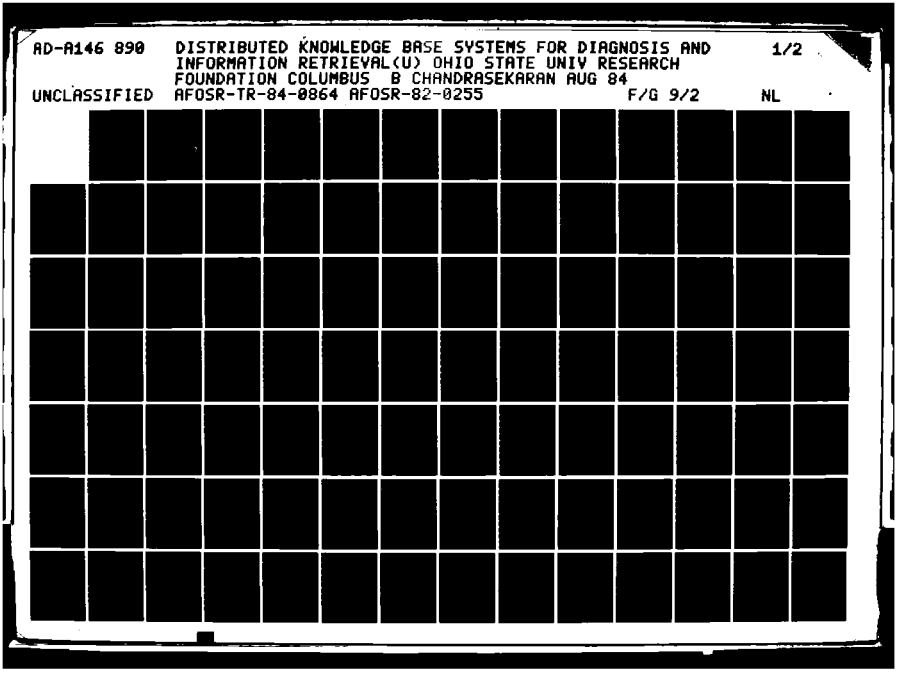
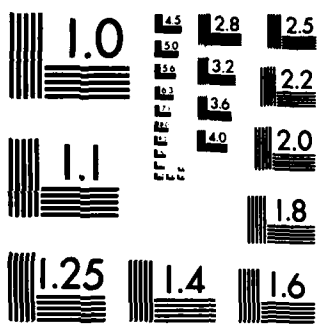
AD-A146 890 DISTRIBUTED KNOWLEDGE BASE SYSTEMS FOR DIAGNOSIS AND INFORMATION RETRIEVAL(U) OHIO STATE UNIV RESEARCH FOUNDATION COLUMBUS B CHANDRASEKARAN AUG 84 1/2

UNCLASSIFIED AFOSR-TR-84-0864 AFOSR-82-0255 F/G 9/2 NL





(5)

AFOSR-TR- 84-0864

AD-A146 890

RF Project 763180/714659
Annual Report

# DISTRIBUTED KNOWLEDGE BASE SYSTEMS FOR DIAGNOSIS AND INFORMATION RETRIEVAL

B. Chandrasekaran
Department of Computer and Information Science

For the Period
July 1, 1983 - June 30, 1984

DEPARTMENT OF THE AIR FORCE
Air Force Office of Scientific Research
Bolling Air Force Base, D.C. 20332

Grant No. AFOSR-82-0255





August, 1984

The Ohio State University
Research Foundation
1314 Kinnear Road
Columbus, Ohio 43212

# REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION<br>UNCLASSIFIED | 1b. RESTRICTIVE MARKINGS |
| 2a. SECURITY CLASSIFICATION AUTHORITY | 3. DISTRIBUTION/AVAILABILITY OF REPORT<br>Approved for public release; distribution unlimited. |
| 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S)<br>763180/714659 | 5. MONITORING ORGANIZATION REPORT NUMBER(S)<br>AFOSR-TR- 84-0864 |
| 6a. NAME OF PERFORMING ORGANIZATION<br>Ohio State University<br>6b. OFFICE SYMBOL (If applicable) | 7a. NAME OF MONITORING ORGANIZATION<br>Air Force Office of Scientific Research |
| 6c. ADDRESS (City, State, and ZIP Code)<br>Department of Computer and Information Science, Columbus OH 43212 | 7b. ADDRESS (City, State, and ZIP Code)<br>Directorate of Mathematical & Information Sciences, AFOSR, Bolling AFB DC 20332 |
| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION<br>AFOSR<br>8b. OFFICE SYMBOL (If applicable)<br>NM | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER<br>AFOSR-82-0255 |
| 8c. ADDRESS (City, State, and ZIP Code)<br>Bolling AFB DC 20332 | 10. SOURCE OF FUNDING NUMBERS<br>PROGRAM ELEMENT NO. 61102F; PROJECT NO. 2304; TASK NO. A7; WORK UNIT ACCESSION NO. |

11. TITLE (Include Security Classification)
DISTRIBUTED KNOWLEDGE BASE SYSTEMS FOR DIAGNOSIS AND INFORMATION RETRIEVAL

12. PERSONAL AUTHOR(S)
B. Chandrasekaran

| 13a. TYPE OF REPORT | 13b. TIME COVERED | 14. DATE OF REPORT (Year, Month, Day) | 15. PAGE COUNT |
|---|---|---|---|
| Interim | FROM 1/7/83 TO 30/6/84 | AUG 84 | 119 |

16. SUPPLEMENTARY NOTATION

| 17. COSATI CODES | | |
|---|---|---|
| FIELD | GROUP | SUB-GROUP |
| | | |

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Expert systems; diagnostic reasoning; deep models; functional representations; expert systems for design; qualitative simulation; the CSRL language.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

During the year progress was made in a number of directions: (1) The investigators developed in significant detail a language for representing an agent's ~~understanding~~ of aspects of how a device works, and also developed a compiler which can produce a diagnostic expert problem solving system from this deep level functional representation. (2) The researchers continued their investigation of how design knowledge can be organized as plans and design problem solving can be viewed as design refinement by plan selection and re-design. They have completed the construction of a prototype design expert system called AIR-CYL, which designs a moderately complex mechanical component called an aircylinder for a range of specifications. (3) They continued investigation of high-level languages for expert system construction; in particular they have refined their design of the CSRL language for diagnostic expert system, and implemented it in Interlisp for the Xerox family of Lisp machines. (4) They have initiated a new investigation in reasoning about the behavior of physical systems by qualitative simulation by using a novel technique. (CONTINUED

| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT | 21. ABSTRACT SECURITY CLASSIFICATION |
|---|---|
| ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS | UNCLASSIFIED |

| 22a. NAME OF RESPONSIBLE INDIVIDUAL | 22b. TELEPHONE (Include Area Code) | 22c. OFFICE SYMBOL |
|---|---|---|
| Dr. Robert N. Buchal | (202) 767- 4939 | NM |

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

ITEM #19, ABSTRACT, CONTINUED: called consolidation, which infers the behavior of a composite component from the behaviors of its subcomponents.

RF Project 763180/714659
Annual Report

# DISTRIBUTED KNOWLEDGE BASE SYSTEMS FOR DIAGNOSIS AND INFORMATION RETRIEVAL

B. Chandrasekaran
Department of Computer and Information Science

For the Period
July 1, 1983 - June 30, 1984

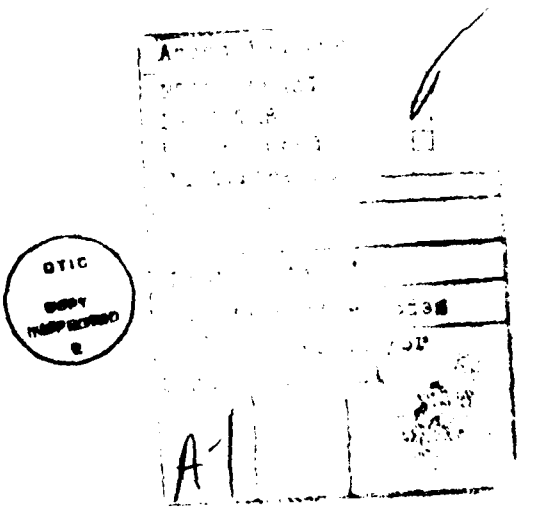
DEPARTMENT OF THE AIR FORCE
Air Force Office of Scientific Research
Bolling Air Force base, D. C. 20332

Grant No. AFOSR-82-0255

August, 1984

## Table of Contents





## 1. Technical Progress During Period July 1, 1983 - June 30, 1984

This report period covers the second year of our research program on the foundations of knowledge-based reasoning, with particular reference to diagnostic, design and information retrieval tasks. We will proceed in this interim report by first giving an outline of our general approach, then briefly describe some problems in which we have made progress during the year. We will enclose as appendices some papers which further describe some of our accomplishments in greater detail. These papers typically have either appeared in the literature during the year or will shortly appear.

## 2. Our Conceptual Approach

The primary focus of our work is on analyzing knowledge-based problem solving in information processing terms, i.e., in terms which highlight what kinds of information is being input to a reasoning process and is being output by it, rather than in terms of the tools which are used to implement an expert system. Analysis in terms of the latter will typically talk in terms of "rule-based system"s, "frame-based systems," or "prolog systems," etc., while we would talk about the nature of the tasks themselves: "the classificatory task," the "abduction task," etc. The latter level of analysis enables us to isolate the terms and relations that characterize the essence of the knowledge structures that are needed to perform the variety of tasks that knowledge-based reasoning is capable of handling.

Our emphasis on the close relationship between the knowledge structures and the information processing for various tasks has increasingly taken us away from the earlier generation of expert system theories which separate the knowledge base from the inference engine. Our systems tend to be highly organized symbolic structures built up of active specialized knowledge-using agents.

The search for the fundamental types of knowledge-based problem solving has

led us to think in terms of generic information processing tasks. The basic idea is that each fundamental type of problem solving activity accomplishes a certain generic task, and has its own characteristic way of using knowledge. A grasp of the "atoms" of intelligent information processing should provide the basic building blocks out of which more complicated forms of intelligent problem-solving can be built.

We have identified several such generic tasks from our work on medical reasoning and reasoning about mechanical devices. Diagnostic (classificatory) problem solving, a particular form of predictive reasoning, a form of knowledge-directed data retrieval, and a form of design activity are examples of distinct problem solving types. We make no claim that these types are exhaustive, in fact as our research proceeds we expect that we will identify more generic types.

This theory of generic types of problem solving is discussed more fully in [6]. The basic idea is that a complex task is broken down into a number of generic subtasks, and each subtask is then solved by an appropriately organized community of specialists. That is, the knowledge structure corresponding to a problem solving type can be decomposed into a number of specialists who cooperate in solving that class of problems. We have developed approaches for a number of problems based on this overall approach. Our work on these theoretical ideas is presented in [8, 1] and [5].

Recently we have been concerned with developing deep models of expert reasoning. Most of the diagnostic systems that have been developed in medicine and other domains have been called "compiled" or "shallow" knowledge systems, pointing out that the knowledge base encodes in a fairly direct way the relationships between findings and hypotheses. Yet often a human expert's knowledge of how the device functions is used to generate new relationships during the reasoning process. So far we have developed a primitive language

for representing the functioning of devices, and a "compiler" capable of building a diagnostic problem-solver from a device representation made in this language [7, 10].

We made progress on a number of problem areas during the year under report. We outline the projects and the basic ideas in the next few sections. As mentioned earlier, we have also added appendices consisting of papers written by us during the year which give the research progress in greater detail.

## 3. Project Progress Reports

### 3.1. A Method for Representing the Functional Organization of Devices which Supports the Automatic Compilation of Diagnostic Systems

V. Sembugamoorthy and B. Chandrasekaran

Human experts often use in their problem solving a deeper understanding of their knowledge domain than has been captured in the first generation of expert systems. We have developed a representation for one aspect of this deeper knowledge, corresponding to an expert's understanding of how the functioning of a complex device results from its structural properties. We have built a compiler which automatically generates a diagnostic expert system from this functional representation of a device [10].

The first idea is that an agent's understanding of how a device works is organized as a representation that shows how an intended function is accomplished as series of behavioral states of the device, and how each behavior state transition can be understood as either due to a function of a component, or in terms of further details of behavior states. This can be repeated at several levels so that ultimately all of the functions of a device can be related to its structure and the functionality of the components in the structure. For example, the function that we may call "buzz" of a household electric buzzer may be represented as:

```
FUNCTION:  Buzz : TOMAKE buzzing(buzzer) IF pressed (switch)*
                  by behaviorl
```

and the relevant behavior, behaviorl, can be represented as:

```
BEHAVIOR: behaviorl:

  Pressed(switch)*
         |
         | BY behaviorl
         V
{Clapper electrical connection alternates}
         |
         | USING-FUNCTION mechanical OF
         | clapper
         V
  Repeated-Hit(Clapper)
         |
         | USING-FUNCTION electrical OF
         | clapper
         V
  Buzzing(Clapper)
        |||
        |||
  Buzzing(Buzzer)
```

Intuitively what is being said here is that the Buzz function is accomplished when, if the switch is pressed, the buzzer goes to a state called "buzzing," and this is accomplished by a series of behavioral states that is named behaviorl. Behaviorl says that the buzzer, on the occasion of the switch being pressed, goes to a state where the electrical connections in the clapper alternately close and open, which results in the state where the clapper is repeatedly hit, which results in the buzzer being in the state of buzzing. Each transition is further explained, either in terms of further details in the state transition, or in terms of the functions of the components. For example, the transition from the clapper being alternately electrically connected and disconnected, to its being in the repeated-hit state, is explained by relating it to the mechanical function of the clapper.

Let us see how this fragment of functional representation can be used to generate a piece of diagnostic knowledge that may be used by a diagnostic expert system. A diagnostic compiler will function as follows. Suppose a buzzer does not buzz when its switch is pressed. In order to find out what malfunctions are causing this, the diagnostic compiler will reason thus on the basis of the functional specification and the behaviorl specification: The functional specification tells it that the problem is in behaviorl, since the Buzz function is failing. Behaviorl, on examination, can result in a series of hypotheses:

R1: If switch is pressed, but the clapper is not alternately electrically connected and disconnected, problem is in behavior2.

R2: If switch is pressed, the clapper's electrical connectivity alternates, but the clapper doesn't hit-repeatedly, the cause of buzzer not buzzing is mechanical malfunction of the clapper.

The power of this method for representing how a device works is due in large measure to explicitly distinguishing five aspects of an agent's understanding of the device, and treating each aspect appropriately. The distinctions are made at every level of organization on which the device is represented. The five aspects are:

- **STRUCTURE** - this specifies the relationships between components.
- **FUNCTION** - this captures the intended purpose of a device or component, specified as WHAT the response is to a stimulus.
- **BEHAVIOR** - this specifies HOW, given a stimulus, the response is accomplished.
- **GENERIC KNOWLEDGE** - chunks of deeper causal knowledge that have been compiled from various domains to enable the specification of behavior.
- **ASSUMPTIONS** - other specifications of the conditions under which various behaviors or conditions occur.

Directions for future research include the following: We need to develop methods to check the correctness/consistency of a given device representation. We need to investigate the design of two other needed dimensions of device representation, namely the temporal dimension and interactions of functional units by way of feedback and communication. Also the causal dimension, which we have discussed, has to be integrated with the other two in a disciplined, practically useful, and cognitively meaningful framework. We need to identify the compilation processes that come into play to generate other types of expert problem solving structures, such as those that can predict the functional and behavioral consequences of changes of structure.

In broader terms, This work is part of our on-going effort to uncover the multiplicity of generic structures and processes involved in knowledge-based problem solving. Whether or not one accepts the hypothesis that homogeneous and unitary architectures such as production systems are adequate at the level of symbol processing in the mind, we nevertheless believe that in order to account for knowledge-based problem-solving activity at the information processing level, there is a need to identify a richer collection of generic knowledge structures and a correspondingly rich collection of knowledge-processing mechanisms that operate on them.

We enclose as Appendix A a paper that describes the problem and our approach in greater detail. This paper is being prepared for submission to journals, and is an elaboration of a paper presented at the 1983 Joint Services Workshop on AI Applications to Diagnosis and Maintenance that was held in Boulder, Col.

### 3.2. Expert Systems for Design Problem-Solving using Design Refinement with Plan Selection and Redesign

David C. Brown and B. Chandrasekaran

This research is concerned with the design of mechanical components, and

views design as a problem-solving activity. The theory explains the activity of a human designer when solving a problem that falls into a particular subclass of mechanical design. An expert system called AIR-CYL has been implemented that embodies the theory. The system will design a particular type of Air-cylinder according to some set of user given requirements. The behavior of the system closely follows that of the human designer.

Design activity in general has many components; such as planning, the use of prestored plans, refinement of descriptions and the use of large amounts of knowledge. Not all designing involves all of these. We have established three classes of design activity which vary according to their problem-solving components. Our work refers only to the third class, which requires that at every stage of the design the designer knows both what sequences of design steps are appropriate and also what knowledge is required. The theory hypothesizes that such activity is organized around a hierarchy of concepts, where *each concept is active in the design*, and may be considered to be a specialist about some portion of the design. The hierarchy reflects the way that the designer thinks of the object during design, and it shapes the design process.

Each Specialist has its own set of Plans from which to select depending on the current stage of the design. The plans may request portions of the design from other specialists lower in the hierarchy, or may use Tasks to make small additions to the design itself. Tasks use Steps to decide the value of each attribute for which it is responsible. For example, a hole might be designed by a Task, while a Step would decide the radius. Constraints may be planted at any point in order to test the validity of the design. The Design Database contains the current state of the design and a record of its progress, plus the collected requirements from the user. Each task suggests changes to the design, and when it is satisfied that they are locally coherent it produces an Update which alters the state of the design.

The complete design process proceeds by first obtaining and checking requirements for consistency. It then does rough-design to establish whether full design is worth pursuing. If the rough-design succeeds, then the full design is attempted by requesting a design from the top-most specialist. The rough-design hierarchy consists of only the upper portion of the design hierarchy. Communication between active design agents is done by passing messages that give instructions and report on success or failure. This is the only way that a specialist can know what has occurred at lower levels.

If a step fails due to a failing Constraint a redesign phase is entered until the problem can be fixed and design can continue. Step failure can lead to task failure and subsequently to specialist failure. The redesign process is controlled by Suggestions about what might fix the problem. These suggestions are produced by each agent that fails. Suggestions are examined by a redesign strategy in the agent immediately above the failing agent. Appropriate measures are then taken to follow the suggestions in order to correct the problem. Different types of agents have different strategies. This results in backing-up over prior design decisions in a manner which is dependency-based by suggestion. The change/update mechanism of the design data-base supports the redesign phase.

To facilitate the building of the AIR-CYL system, and class 3 design problem-solvers in general, a language has been provided in which to declare design agents and describe plans. The Design Specialists and Plans Language (DSPL) has been used to capture the Air-cylinder design knowledge. The system takes about 5 minutes to design an Air-cylinder given 20 requirements.

We have presented an approach to building expert systems for a particular class of design activity in the domain of mechanical components. Much work remains to be done in this area before we fully understand what design is and how best to build systems to do it. However we feel that by using an

hierarchically structured system of conceptual specialists with plan selection and failure handling we have captured the essential qualities of routine design, while discovering many interesting and difficult issues.

We enclose as Appendix B a paper [2] that describes our approach to design problem solving in greater detail. This paper will appear in the proceedings of the IFIP WG5.2 Conference on Knowledge Engineering in Computer-Aided Design, Budapest, Hungary.

### 3.3. CSRL: A Language for Designing Diagnostic Expert Systems

Tom Bylander, B. Chandrasekaran, Sanjay Mittal[1], and Jack W. Smith

Many kinds of problem solving for expert systems have been proposed within the AI community. Whatever the approach, there is a need to acquire the knowledge in a given domain and implement it in the spirit of the problem solving paradigm. Reducing the time to implement a system usually involves the creation of a high level language which reflects the intended method of problem solving. For example, EMYCIN was created for building systems based on MYCIN-like problem solving. Such languages are also intended to speed up the knowledge acquisition process by allowing domain experts to input knowledge in a form close to their conceptual level. Another goal is to make it easier to enforce consistency between the expert's knowledge and its implementation.

CSRL (Conceptual Structures Representation Language) is a language for implementing expert diagnostic systems that are based on our approach to diagnostic problem solving [3]. In this approach, diagnostic reasoning is one of several generic tasks, each of which calls for a particular organizational and problem solving structure. This approach is an outgrowth of our group's

---

[1]Currently at Knowledge Systems Area, Xerox PARC, 3333 Coyote Hill Rd., Palo Alto, CA 94304 USA

experience with MDX, a medical diagnostic program, and with applying MDX-like problem solving to other medical and non-medical domains.

A diagnostic structure is composed of a collection of specialists, each of which corresponds to a potential hypothesis about the current case. They are organized as a classification or diagnostic hierarchy, e.g., a classification of diseases. A top-down strategy called establish-refine is used in which either a specialist establishes and then refines itself, or the specialist rejects itself, pruning the hierarchy that it heads.

CSRL facilitates the development of diagnostic systems by supporting constructs which represent diagnostic knowledge at appropriate levels of abstraction. Message procedures describe the specialist's behavior in response to messages from other specialists. Knowledge groups determine how data relate to features of the hypothesis. Rule-like knowledge is contained within knowledge groups. See [4] for a discussion on encoding diagnostic knowledge in CSRL in the medical domain.

We have used CSRL in the implementation of two expert systems. Auto-Mech is an expert system which diagnoses fuel problems in automobile engines [12]. It consists of 34 specialists in a hierarchy which varies from 4 to 6 levels deep. Red is an expert system whose domain is red blood cell antibody identification [11]. CSRL is used to implement specialists corresponding to each antibody that Red knows about (around 30 of the most common ones) and to each antibody subtype.

Appendix C is a paper that will appear in International Journal of Computers in Mathematics, special issue on AI applications. This describes the CSRL language in greater detail. CSRL has been implemented in two environments: UCI-Rutgers Lisp in Tops 20 for the DEC20 series, and for the Interlisp/Loops environment that runs on the Xerox family of Lisp machines.

## 3.4. Using Consolidation for Reasoning about the Behavior of Physical Systems

Tom Bylander

A recent AI approach for reasoning about the behavior of physical systems is qualitative simulation. The structure of the physical system, and knowledge about the behavior of its components are used to derive a collection of constraints. Using these constraints, the simulation is performed and its results are interpreted.

This research investigates a new method of reasoning for this problem which we call consolidation.

The major processing sequence of consolidation is to hypothesize a composite component consisting of a selected subset of components, and then to infer the behavior of the composite from the behaviors of the components. Successful application of this sequence on increasingly larger composite components results in inferring the behavior of the whole system. As a byproduct, a hierarchical behavior structure is produced which explains how the overall behavior is caused by the components' behavior. Also note that each reasoning step is localized over a small number of components and subsystems, avoiding the global problem solving required for qualitative simulation.

This research also proposes a novel representation for behavior. Current theories describe behavior as arithmetic constraints on variables and their derivatives, which would imply that consolidation is purely a matter of mathematical manipulation. Instead, we describe the behavior of a component by the actions that the component performs upon "substances," e.g., fluids, electric currents, control activations, or other stuff that can potentially move. We claim that there is a small set of behavior schema which can directly represent these actions, and which allow inferences about the behavior of composite components. Example schema include: permitting a

substance to move from one place to another, and influencing a substance to move. A behavior can be hypothesized based on patterns of other behaviors. Its existence is confirmed, and its parameters are determined using knowledge about the physics of the substance being acted upon. Consolidation controls the inference of behavior by specifying the context (the composite component) in which inference can take place.

We are implementing a version of consolidation, which will depend upon a few simplifying assumptions. The structural description will be limited to connection of components and containment of substances, thus reducing the amount of spatial reasoning required. Numerical attributes of behaviors (such as amount of influence or rate of movement) will be specified qualitatively. The qualitative language used will be similar to that developed by Kuipers [9], and will also include a simple temporal component for expressing the sequence of events. We hope to discover the limits of consolidation under these assumptions, and to learn how more complex spatial and temporal reasoning can be integrated into this process.

## 4. Computing Environment

We are also in receipt of another AFOSR grant (Grant AFOSR 83-0300), under the DOD-University Research Instrumentation Program, for a Lisp machine facility for expert systems research. During the year under report, we acquired for Xerox 1108 Lisp machines, which are currently connected in an ethernet configuration with each other and with a VAX 11/780. Much of the research for the AFOSR research program is moving to this environment from the earlier DEC20/60 UCI-Rutgers Lisp one. During the year, we implemented the CSRL language (see section 3.3) in this new environment, and have used it to implement a number of prototype diagnostic systems.

## References

[1] Brown, D.C. / Chandrasekaran, B.
An Approach to Expert Systems for Mechanical Design.
In Proc. Trends & Applications '83, pages 173-180. IEEE Computer Society, Gaithersburg, MD, May, 1983.

[2] Brown, D.C. / Chandrasekaran, B.
Expert Systems for a Class of Mechanical Design Activity.
1984
Paper for IFIP WG5.2 Working Conference, Sept. 84.

[3] T. Bylander, S. Mittal, and B. Chandrasekaran.
CSRL: A Language for Expert Systems for Diagnosis.
In IJCAI-83. 1983.

[4] T. Bylander, and J. W. Smith.
Using CSRL for Medical Diagnosis.
In Proc. Second International Conf. on Medical Computer Science and Computational Medicine. 1983.

[5] Chandrasekaran, B.
Natural and Social Metaphors for Distributed Problem Solving: Introduction to the Issue.
IEEE Transactions on Systems, Man, and Cybernetics SMC-11(1):1-5, Jan, 1981.

[6] Chandrasekaran, B.
Towards a Taxonomy of Problem-Solving Types.
AI Magazine 4(1):9-17, Winter/Spring, 1983.

[7] Chandrasekaran, B. / Mittal, S.
Deep Versus Compiled Approaches to Diagnostic Problem Solving.
International Journal of Man Machine Studies 19:425-436, 1983.

[8] Gomez, F. / Chandrasekaran, B.
Knowledge Organization and Distribution for Medical Diagnosis.
IEEE Transactions on Systems, Man, and Cybernetics SMC-11(1):34-42, January, 1981.

[9] Kuipers, B.
Commonsense Reasoning about Causality: Deriving Behavior from Structure.
Technical Report 18, Working Papers in Cognitive Science, Tufts University, 1982.

[10] Moorthy, V.S. / Chandrasekaran, B.
Functional Representation of Devices and Compilation of Diagnostic Problem Solving Systems.
August, 1984
The Ohio State University.

[11] J. W. Smith, J. Josephson, C. Evans, P. Straum, and J.Noga.
Design For a Red-Cell Antibody Identification Expert.
In Proc. Second International Conf. on Medical Computer Science and Computational Medicine. 1983.

[12] M. C. Tanner and T. Bylander.
Application of the CSRL Language to the Design of Expert Diagnosis Systems: The Auto-Mech Experience.
In Proc. of the Joint Services Workshop on Artificial Intelligence in Maintenence. 1984.

# APPENDIX A

## FUNCTIONAL REPRESENTATION OF DEVICES AND COMPILATION OF DIAGNOSTIC PROBLEM SOLVING SYSTEMS

# FUNCTIONAL REPRESENTATION OF DEVICES AND COMPILATION OF DIAGNOSTIC PROBLEM SOLVING SYSTEMS

V. Sembugamoorthy and B. Chandrasekaran
Artificial Intelligence Group
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210

August 1984

Acknowledgement: This research was supported by National Science Foundation grant MCS-8305032 and by Air Force Office of Scientific Research grant AFOSR 82-0255.

## Table of Contents



## List of Figures

# FUNCTIONAL REPRESENTATION OF DEVICES AND

# COMPILATION OF DIAGNOSTIC PROBLEM SOLVING SYSTEMS

V. Sembugamoorthy and B. Chandrasekaran
Artificial Intelligence Group
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210

## ABSTRACT

Most of the diagnostic systems that have been developed in medicine as well as other domains can properly be called "compiled" knowledge systems in the sense that the knowledge base contains the relationships between symptoms and malfunction hypotheses in some form. However, often in human reasoning, an expert's knowledge of how the device "functions" is used to generate new relationships during the reasoning process. This deeper level representation which can be processed to yield more compiled diagnostic structures is the concern of this paper. Using the example of an household buzzer, we show in this paper what our functional representation of a device looks like. We discuss the nature of the compilation process that can produce the diagnostic expert from this deeper representation. We also outline how another form of problem solving, viz., predicting consequences to device functionality of changes in the structure of a device, can also be supported by this representation.

## 1. Motivation

The work to be described in this paper can be motivated by reference to a number of issues that have recently attracted attention in knowledge-based reasoning. Three of them are as follows:

1. What does it mean to understand how a device works; in particular,

to understand how its function is related to and arises from its structure? How to represent the result of this understanding in such a way that this representation can be used to support problem solving, such as trouble-shooting a malfunctioning device, or to predict consequences to device functionality of changes in its structure? With human beings at least, it seems reasonable to expect a person who claims to understand how a device works to be able to engage in the above forms of problem solving tasks. Some recent work in artificial intelligence [5, 6, 11] deals with these issues. In the context of this set of research issues, our goal in this paper is to present a framework for approaching this problem, to describe a language for representing functioning of a class of devices, and to show how this representation can support problem solving of the types mentioned above.

2. A related concern, especially in the literature on systems for medical diagnosis, has been on causal reasoning. Typically this has meant representing detailed causal relationships between pathophysiological states that underlie a disease process and using this information to make conclusions about disease entities, given symptomatic information. The work in [13] and [14] exemplify this approach. Such representations have been called deep models [9, 12], in contrast to systems, such as MYCIN, whose knowledge base contains the evidentiary relationships between disease hypotheses and symptoms directly, without specifying the causal pathways between them. From this viewpoint, the functional representation advanced in this paper can be thought of as a proposed form of deep model for diagnostic expert systems. It can yield causal chains of behavioral states at several levels of detail, and can also generate evidentiary relationships mentioned above to the extent that they are derivable from causal models.

3. Related to the above notion of a deep model of a domain is the idea of compiling from it different forms of knowledge structures useful for different kinds of reasoning [1]. For example, for diagnostic reasoning, we need malfunction hypotheses and pieces of knowledge that relate symptoms to these hypotheses (the evidentiary knowledge of the previous paragraph); for reasoning about consequences of actions that may be performed on a system, we need knowledge that relates state changes at different levels of system description. The intuition is that an agent with an appropriate deep model can generate from it knowledge in these forms, and then use them directly for the relevant problem solving task. An adequate deep model can give rise to different compiled structures for different tasks. If the tasks are generic in some sense, then one might look for compilation processes which are device-independent. (A theory of generic types of knowledge-based problem solving tasks is developed in [2].) In this perspective, the work to be presented here proposes an approach for compiling diagnostic problem solving structures from deep models corresponding to a knowledge of how devices function. We also outline how another problem solving structure for predicting consequences of proposed actions can also be compiled from the same representation, but the major emphasis is on diagnostic reasoning. In these cases, compilation is meant to

capture the idea that knowledge in a certain more general form is transformed into knowledge of a more particular form, suitable for particular classes of uses. (Whether a complete diagnostic structure is compiled initially or portions needed for particular diagnostic problems are compiled as these problems are encountered is not of concern in this research i.e, the word "compilation" is not used to contrast the process with "interpretation" in the computer science sense of the terms.) While the functional representation will be more economical in storage, the compiled structures will be more efficient for the particular problem solving tasks.

## 2. Components of a Functional Representation

We envisage that an agent represents the functioning of a device in many dimensions which include causal, temporal and interaction. In the causal dimension a "unit of functioning" (e.g: buzzing of a household buzzer) is represented as a causally related sequence - a genetic, not a temporal sequence - of device (or component) states. In the temporal dimension, these units obey time constraints. For example, two units of functioning should happen sequentially or overlap or their duration cannot exceed a certain amount of time, etc. In the dimension of interaction they interact through feed-back or by communicating information. For example, kidneys and lungs interact with the "acid-base" buffer system by communicating through changes in the concentration of bicarbonate and carbonic acid in blood. The functional representation of a device is an integrated whole of these various dimensions.

In this paper we briefly describe the salient features of the causal dimension of our functional representational scheme, and how it can be used for automatically compiling a diagnostic expert structure by using a device-independent diagnostic compiler. The compiled diagnostic structure has an architecture similar to the MDX system [3, 8] ie., it is a hierarchical collection of diagnostic (more specifically, classificatory) specialists.

Our representational scheme is rich in the number of primitives it employs to represent many aspects of functional knowledge. This richness is necessary to capture all the uses to which this representation can be put. While we will attempt to be complete in describing the causal dimension of the scheme in this paper, only a portion of the scheme will be utilized by the diagnostic compiler to be described.

## 3. A Representational Scheme for the Functioning of Devices

One of the significant tools available to humans as well as machines to combat complexity is abstraction. Accordingly, our scheme for functional representation allows one to represent functional knowledge at many levels of abstraction. Each level recursively describes the functioning of a device or component *in terms of* the abstractions of, and relations between, its components. At each level there are five significant aspects to an agent's functional knowledge:

- STRUCTURE: this specifies relationships between components, and abstractions of these components from lower levels.
- FUNCTION: this specifies WHAT is the response of a device or a component to an external or internal stimulus.
- BEHAVIOR: this specifies HOW, given a stimulus, the response is accomplished.
- GENERIC KNOWLEDGE: chunks of deeper causal knowledge that have been compiled from various domains to enable the specification of behavior; for example, a specialized version of Kirchoff's law from the domain of electrical circuits.
- ASSUMPTIONS: using which the agent chooses a behavioral alternative over other possible ones.

Next we describe how these five aspects together represent the functioning of devices at each level of abstraction. Following de Kleer and Brown [5, 6], we shall use the household buzzer (shown in Fig. 1) to illustrate our ideas.

FUNCTION

The functional specification of a device is illustrated below by describing one of the functions of the buzzer, namely "buzz".

```
FUNCTIONS:
  buzz: TOMAKE buzzing(buzzer)
        IF pressed(manual-switch)*
        PROVIDED  assumption1
        BY behavior1
  stop-buzz: ....
END FUNCTIONS
```

(Attached)

**Figure 1: A Schematic Diagram of a Household Buzzer**

In the above description "buzzing(buzzer)" is a state description.[1] "*" denotes repetition of a state. The buzzer is represented as having a number of functions viz., "buzz", "stop-buzz", etc. The initial state, viz., "t7" and "t8" (refer to figure 1) are electrically connected is specified by "assumption1" (more about assumptions later). The "BY" clause relates the function with its behavior i.e., the manner in which the function is accomplished (behavioral specification is described below). As we shall see in Sec. 4.2, this association between function and behavior is useful during compilation. Note that primitives such as TOMAKE (written out in capital letters in our description) trigger specific subprocesses during compilation and thus the compiler (described in Sec. 4.2) can be said to "understand" them (and their syntactic constraints on their arguments). On the other hand names

[1] More precisely, they are partial state descriptions of the device as a whole. We will use the term "state" for simplicity.

such as "behaviorl" are used for indexing; state descriptions are treated as strings and used to synthesize pieces of diagnostic knowledge.

STRUCTURE

The structure of a device (component) is represented using the abstractions of its components (subcomponents) and relations between them. As an illustration consider the structure of the buzzer given below. In this illustration "t1", "t2",etc., are terminals of components. Relations such as "serially-connected" are not understood by the compiler. "T1", "T2", etc., are local terminals. The function "magnetic" is defined at the next level as a function of the clapper in the same way as the functions of the buzzer are described. Its function is to disconnect T1 and T2 electrically if space is magnetized.

It is important to note firstly that the functional knowledge of a component is specified independent of that of a device which comprise the component. An abstraction of a component inside the specification of a device represents the role of the component in the functioning of the device. This not only concurs with the manner in which we store functional knowledge (e.g.,we know the function of a battery independent of its role in a car,camera,etc.) but also has an important practical significance, namely storage efficiency. Secondly, what is carried over from one level to another are not behavioral specifications but names of functions. This is important when an agent needs to replace a malfunctioning component by a functionally equivalent but a behaviorally different one.

```
STRUCTURE:
   COMPONENTS:
        manual-switch (t1,t2), battery (t3,t4),
        coil (t5,t6,space1), clapper (t7,t8,space2)
   RELATIONS:  serially-connected (manual-switch,
                          battery,coil,clapper)
               AND includes(space1,space2)

   ABSTRACTIONS-OF-COMPONENTS:
       COMPONENT  clapper (t1, t2, space)
           FUNCTIONS: magnetic,acoustic,mechanical
           STATES: elect-connected (t1, t2),
                   repeated-hit(clapper)
           ASSUMPTIONS: assumption2, assumption3
       END COMPONENT

       COMPONENT  coil (t1, t2, space)
           . . .
       END COMPONENT
   END ABSTRACTIONS-OF-COMPONENTS
END STRUCTURE
```

BEHAVIOR

The behavioral specification of a device describes the manner in which an agent has composed the functions of the components to obtain the functions of the device. This specification also has pointers to any generic domain knowledge and assumptions (relating to behavioral alternatives) used by the agent in the *process of composition*. Fig. 2 illustrates how the 'buzz' function discussed above is realized. We have made use of three conceptually important notations in behavioral specification. They are described below:

(Attached)

**Figure 2: An Illustration of Behavioral Specification**

```
1:        s1
          ||
          ||  BY <name-of-a-behavior>
          \/
          s2
```

For example, step 1 in figure 2. This is intended to represent that the state s1 (more specifically a state complex; see the states in step 1 of fig.2) causes the state s2 and the details are in another behavioral specification ( "behavior2"). This relation enables a behavioral specification of a device (or a component)to be made at many further levels of detail, but still within the overall level of the device (or component). We call this hierarchy the "Hierarchy of Details".

```
2:    s1
      ||
      ||  USING FUNCTION  <name-of-a-function>
      ||          OF <component>
      \/
      s2
```

For example,

```
 magnetized(space2)
      ||
      ||  USING FUNCTION magnetic
      ||     OF clapper(t7,t8,space2)
      ||     WITH assumption2
      \/
 ~elect-connected(t7,t8)
```

The specification in "assumption3" is to represent the idea that if space2 is magnetized, the resulting force will be greater than the spring force. The above notation means that state s2 is caused from s1 by making use of a function ("magnetic") of the component ( "clapper"). (Recollect the comments on the "magnetic" function in the specification of STRUCTURE of the buzzer above.) This makes it possible to glue the functions of the components together to obtain a behavior. In other words, it enables causal knowledge at the level of device (component) be represented in terms of causal knowledge at the level of its components (subcomponents). This hierarchy of causal

knowledge is called "Component Hierarchy".

```
3:        s1
          ||
          ||  AS-PER <name-of-a-knowledge-chunk>
          ||    IN-THE-CONTEXT-OF  < a state
          \/       description >
          s2
```

For example,

```
elect-connected(t7,t8)
          ||
          ||  AS-PER knowledge1 IN-THE-CONTEXT-OF
          ||       voltage-available( t3,t4)/\
          ||    serially-connected(battery,coil,
          ||              clapper,manual-switch)
          \/
voltage-applied(t5,t6)
```

This means that if the terminals t7 and t8 are electrically connected, then voltage will be applied between t5 and t6. This is true as per the knowledge chunk called 'knowledge1' when it is applied in the context of battery, coil, clapper and manual switch being serially connected, and voltage being available at the battery's terminals. ('knowledge1' is specified below.) This primitive enables causal knowledge (i.e., s1 causes s2) to be represented using more general causal knowledge (i.e., knowledge1) and as described below, the latter be represented using still more generic knowledge (ie., Kirchoff's law). We call this hierarchy the " Generalization Hierarchy".

GENERIC KNOWLEDGE

The generic knowledge specification of a device (component) describes all chunks of deeper knowledge used in its behavioral specifications. The following is a specification of 'knowledge1'. There are other types of generic knowledge requiring notations other than the ones used below.

```
GENERIC KNOWLEDGE:
    knowledge1:

     voltage-applied (T1,T2)
          ||
          || AS-PER kirchoff's-law
          ||  IN-THE-CONTEXT-OF
          ||  elect-connected(T1,T3)
          ||    ∧elect-connected(T2,T4)  /* T1...T4 are local symbols.
          \/
     voltage-applied ( T3,T4)
END GENERIC KNOWLEDGE
```

We would like to draw particular attention to the notion of GENERIC KNOWLEDGE in our representation. It enables us to capture the relation between functional representation and deeper causal knowledge such as Kirchoff's law. This link will be useful when the correctness of an application of a generic knowledge in a functional representation is checked. Moreover, an agent using the functional representation can justify a step in a behavioral specification by quoting the generic knowledge employed.

ASSUMPTIONS

All assumptions made use of in the behavioral specifications of a device (component) are described in ASSUMPTIONS as illustrated below with reference to the clapper.

```
ASSUMPTIONS:

  assumption2: IF magnetized(space)  THEN  magnetic-force > spring-force
  assumption3: IF ~magnetized(space) THEN magnetic-force < spring-force

END ASSUMPTIONS
```

de Kleer and Brown [6] state that a difference between a novice and an expert is that the latter has made explicit all the assumptions underlying behavior of devices. Our functional representation has constructs to represent

assumptions and their role in behavioral specification. It is perhaps worth restating here that though the compiler does not understand "magnetic-force", etc., it can use these strings to compose pieces of diagnostic knowledge.

## 4. Compilation of a Diagnostic Problem Solving System

Now we can proceed to a discussion of how a diagnostic problem solving system can be generated from the functional representation by using a device-independent compiler. As a prerequisite for this compilation, the compiler needs to check the correctness/consistency of those portions of the functional representation that it will use. An example of incorrect specification is: A behavioral specification may specify that s1 causes s2 as per some knowledge chunk in some context. But when the knowledge chunk is applied in the context, *s1 may not cause s2*. We have not yet investigated this form of reasoning. The compiler described here assumes that the representation is correct/consistent in those aspects that it uses.

### 4.1. The Structure of a Generated Diagnostic System

As shown in Fig. 3, the generated expert system is a hierarchy of specialists. The structure and problem solving of the diagnostic system are similar to those of a medical diagnostic system called MDX [3, 8]. Each specialist corresponds to a malfunction in the device at a certain level of abstraction, e.g., a bad clapper, bad serial connection, etc. Specialists corresponding to more general or abstract malfunctioning are higher in the hierarchy. For example, node 2 corresponds to the following malfunction: the buzzer does not buzz when the manual-switch is pressed. One of its sub-specialists (node 7) corresponds to acoustically bad clapper. Every specialist has knowledge to establish the associated malfunctioning. As shown in Fig. 3, the knowledge is in the form of two types of rules: confirmatory

and exclusionary rules. A malfunction is diagnosed top-down by establishing a specialist and refining the malfunction represented by it by calling its sub-specialists (see [8]).

(Attached)

**Figure 3: An Example of a Generated Diagnostic Expert**

We have identified three types of malfunctioning, namely a violated assumption, faulty function or faulty relation. The corresponding specialists can be viewed as "assumption checker", "function checker" and "relation checker."

### 4.2. *The Compilation Process*

The compiler first generates the root specialist which corresponds to a "malfunctioning buzzer." The root specialist contains no rules. Invocation of the diagnostic expert will automatically establish the root specialist. The compiler then generates a function checker for each function of the device (since the malfunctioning must be due to one or more of the faulty functions). For example, given the "buzz" function in Sec. 3, the compiler will generate a function checker with the following rule:

```
IF  pressed(manual-switch)* ∧ ~buzzing(buzzer)
    THEN confirm
    ELSE reject
```

(How the compiler operates on the PROVIDED clause is discussed later.) The function checkers generated as above will be attached to the root specialist. Afterwards the compiler, using the "BY" clause in the functional specification, obtains the behavior associated with each function and compiles it. For example, if the behavior is specified in the form (we do not discuss

here multiple causes and multiple effects):

s1 ----> s2 ----> ..... ----> sn

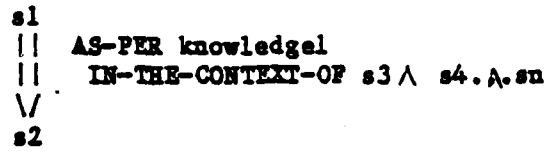
the compiler will generate a set of n-1 function checkers. (This is because if the function is faulty, then one of the steps in the corresponding behavior may be malfunctioning.) The rules for the ith specialist will be: "IF Si-1∧ ∼Si THEN confirm ELSE reject". For the "buzz" function example given above, "behavior1"(given in figure 2) will be used to generate nodes 5,6 and 7 (in figure 3). Another possible reason for a function not working is that the condition in the PROVIDED clause may not be holding. Therefore, the compiler will generate an assumption checker using the PROVIDED clause, if any. For example, node 4 corresponds to the PROVIDED clause in the "buzz" function. Also, note that the condition in the rule associated with node 5 does not include "pressed (manual-switch)*" since it is checked at the parent node.

Further processing of a behavioral specification depends on the kind of composition of behavior. The following cases arise:

1. Step 1 in figure 2 will also result in compiling 'behavior2' ('behavior2' is not specified in this paper) as described above, and attaching the generated specialists to node 5.

2. Step 2 in figure 2 will result in obtaining the behavior associated with the function "mechanical" from the functional representation of clapper and compiling it. If there is no behavioral specification for the function, the specialist will be a tip specialist (e.g. node 6). Also, If a function is used in a behavioral specification under an assumption (as illustrated in section 2), say "assumption1" and the specification of "assumption1" is of the form " IF s3 THEN s4" then the additional specialist with the rule "IF s3 ∧ ∼s4 THEN confirm ELSE reject " will be generated (since the function may be faulty due to a violated assumption).

3. The piece of causal knowledge




will result in n-3 sub-specialists with the rule for the ith (3 < i < n) specialist being "IF ~ Si THEN confirm ELSE reject". This is because the reason why the above step does not hold is that the context of the application of 'knowledge1' is different ( Note that the compiler assumes the representation to be correct/consistent). The ith specialist will turn out to be a relation checker if si corresponds to a relation.

We have implemented the compiler described above in ELISP on a DECsystem 20/60. The compiler is being tested extensively in the medical and electromechanical domains.

4.3. Meaning of Function in the Representation

In what sense can the representation proposed be said to correspond to "understanding" how the device works? We can point immediately to several aspects of understanding that is not meant to be incorporated in the representation: e.g., it does not understand "buzzing" or "electrically connected" or any of the terminals that are treated as strings of symbols, not to speak of any common sense substratum of knowledge such as objects and actions. What the functional representation really does in to organize the agent's understanding of how the device's functions result from behaviors made possible by the structure of the device, and contains explicit pointers to generic domain knowledge and assumptions about behavioral alternatives used by the agent in this process. Thus this representation is a piece in the total understanding structure, and is responsible for elucidating the role of the structure in the functioning of the device.

There is a need to distinguish between the "intrinsic" function of a device and the variety of functions it may be put to as part of a larger system. Consider the example[2] of a steam valve which opens and lets steam escape when the steam pressure goes over a certain limit. One designer may use this to make a high-pressure alarm by attaching it to a whistle, and another may use it as an "explosion-preventer" in a steam engine. But the typical functional representation of the steam-valve will not have either of these functions represented in it, since representing them will go beyond the intrinsic function of the device itself. What is an intrinsic function of a device is related to the "no-function-in-structure" (NFIS) principle that is suggested by [5] [10] as a way of ensuring that the agent's representation of a component is specified independently of the contexts in which the components may be used. Thus, this principle will forbid representing the battery's function in the buzzer as something that will help it to buzz, or that it is to be connected in series with a clapper in a buzzer, etc.

## 5. The Diagnostic Task and the Structure Produced by the Compiler

It is important to note that diagnostic reasoning in general needs strategies and knowledge that go beyond what is obtainable from considerations of functioning of the device alone. The final form of the diagnostic problem solver will reflect an integration of the diagnostic structure obtained from the functional representation and these additional strategies and knowledge. Some of this additional sources of complexity in diagnostic reasoning are given in the following.

1. Not all diagnostic knowledge about a system is derived from a

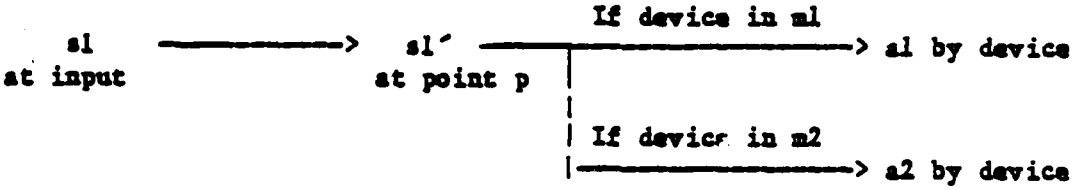
---

[2] Due to Ben Kuipers, personal communication. Note that the scheme permits representation of "negative" functions; the TOMAKE construct can be followed by the negation of a partial state, e.g. "TOMAKE ... enabled (engine) BY < > ."

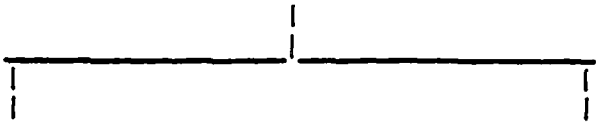
functional understanding of the system in question. This is especially true of complex systems such as the human body, where a sizable portion of the diagnostic knowledge of physicians comes from empirical and statistical means, rather than by a device-based understanding of the body. Our approach can account for those portions of the diagnostic knowledge base that are derivable from a deeper functional model.

2. The diagnostic system in Fig. 3 has diagnostic rules which use results of some tests as evidence for or against malfunction hypotheses. Often, however, these tests cannot be performed, or their results cannot be observed. E.g, one of the functions of liver is to secrete bile into the duodenum, and our functional compiler will produce, "Check if there bile is secreted in the duodenum" as a test to be performed for a liver malfunction. "Bile in the duodenum" is not directly observable, and thus additional information-gathering processes will need to be launched to infer this datum. There are several strategies typically available for this. One of them is to regard the presence or absence of bile as state changes in another functional system that characterizes the action of bile in the body, derive the consequences (e.g, bile not secreted into the duodenum ——> high bilirubin in blood), and use the latter data as evidence about the unobservable test values, and thus about the hypothesis. Sometimes, such an iterated reasoning process will only be able to give probabilistic evidence. E.g., the test "Check if <organ> is enlarged" may not be directly observable, but pain in that region may result from the enlarged organ. But other organs in that region, if enlarged, may also result in pain. Thus "pain in <region>" may give a probabilistic evidence for the hypothesis for which "Check if <organ> is enlarged" was to be a test.

3. Another strategy useful when the functional compiler produces nonobservable tests can be illustrated by the following example. Assume that a certain input, say s1, to a device will produce action a1 if the device is in mode m1, and action a2 if it is in mode m2. Consider the function corresponding to action a1, and let us assume that the behavioral sequence that results in a1 can be simply diagrammed as follows:




The fragment of the diagnostic hierarchy with the attached rules will be as follows:




Let us now assume that s1′ at p is not observable, but s1 is at the input and the device is not outputting a1 as action. Now a typical strategy in diagnostic reasoning is to ask what other functions s1′ at participates in, (in this case it will be a2 at output if device is in mode m2), and use it for ruling out. E.g., in this situation, the strategy will call for trying to get the device in mode m2, send s1 at input, and see if action a2 takes place.

The point about this strategy and the one in 2. above is that they are external to the functional representation and the compilation process. The diagnostic structure produces by the compiler has extracted the diagnostic knowledge directly derivable from functional knowledge, and other strategies need to be called upon to transform it further.

4. Another transformation of the diagnostic structure is quite common, but also beyond the responsibility of the functional representation and the diagnostic compiler. This transformation involves incorporating knowledge about costs of malfunctions and their relative probabilities into the diagnostic process so that certain malfunctions may be considered before others. E.g., in many electronic appliances, the first thing to check when it malfunctions is the battery, since it is the most common source of failure. Similarly some malfunctions may be more costly, or tests for them may be more economical, so they may be investigated before others. This form of knowledge, and its incorporation within the diagnostic structure, is not related to how the device's function arises from its structure, and thus requires processes outside of the compiler described earlier.

5. Because of the simplicity of the device, all the nodes in Fig. 3 could be established or rejected on the basis of one test. Typically, however, the knowledge needed for confirmation or rejection of a malfunction hypothesis in complex systems would consist of a number of pieces, each contributing some evidence for or against the hypothesis, and these pieces of evidence may need to be combined in a complex way. This complexity in the knowledge needed for confirmation or rejection of a malfunction hypothesis arises from a number of sources mentioned earlier: the fact that some of the knowledge is empirical and needs to be integrated; and the introduction of probabilistic aspects due to the need to convert the tests to observables.

6. Another example where additional knowledge and reasoning beyond the

functioning of the device is called for in reaching interesting diagnostic conclusions can be illustrated by the following example. Assume that the manual switch in the buzzer example has been so altered that the circuit is closed when the switch is open, and vice versa.[3] Now both the "buzz" and the "stop buzz" functions are not being fulfilled by the device. The diagnostic structure will be able to recognize this, and will make two sets of diagnoses: "buzz" function blocked because the Switch is not on when pressed, and the "stop buzz" malfunction happens because the switch does not go off when not pressed. To go from these correct conclusions to hypothesize that possibly the same structure change is causing both the malfunctions involves a form of reasoning which is distinct from the task of the diagnostic compiler.

7. In our approach, we define the diagnostic task as one of finding the structure change that is responsible for an observed failure of a function of a device. This is the diagnostic task that is handles by the compiled problem solver such as the one in Fig. 3. Note that this explicitly does not include having to account for how the observed (malfunction) behavior is generated from the changed structure. Human problem solvers sometimes do the latter as part of their diagnostic reasoning, sometimes they don't. The latter process may generally require a use of qualitative simulation, called envisioning [7], in order to reason from structure to behavior. In this paper we do not address the problem of envisioning, which is also related to how an agent may construct a functional representation from the structure and functions of components. Some of the references that propose some theories of how this may be done are proposed in [11, 7]. In Sec. 7.1, we discuss this and related issues further.

8. A malfunction may be caused by introduction of alternate causal pathways, without any particular component being faulted. In the buzzer example, imagine that some form of leakage or short circuit exists between terminals t5 and t6, thus depriving the coil of any current. There is nothing in the functional description that specifically states that such a short circuit should not be present for the device to work, nor would it be reasonable to expect it to say so, since such a statement will need to be made about every component. This is general electrical knowledge that is implicit in an agent's understanding of how a whole class of such devices work. During diagnosis, hypothesizing such short circuit for every component would be prohibitive. The diagnostic system in Fig. 3 would identify the problem as one of transiting from "elect-connected(t1,t2)" to "voltage-applied(t5,t6)" in behavior3 (see appendix). Under normal conditions this can be interpreted as failure of the coil (say an open-circuit in it) indicating that it should be replaced. However, when that fails to solve the problem, other (typically pre-compiled) causes of no voltage across the coil

---

[3] Example due to Wendy Lehnert, personal communication.

terminals can be tested, and one would assume that such a list will include short circuits around the component. Note that, while this portion of the diagnostic reasoning is itself not driven by any functional knowledge about the device, the problem solver in Fig. 3 derived from functional knowledge has nevertheless strongly focused the problem. Without this focusing, the number of alternate pathway hypotheses that will need to be considered will be prohibitively large.

6. "What Will Happen If" Problem Solving Using the Functional Representation

It is natural to expect an agent who understands how a device works to be able to say what consequences to functionality a change in the structure of the device will produce. Again, similar to our comment in item 7, Sec. 5, this task has two components: one is to know which of the intended functions will be affected by a proposed structure change, and the other is to deduce what new behavior will follow. E.g., given an understanding of how amplification is produced in an electronic amplifier, and given a change in the value of some resistors, an agent may be able to say, "the device will fail to amplify," while another agent may be able to augment this with, "instead, the device will oscillate." Our functional representation can support the former component (as we will show shortly). The latter, as in the case with the diagnostic task, will in general require construction of a new functional representation.[4]

We can view this WWHI problem solving as being performed by a problem solving structure that is compiled from the functional representation, similar

---

[4]However, when alternative functions (such as oscillation) are the consequence of certain assumptions in the functional representation being violated (such as the value of certain resistors being greater than a certain amount), it is possible to construct an augmented representation that essentially encodes the functions that would result from alternative assumptions. In such a case both parts of the WWHI task can be answered from the augmented functional representation. This augmented representation bears the same relation to our functional representation that the "intrinsic mechanism" of [6] does to their causal state sequences.

to the diagnostic problem solving described earlier. While we have not implemented this compiler, we can outline this process in a fairly straightforward manner. Let us motivate the discussion with the following concrete questions about the buzzer that we might wish such a structure to answer. (These questions are slightly modified versions of those given in [5].)

1. What happens if we reverse the leads of the battery?

2. What happens if we switch the leads of the coil?

3. What happens if we remove the battery from the circuit?

4. What happens if we make the clapper spring tension lighter?

The general procedure is that given a proposed change in the structure, the compiler looks to see in which behaviors the component affected by the structure change participates. Referring to the appendix, which gives complete description for the BUZZ function, we see that the battery participates in behavior2 and behavior3. For question 1 above, then, the relevant issue with respect to behavior2 and behavior3 is whether "FUNCTION voltage OF battery" is still delivered, and relation "serially-connected(battery, coil, clapper, manual-switch)" is still valid. The functional representation itself cannot answer these questions, but points to which questions need to be answered by further domain knowledge. We know from domain knowledge that for question 1 the answer is that neither the function nor the relation is affected, thus the two behaviors are not affected, and the "buzz" function is not affected. For question 3, a similar analysis will result in the following sequence of reasoning steps. In behavior3, the

transition from "elect-connected (t1, t2)" to "voltage-applied (t5,t6)" will fail, which will cause behavior3, viz., the transition from "pressed(manual-switch)" to "~elect-connected(t7,t8)" to fail. This will result in behavior2 failing, which will in turn result in behavior1 failing in the first transition, i.e, continuous pressing of manual switch will not result in the alternate connection and disconnection of t7 and t8. As a result, function "buzz" will fail, since behavior1 is used to realize this function.

Answering question 2 is quite similar. Question 4, however, is worth considering in some detail. The compiler will note that this part of the structure (i.e., spring tension) plays a role in assumption2 and assumption3 (see Assumptions subsection in Sec. 3. Again further domain knowledge is used to conclude that spring tension, if sufficiently weak, will violate assumption3. The clapper's functional representation describes its magnetic function as consisting of:

```
FUNCTION: magnetic: TOMAKE elect-connected(t7,t8)
                    IF magnetized(space2)
                    PROVIDED assumption2
                    AND
                    TOMAKE ~elect-conn(t7,t8)
                    IF ~magnetized(space2)
                    PROVIDED assumption3
```

Possible failure of assumption3 will imperil the second part of the above magnetic function, which will endanger the

~magnetized(space2) ————> elect-connected(t7,t8)

transition in behavior4 in "buzz" function (see appendix), thus putting the "buzz" function into question. Thus the final conclusion is that, for sufficiently weak tension (or sufficiently light arm) the "buzz" function will not be delivered by the device.

In the above description, we have used the word compilation, but in fact actually described the problem solving. More precisely, we should show that the compiler actually builds a problem-solving structure (like the diagnostic structure in Fig. 3) which then solves the WWHI problem. However, for purposes of this discussion, our aim was only to show how the functional representation can support this type of problem solving.

Note that the WWHI problem solving described above moves through behavioral levels of abstraction made possible by the hierarchical structure in the functional representation. This is consistent with the nature of information processing for this generic task as described in [2].

## 7. Relation to Other Work

Our research differs from those of de Kleer and Brown [5, 6], Patil [13], and Davis [4] in the following three significant respects:

1. Our representation identifies and relates the five aspects of functional knowledge, namely "function", "behavior", "structure", "assumptions" and "generic knowledge" in a specific scheme.

2. We have refined the notion of causation in terms of three hierarchies, namely "Hierarchy of Details", "Component Hierarchy" and "Generalization Hierarchy", and used it to represent the functional knowledge of devices.

3. We have been directly concerned with compilation of expert problem solving structures from the functional representation.

Kuipers [11] discusses a form of envisionment about changes to an equilibrium condition in a physiological system, and proposes this as a kind of understanding of the human body that may be incorporated in medical expert systems. The issues discussed by him are orthogonal to our concerns here. In [10] he critiques the representation proposals of de Kleer and Brown. Our discussion in Sec. 7.1 subsumes his critique.

7.1. The Work of de Kleer and Brown

We were originally motivated in our work by our concern with deep models for problem solvers, and thus we were searching for "mental models" of devices that would permit compilation of problem solving. Once we formulated our theory, we noted that the work of de Kleer and Brown, which proposed using envisioning to produce such a mental model, had some elements in common, but the representation was significantly different. The desire to understand the points of contact and departure between our points of view was in fact one of the reasons we chose the same device they had used, viz., the buzzer. In this section we compare the representation and the underlying points of view.

De Kleer and Brown's proposal for the "mental model" for the understanding of devices consists of representing qualitatively the causal sequence of behavioral states that the device needs to pass through during its functioning. In the buzzer example, a portion of this mental model can be stated as follows: "The clapper being open results in no current through it, which results in no current into the coil, which results in no magnetic field, which results in clapper being closed, which results in current flowing through its input and output, which results in current into the coil, which results in a magnetic field, which results in the clapper being open." The alternative representation proposed in [10], while different in a number of details, is also essentially a sequence of behavioral states. In our approach, however, the function is distinguished from the behavior of the device to accomplish it. Our functional representation is a hierarchical organization of behavioral segments of components into a representation which itself is not a causal sequence of partial states, but can be processed to obtain such a sequence. Further, unlike the causal sequence of [6], the

sequence that can be produced by our representation can be in varying levels of detail, because its hierarchical structure reflects the component hierarchy in the device. The hierarchical nature of the functional description is very important; otherwise describing the functioning of a complex system will involve an excruciatingly long sequence of causal states at a low level of abstraction. Simply out, we distinguish the description by an agent of the causal/behavioral sequence that a device undergoes, from the representation used by the agent to produce such a description, and identify the latter with the functional representation or the mental model of the agent.

Another advantage of distinguishing between function and behavior in the manner we do is that we are able to represent functions which prevent certain things from happening. (See footnote on "explosion-preventer" in Sec. 4.3.) This would be difficult to do if function is represented as the actual causal sequence followed by the device: prevented situations will hardly occur unless the device were malfunctioning!

Historically, de Kleer and Brown have been concerned with the issue of how an agent composes the behavior of the components of a device to obtain the behavior of the device itself. This process they have called envisioning. While this is an important part of the comprehension process, we feel that understanding the functioning of the device consists of a further activity of constructing a functional representation along the lines advanced in this paper. How an agent constructs such a functional representation by combining envisioning, functional representation of components of the device, some global properties of the device, and assumptions about the device itself is an open question.

We have said several times that a test of a functional representation is its ability to support different types of problem solving such as trouble-shooting. While, to our knowledge, de Kleer and Brown have not reported on any implementation of a diagnostic reasoning system based on their causal sequences as mental models, they have described how such a process may work [6], and it is instructive to compare that process with our approach.

They regard diagnostic reasoning as a task of accounting for how an observed behavior, which differs from an intended behavior, is actually produced by a malfunctioning device. They indicate that if the malfunctioning device essentially follows the same causal sequence (with some of the attributes of the states or assumptions possibly being different) as the one corresponding to the functioning device, then the structure causing the difference in behavior can be identified relatively easily. (For large systems, this might still require following an equally large state causation chain.) If the structure change is such that the device is not undergoing essentially the same causal chain, this task calls for a new envisioning for each structure change hypothesis. The causation chain itself is not useful for generating structure change hypotheses efficiently, or for the new envisioning that would be needed for each candidate structure change hypothesis. They correctly point out that the diagnostic process would now face a prohibitively combinatorial search.

The diagnostic task, as mentioned in item 7, Sec. 5, can be decomposed into two components: i. identifying the part of the structure that is responsible for the original function (for which the agent has a functional representation) not being delivered, and ii. explaining, if possible, how the changed structure produced the observed malfunctioning behavior. To use an

example well-known in electronics, amplifiers may sometimes malfunction, and behave as oscillators. The first part will correspond to saying, "Amplification is not taking place because resistor R1's value is less than required," and the second part will explain how the change in the value of the resistor resulted in the device acting as an oscillator. Note that if one can do the former without being able to do the latter, one would still have accomplished a significant diagnostic act. It is not clear that most trouble-shooting situations necessarily require the latter part in any case, since the goal of the trouble-shooting process if often identifying the components to replace that are causing the malfunction.

The first part is not subject to a combinatorial search problem in our theory: the functional representation and the compiler in principle can identify the component at fault[5] <---------- > because of the hierarchical structure of the resulting diagnostic structure as well as because the representation encodes the relation between structure and the intended function. The second part, viz., explaining how the new behavior is produced, may or may not be subject to combinatorial search, depending upon the degree of change to the structure and the resulting need for new a functional representation.

---

[5]See item 8, Sec. 5 for some qualifications to this statement, but the argument here is not affected.

## 8. Concluding Remarks

As we have indicated the task of accounting for how complex systems and devices are understood has many components. We have developed a framework in which this comprehension process and its relation to problem solving can be investigated. In particular, we have proposed a language for the representation of what we call the causal component of this comprehension that captures some aspects of how a function of a device arises from its structure, and shown that this structure can support problem solving for diagnosis and some kinds of prediction.

Directions for future research include the following: First, we need to develop methods to check the correctness/consistency of a given functional representation. This requires domain knowledge for interpreting state descriptions as well as the relations such as "serially-connected." Second, we need to investigate the design of the other two dimensions of our representational scheme, namely temporal and interaction. Also, the causal dimension has to be integrated with the other two in a disciplined, practically useful and cognitively meaningful framework. Third, we need to identify the compilation processes that come into play to generate other types of expert problem solving structures, such as predicting functional and behavioral consequences of changes in structure or assumptions. Finally, it is a matter of significant theoretical and practical interest to ask how an agent can incrementally acquire a functional representation of a device from its structure, generic knowledge, inadequacy of the current representation for supporting the compilation of adequate problem solving systems, etc.

In broader terms, this research is part of our on-going effort to uncover the multiplicity of generic structures and processes involved in knowledge-

based problem solving. Whether or not one accepts the hypothesis of homogeneous and unitary architectures such as production systems as adequate at the level of symbol-processing in the mind, we feel we need a richer collection of generic knowledge structures and a correspondingly rich collection of knowledge-processing mechanisms that operate on them, in order to account for knowledge-based problem-solving activity at the information-processing level. In expert systems work especially we feel there is a need to explore richer architectures that capture the information-processing activity. From this perspective, the functional representation is one of the information-processing level theories that are needed for understanding knowledge-based reasoning.

## REFERENCES


[1] Chandrasekaran, B. / Mittal, S.
On Deep Versus Compiled Approaches to Diagnostic Problem Solving.
International Journal of Man Machine Studies 19:425-436, 1983.

[2] Chandrasekaran, B.
Towards a Taxonomy of Problem-Solving Types.
AI Magazine 4(1):9-17, Winter/Spring, 1983.

[3] Chandrasekaran, B. / Mittal, S.
Conceptual Representation of Medical Knowledge for Diagnosis by Computer: MDX and Related Systems.
In M. Yovits (editor), Advances in Computers, , pages 217-293. Academic Press, 1983.

[4] Davis, R. / Shrobe, H. / Hamsher, W. / Wiekert / Shirley, K. / Polit, S.
Diagnosis Based on Description of Structure and Function.
Proc. National Conf. on AI : , 1982.
Pittsburgh, PA.

[5] de Kleer, J. / Brown, J.S.
Mental Models of Physical Mechanisms and Their Acquisition.
Erlbaum, 1981, .

[6] de Kleer, J., Brown, J.S.
Assumptions and Ambiguities in Mechanistic Mental Models.
CIS 9, Xerox Palo Alto Research Center, 1982.

[7] de Kleer, J. / Brown, J.S.
Foundations of Envisioning.
Proc. of AAAI :434-437, 1982.

[8] Gomez, F. / Chandrasekaran, B.
Knowledge Organization and Distribution for Medical Diagnosis.
IEEE Transactions on Systems, Man, and Cybernetics SMC-11(1):34-42, January, 1981.

[9] Hart, P.E.
Direction for AI in the Eighties.
SIGART Newsletter 79:11-16, 1982.

[10] Kuipers, B. / de Kleer, J. / Brown, J.S.
"Mental Models", A Critique.
TUWPICS No. 17, Working Papers in Cognitive Science, Tufts Univ.
1981

[11] Kuipers, B.
Commonsense Reasoning about Causality: Deriving Behavior from Structure.
1982
Tufts University, Working Papers in Cognitive Science.

[12] Michie, D.
High-road and Low-road Programs.
AI Magazine 3:21-22, 1982.

[13] Patil, R.S.
Causal Representation of Patient Illness for Electrolyte and Acid-Base Diagnosis.
1981
Doctoral Dissertation, TR-267, MIT Lab for Computer Science, Cambridge, MA.

[14] Weiss, S.M. / Kulikowski, C.A. / Amarel, S. / Safir, A.
A Model-based Method for Computer-aided Medical Decision-Making.
Artificial Intelligence (11):145-172, 1978.

# APPENDIX

To the paper

"A REPRESENTATION FOR THE FUNCTIONING OF DEVICES
THAT SUPPORTS COMPILATION OF EXPERT PROBLEM SOLVING STRUCTURES"

by V.S. Moorthy and B. Chandrasekaran

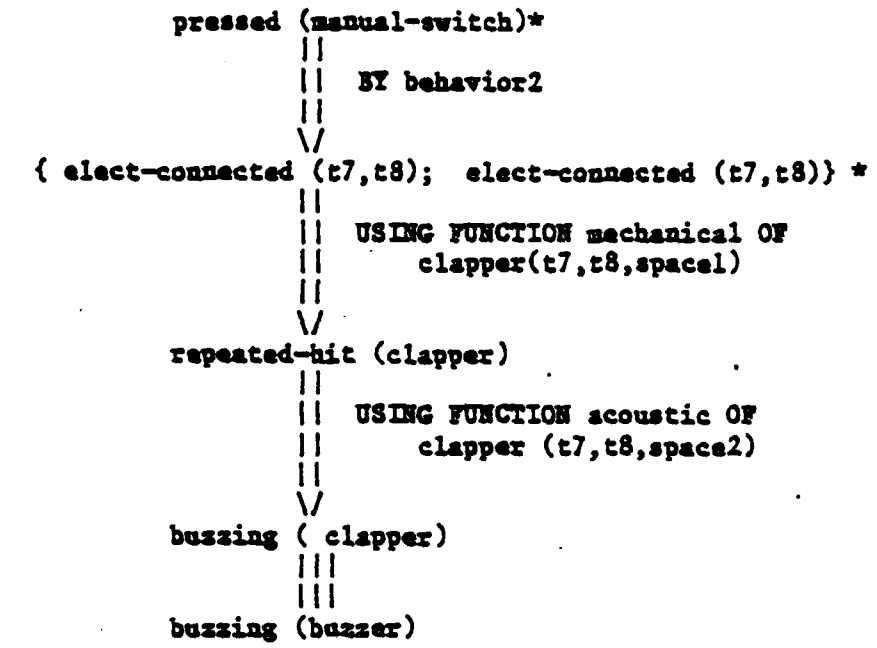
Details of the functional representation of FUNCTION: buzz of the buzzer

NOTE: We have represented below only the buzzer;
Battery,coil,clapper and manual switch have NOT been represented.

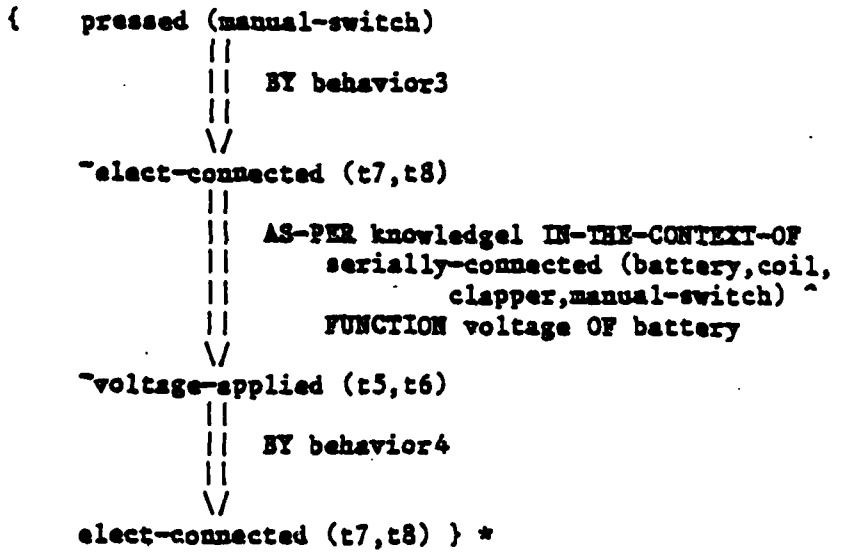
```
DEVICE   buzzer
   FUNCTION:
        buzz: TOMAKE buzzing (buzzer)
              IF pressed (manual-switch)*
              PROVIDED  INITIAL elect-connected (t7,t8)
              BY behavior1

        stop-buzz:TOMAKE  ~buzzing (buzzer)
                  IF  ~pressed (manual-switch)
                  PROVIDED INITIAL buzzing (buzzer)
                  BY behavior5

   STRUCTURE:
        COMPONENTS:
            manual-switch (t1,t2), battery (t3,t4),
            coil (t5,t6,space1), clapper (t7,t8,space2)

        RELATIONS:
            serially-connected (manual-switch,battery,coil,clapper),
            includes (space1,space2)

        ABSTRACTIONS-OF-COMPONENTS:
            COMPONENT clapper (T1,T2,SPACE)
                FUNCTIONS: mechanical,acoustic,magnetic
                STATES: elect-connected (T1,T2),
                        repeated-hit (clapper)
            COMPONENT coil (T1,T2,SPACE)
                FUNCTIONS: magnetic
                STATES: magnetized (SPACE), voltage-applied(T1,T2)

            COMPONENT  manual-switch(T1,T2)
                FUNCTIONS: connect
                STATES: elect-connected (T1,T2),
                        pressed (manual-switch)

            COMPONENT  battery (T1,T2)
                FUNCTIONS: voltage
```

BEHAVIOR:

behavior1:




behavior2:



behavior3:

pressed (manual-switch)
||
|| USING FUNCTION connect OF
||  manual-switch (t1,t2)
||
\/
elect-connected (t1,t2)
||
|| AS-PER knowledge1 IN-THE-CONTEXT-OF
||  FUNCTION voltage OF battery ,
||  serially-connected (battery,coil,
||  clapper,manual-switch)
||
\/
voltage-applied (t5,t6)
||
|| BY behavior4
||
\/
~elect-connected (t7,t8)

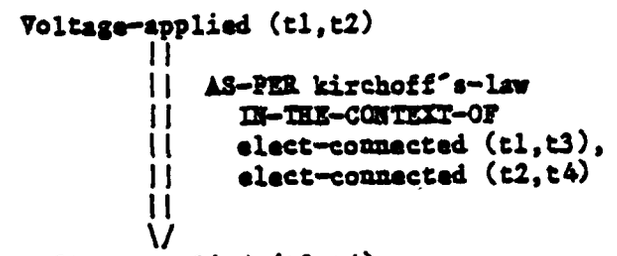
behavior4: IFF

voltage-applied (t5,t6)
||
|| USING FUNCTION magnetic OF
||  coil (t5,t6,space1)
||
\/
magnetized (space1)
||
|| AS-PER knowledge2 IN-THE-CONTEXT-OF
||  includes (space1,space2)
||
\/
magnetized (space2)
||
|| USING FUNCTION magnetic OF
||  clapper (t7,t8,space2)
||
\/
~elect-connected (t7,t8)

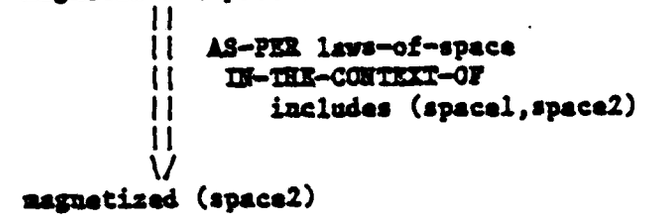
Note: IFF (S1 ==> S2 ==> S3) is the same as
S1 ==> S2 ==> S3 and ~S1 ==> ~S2 ==> ~S3

GENERIC KNOWLEDGE:

knowledge1:




knowledge2:




END-DEVICE buzzer

Fig. 1

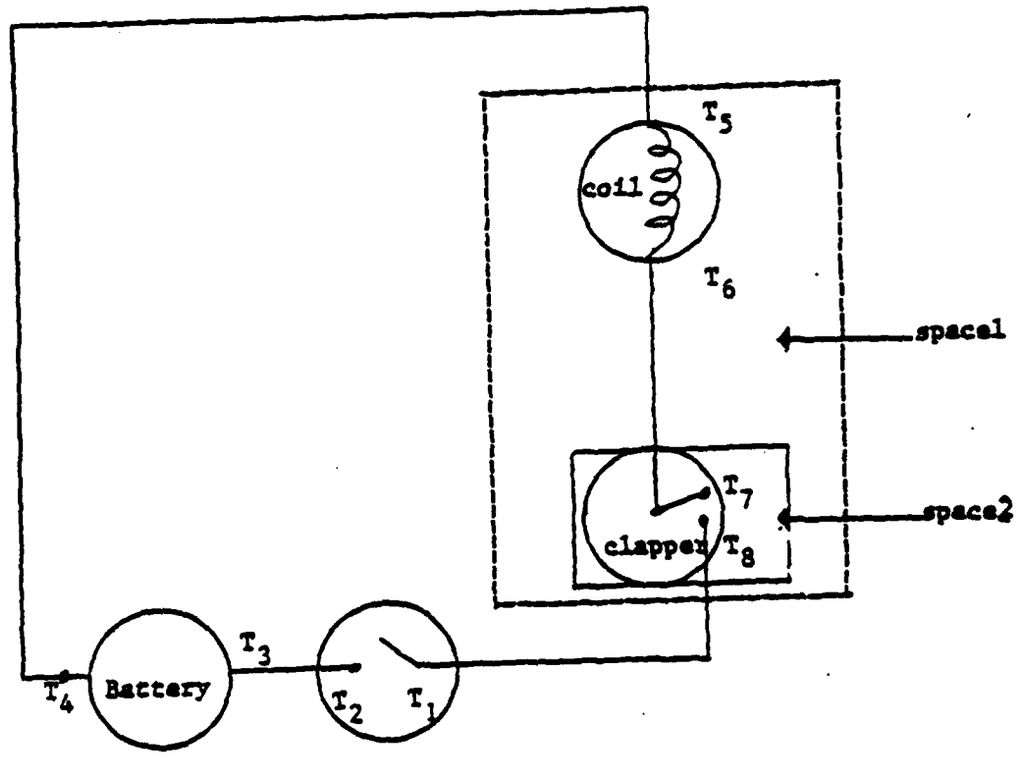
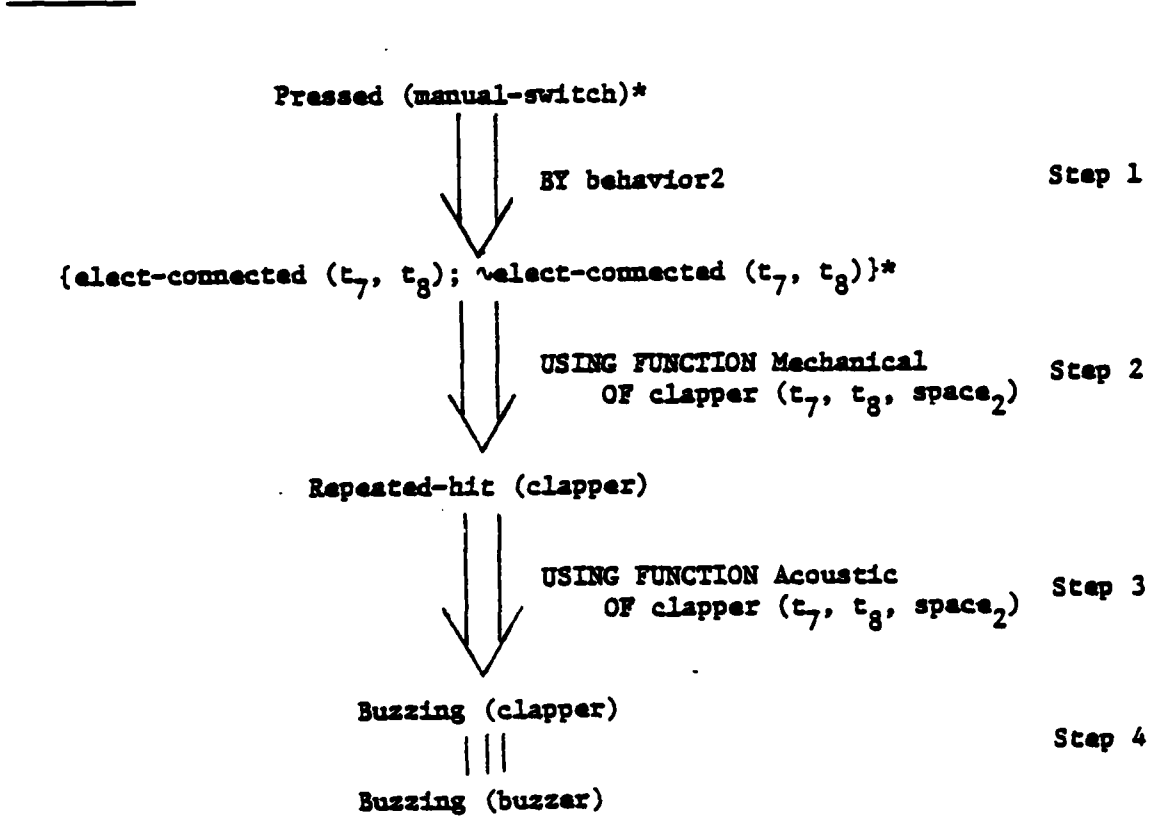
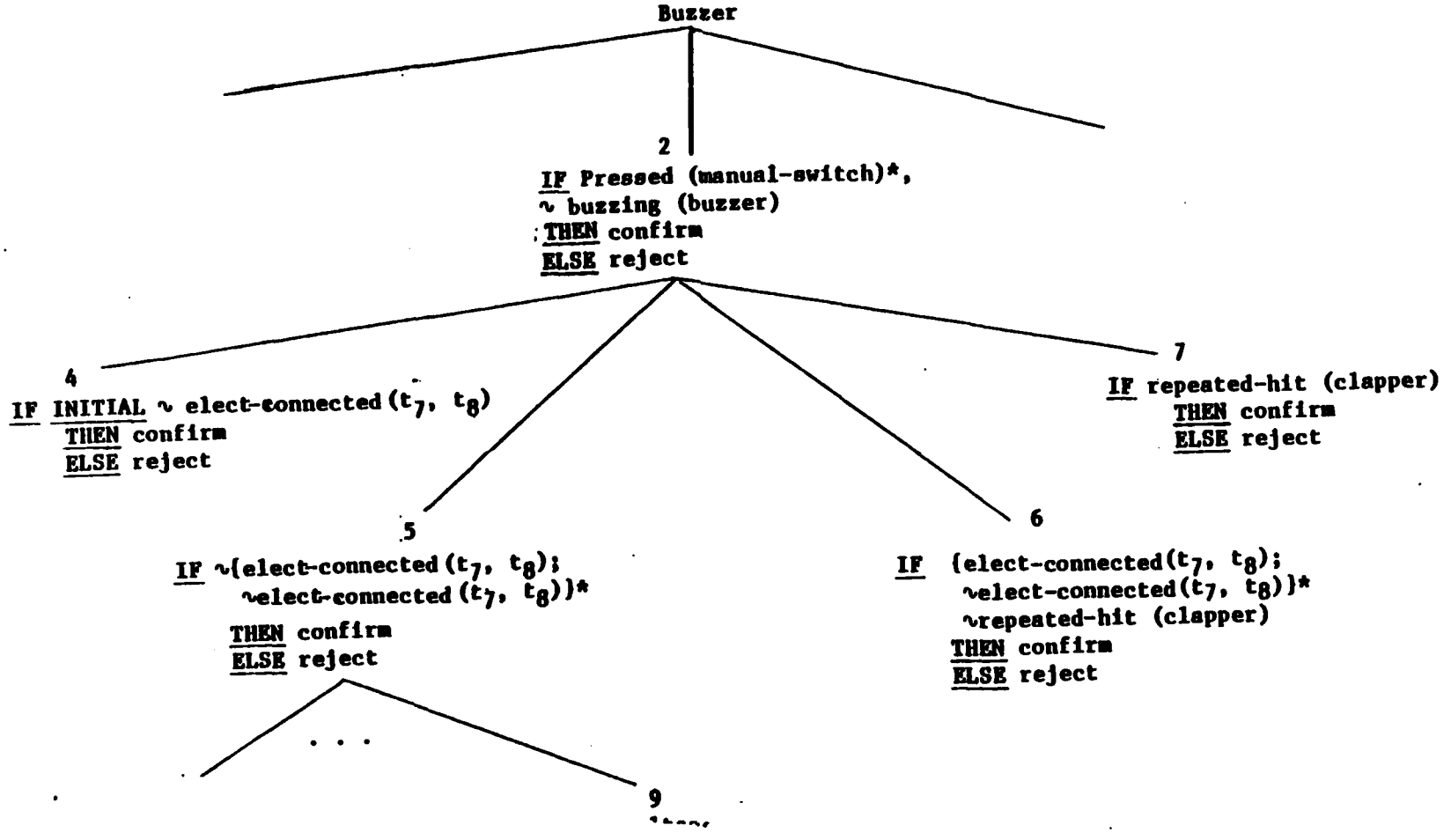

$T_5$
coil
$T_6$
space1
$T_7$
space2
clapper
$T_8$
$T_3$
$T_4$
Battery
$T_2$
$T_1$
manual switch

Fig. 2




Notes:
1. "≡" equivalences two states
2. $s_1;s_2$ means $s_2$ follows $s_1$

FIG. 2


Buzzer
2
IF Pressed (manual-switch)*,
∿ buzzing (buzzer)
THEN confirm
ELSE reject
4
IF INITIAL ∿ elect-connected ($t_7$, $t_8$)
THEN confirm
ELSE reject
5
IF ∿{elect-connected ($t_7$, $t_8$);
∿elect-connected ($t_7$, $t_8$)}*
THEN confirm
ELSE reject
6
IF {elect-connected ($t_7$, $t_8$);
∿elect-connected ($t_7$, $t_8$)}*
∿repeated-hit (clapper)
THEN confirm
ELSE reject
7
IF repeated-hit (clapper)
THEN confirm
ELSE reject
. . .
9

# APPENDIX B

# EXPERT SYSTEMS FOR A CLASS OF MECHANICAL DESIGN ACTIVITY

To appear in Proc. of IFIP WG5.2 Working Conference on Knowledge Engineering in Computer Aided Design, Budapest, Hungary, Sept. 84.

# EXPERT SYSTEMS FOR A CLASS OF MECHANICAL DESIGN ACTIVITY

D. C. Brown, B. Chandrasekaran

Artificial Intelligence Group
Department of Computer & Information Science
The Ohio State University
Columbus, Ohio 43210, U.S.A.

We are investigating the structure and operation of expert systems for the design of mechanical components. Our approach, referred to as Design Refinement, applies to a particular class of design activity. A hierarchical structure of conceptual specialists solves the design problem in a distributed manner, top-down, choosing from sets of plans and refining the design at each level of the hierarchy.

## 1. Introduction

### 1.1. Our Research

The research reported here is concerned with the design of mechanical components by computer. We are investigating the structure and operation of expert systems for this task. As design in general is a very poorly understood activity we have chosen to limit ourselves to a particular class of design activity, understand it thoroughly, and build expert systems that will simulate that activity in a realistic way. Our initial discussion of this problem and presentation of a simple prototype system has been presented elsewhere.[1]

### 1.2. Other Work in Design

While there has been some AI-related work in constructing aids to design[2], in particular in the electronics domain[3, 4, 5], the key issues of the structure of knowledge and control in design problem-solving have been given less attention [6, 7, 8, 9, 10, 11, 12, 13]. One promising short term approach is that of extending the existing CAD concentration on design support systems to include AI. Some research has been done by our group and others on intelligent graphical aids, and in knowledgeable data-bases[14, 15].

### 1.3. Problem-solving Types

Most first-generation expert systems have been rule-based with a separate inference engine. The rule-based approach has proven to be both practical and profitable, and resulted in a number of expert systems. However, for handling

more complex forms of expert problem solving, there is a need for knowledge representation approaches with a richer set of constructs. These constructs should be helpful in capturing other more structured forms of knowledge and should be such that they help organize both knowledge and problem solving behavior for more focussed problem-solving.

We have been developing an approach to problem-solving that views a complex body of knowledge as being decomposed into a structure of Specialists engaged in collective, cooperative action. Each specialist does the same kind of problem-solving but contains different domain expertise. The organization of the specialists depends on the problem-solving type and will reflect the conceptual organization imposed on the domain by a human problem-solver. A Problem-solver therefore consists of a well organized collection of specialists each doing the same type of activity, while an Expert system consists of a well organized collection of one or more interacting problem-solvers.

We have identified several distinct types of problem-solving[16] — such as diagnosis, which reasons about how to classify a complex description of reality as a node in a diagnostic hierarchy[17], consequence finding which reasons about the consequences of contemplated actions on complex systems, and design which reasons about providing values for the attributes of some entity which has constraints placed on it. Clearly these are not the only types. Once these types are well understood, we will be in the situation to be able to categorize which kinds of expert problem-solving we know how to mechanize, something with which current approaches offer little help.

## 2. Three Classes of Design

### 2.1. Design in General

In general, design is a highly creative activity involving diverse problem-solving techniques and many kinds of knowledge. We know very little about what creativity is, although there has been some work on <u>discovery</u> processes and heuristics[18, 19]. Often the goals for a design are poorly specified, and these goals may be altered during the design by feedback from successes or failures. Clearly, as we don't know many of the components of design in general, and as we poorly understand those components we do know about (for example, planning[20]), a general approach to design is currently out of reach.

However, opinions in the literature agree about many components of design activity. There is an element of refinement. That is, descriptions get refined into less abstract forms. Plans are used in recognizable situations where

experience has produced a sequence of design decisions that will usually work. Such plans are the result of past planning and validation by repeated use. Design activity often has a rough design phase followed by design proper. Design activity is organized in ways that reflect the structure or functionality of the entity being designed. Similarly the representation of the design is also structured. For example, blueprints will have detailed drawings and general drawings. A single blueprint will have areas reserved for different subcomponents or functionally related entities. During the design various restrictions on what is allowable for this kind of entity will be checked at appropriate points, and the initial conditions (ie. requirements) form a starting set of restrictions imposed on the design from outside.

After discussions with practitioners, keeping the above opinions in mind, we have roughly classified design activity into three classes -- although it is clear that there are subclassifications that make the classes overlap in some ways. They vary from completely open-ended activity (design in general) to the most mundane.

## 2.2. Class 1 Design

The average designer in industry will rarely if ever do class 1 design, as we consider this to lead to a major inventions or completely new products. It will often lead to the formation of a new company, division, or major marketing effort. This then is extremely innovative behavior, and we suspect that very little design activity is in this class. For this class neither the knowledge sources nor the problem-solving strategies are known in advance.

## 2.3. Class 2 Design

This is closer to routine, but will involve substantial innovation. This will require different types of problem-solvers in cooperation and will certainly include some planning. Class 2 design may arise during routine design when a new requirement is introduced that takes the design away from routine, requiring the use of new components and techniques. What makes this class 2 and not class 1 is that the knowledge sources can be identified in advance, but the problem-solving strategies, however, cannot.

## 2.4. Class 3 Design

Here a design proceeds by selecting among previously known sets of well-understood design alternatives. At each point in the design the choices may be simple, but overall the task is still too complex for it to be done merely by

looking it up in a data-base of designs, as there are just too many possible combinations of initial requirements. The choices at each point may be simple, but that does not imply that the design process itself is simple, or that the components so designed must be simple. We feel that a significant portion of design activity falls into this class.

### 2.4.1. A Class 3 Product

In a large number of industries products are tailored to the installation site while retaining the same structure and general properties. For example, an Air-cylinder intended for accurate and reliable backward and forward movement of some component will need to be redesigned for every new customer in order to take into account the particular space into which it must fit or the intended operating temperatures and pressures. This is a design task, but a relatively unrewarding one, as the designer already knows at each stage of the design what the options are and in which order to select them. Thus there is strong economic justification in attacking this problem.

### 2.4.2. Class 3 Complexity

The complexity of the class 3 design task is due not only to the variety of combinations of requirements, but also to the numerous components and sub-components, each of which must be specified to satisfy the initial requirements, their immediate consequences, the consequences of other design decisions, and the constraints of various kinds that a component of this kind will have.

While class 3 design can be complex overall, at each stage the design alternatives are not as open-ended as they might be for class 2 or 1, thus requiring no planning during the design. In addition, all of the design goals and requirements are fully specified, subcomponents and functions already known, and knowledge sources already identified. For other classes of design this need not be the case. Consequently, class 3 design is an excellent place to start in an attempt to fully understand the complete spectrum of design activity.

### 2.4.3. Classification as Class 3

If, during an attempt at class 3 design, all of the design alternatives fail, then it is possible that the designer will switch to class 2 activity. This is most likely to happen if the problem is on the border between classes or if the designer has little experience with this type of component and has not yet fully formed a completely satisfactory class 3 approach. We have no way as yet of knowing whether such a distinct inter-class border exists. It appears that experienced designers are able to judge whether a project is class 3 or not, the

main clue being, of course, whether they have designed that component often before with initial requirements that are judged to be similar.

#### 2.4.4. General Description

It should be clear by now that we consider class 1 and class 2 design to be outside the reach of effective contributions from AI technology at present. Class 3 design however can benefit from other work in knowledge-based systems.

It is our working hypothesis that there is a very specific type of problem solving behavior associated with design activity of the class 3 type. Specifically that a hierarchy of conceptual specialists solve the problem in a distributed manner, top-down, by choosing at each stage of the design from a set of plans, thus refining the design. Specialists can use the expertise of other specialists below them in the hierarchy in ways specified by the plans.

## 3. An Approach to Class 3 Design

### 3.1. Introduction

A design problem-solver will consist therefore of a hierarchical collection of design specialists, where the upper levels of the hierarchy are specialists in the more general aspects of the component, while the lower levels deal with more specific subsystems or components. They all access a design data-base possibly mediated by an intelligent data-base assistant[15, 21]. We will first describe the design agents[1], and then the phases of their interaction.

### 3.2. Design Agents

#### 3.2.1. Specialists

A Specialist is a design agent that will attempt to design a section of the component. The specialists chosen, their responsibilities, and their hierarchical organization will reflect the mechanical designer's underlying conceptual structure of the problem domain. Exactly what each specialist's responsibilities are depends on where in the hierarchy it is placed. Higher specialists have more general responsibilities. The top-most specialist is responsible for the whole design. A specialist lower down in the hierarchy will

---

[1]By the term "Agent" we mean a Specialist, Task, or Step -- ie. any active module of the problem-solver

be making detailed decisions. Each specialist has the ability to make design decisions about the part, parts or function in which it specializes. Those decisions are made in the context of previous design decisions made by other specialists. A specialist can do its piece of design by itself, or can utilize the services of other specialists below it in the hierarchy. We refer to this cooperative design activity of the specialists as Design Refinement.

Every specialist also has some local design knowledge expressed in the form of constraints. These will be used to decide on the suitability of incoming requirements and data, and on the ultimate success of the specialist itself (ie. the constraints capture those major things that must be true of the specialist's design before it can be considered to be successfully completed). Other constraints, embedded in the specialist's plans, are used to check the correctness of intermediate design decisions. Still more constraints are present in the design data-base as general consistency checks.

### 3.2.2. Plans

Each specialist has a collection of plans that may be selected depending on the situation, and it will follow the plan in order to achieve that part of the design for which it is responsible. A Plan consists of sequence of calls to Specialists or Tasks (see below), possibly with interspersed constraints. It represents one method for designing the section of the component represented by the specialist. The specialists below will refine the design independently, tasks produce further values themselves, constraints will check on the integrity of the decisions made, while the whole plan gives the specific sequence in which the agents may be invoked. Typically as one goes down in the hierarchy, the plans tend to become fewer in number and more straightforward.

As each plan is considered to be the product of past planning, refined by experience, one should not expect many failures to occur. However, as not all combinations of values have been handled before or anticipated it is possible for plan failures to occur due to intra-plan and extra-plan constraint violations.

### 3.2.3. Steps, Tasks, and Constraints

We consider a Step to be a design agent that can make one design decision given the current state of the design and taking into account any constraints. For example, one step would decide on the material for some subcomponent, while another would decide on its thickness. A Task is a design agent which is expressed as a sequence of steps, possibly with interspersed constraints. It is responsible for handling the design of one logically, structurally, or functionally coherent section of the component; for example a seat for a seal, or

a hole for a bolt.

A Constraint is an agent that will test for a particular relationship between two or more attributes at some particular stage of the design. Constraints can occur at almost any place in the hierarchy. For example, a constraint might check that a hole for a bolt is not too small to be machinable given the material being used. Constraints will be discussed further when we address failure handling.

## 3.3. The Four Phases

### 3.3.1. Requirements

The design activity can be considered to fall into four phases. Initially, the requirements are collected from the user and are verified both individually and collectively. For example, it may be reasonable to ask for a component to be made of lead, and for it to weigh less than 5 ounces, but the combination will often be unreasonable. Once it has been established that the system is capable of working with those requirements, a rough-design is attempted.

### 3.3.2. Rough-design

Rough-design is poorly understood at present, but it serves at least two purposes. First, those values on which much of the rest of the design depends will be decided and checked. If they can't be achieved then there is little point going on with the rest of the design. This also has the effect of pruning the design search space, as once the overall characteristics of the design are established it reduces the number of choices of how to proceed with the rest of the design. Second, as any mutually dependent attributes can prevent a design from progressing (ie. A depends on B, and B depends on A), rough-design can, as human designers do, pick a value for one of the attributes and use that as if the dependencies didn't exist.

It appears at present that rough-design and design share the same conceptual hierarchical structure. However, that remains to be confirmed. The rough-design hierarchy is in general much shallower than the design hierarchy as more general decisions are being made. We are proposing that specialists have both design and rough-design plans to select from depending on the current phase. Not all specialists will need both. It is entirely feasible that phases could be intermixed during problem-solving, but we have chosen to restrict the rough phase to be first, followed by the design phase.

### 3.3.3. Design

Once rough-design has completed satisfactorily, the design phase can proceed. Design starts with the topmost specialist and works down to the lowest levels of the hierarchy. A specialist S begins by receiving a design request from its parent specialist, which might include some design requirements (constraints). It refers to the specification data-base and obtains a list of specification data relevant to its further work. A plan is selected using these data and the current state of the design. For example, if one of the requirements is low cost, a plan with that quality can be selected. The exact nature of the plan selection process is a matter for further research, and, with a language for plans, will be a major part of the theory of design.

Thus, S fills in some of the design, then calls its successors in a given order with requests for refinement of the design of a substructure. If some of the substructures are independent of each other, then they may be invoked in parallel. The knowledge in the specialist prioritizes the plans, and invokes alternative plans in case of failure by one of the successors. Parts of a plan may indicate immediately that constraints cannot be satisfied. This is considered as failure. When all of a specialist's plans fail, or when failure can be deduced immediately, the specialist communicates that to its parent.

### 3.3.4. Redesign

If any failures occur during the design process then a redesign phase is entered. If the phase succeeds then a return can be made to the design phase. At the lowest level, failures occur when a constraint fails in some step. The system attempts to handle all failure at the point-of-failure before admitting defeat and passing failure information up to its parent. A step, for example, may be able to examine the failure and then produce another value, in order to satisfy the failing constraint, while still retaining local integrity. This failure handling activity and the associated redesign phase will be discussed later.

Other work on redesign in the literature has concentrated on "functional redesign", that is, "the task of altering the design of an existing, well understood circuit, in order to meet a desired change to its functional specifications"[4]. Here we use redesign to mean an attempt by a design agent to change a value to both satisfy a constraint while keep as much as possible intact of the previous design.

### 3.4. Communication

The main means of communication in the system is by passing information and control between specialists across the connections forming the hierarchy. In this way the flow of control is restrained and the system exhibits clear, well-focused problem-solving activity. It remains to be shown whether this form of control is sufficient, but it is based on a belief that Class 3 design systems are "nearly decomposable"[22] and that "the interactions between subsystems are weak but not negligible". We believe that for Class 3 design the structure is dominantly hierarchical and that interactions are handled by specific strategies.

Information is passed in the form of messages that can, for example, request action, report failure, ask for assistance, and make suggestions. This rich variety of messages is the key to handling subsystem interactions. In addition, one part of the emerging theory of design problem-solving will be the form and content of these design oriented messages.

### 3.5. Other Agents

In general, a collection of design specialists will not be sufficient for the design task, and will, at least, need an intelligent data-base to keep track of the ongoing state of the design. Other specialists outside the design specialist hierarchy could provide calculations, such as stress analysis, and other data-base functions such as catalogue lookup. In a more general design system, requests could be made to other types of problem-solvers[16]. It is perfectly acceptable to consider the human user as one of the problem-solvers, as requests for assistance will occur at well defined points in the design with precise pieces of design to do. The expert system can subsequently check the acceptability of the results provided by using constraints. Here the usual image of the designer controlling the invocation of analysis packages and problem solvers is reversed.

## 4. An Instance of Class 3 Design

### 4.1. The Air-cylinder

Let us consider a real, but not overly complex example for illustrative purposes. In our collaboration with AccuRay Corporation, we have selected an Air-cylinder (AC) as a suitable object for our continuing studies of design problem-solving. Our preliminary work on design problem-solving was reported in Brown (1983)[1]. We are now working on extending the theory and examining the issues and problems using the AC as our test case. The AIR-CYL design problem-solving system is

currently under development using Rutgers ELISP and FRL[23] on the OSU CIS department's DEC system-20.

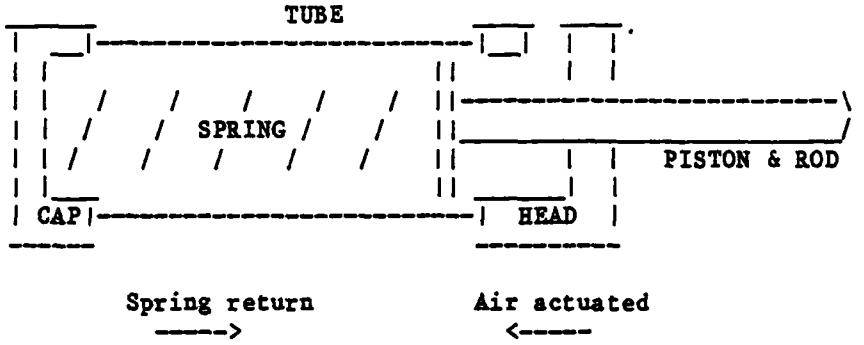



Figure 1: Air-cylinder

The AC has about 15 parts, almost all of which are manufactured by the company according to their own designs, as their requirements are such that the components cannot be purchased. The AC is redesigned and changed slightly for applications with markedly different requirements, and, consequently, the domain is Class 3 in type. In operation, compressed air forces a piston back into a tube against a spring. Movement is limited by a bumper. The spring returns the piston, and the attached "load", to its original position when the air pressure drops.

## 4.2. The Conceptual Structure

An Air Cylinder Designer was interviewed over a period of time, the protocols were analyzed and the "trace" of the design process was obtained. Figure 2 shows the progress of the design over time (from left to right) and the groupings of the decisions (from top to bottom).

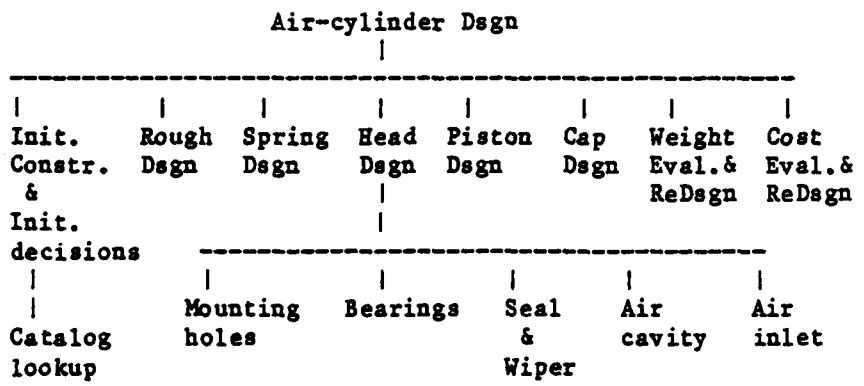



Figure 2: Design Trace

The trace was subsequently analyzed to establish the underlying conceptual

structure. For example, the Head was clearly treated as a separate conceptual entity, as it occupied a substantial portion of the designers time and effort. The Spring was actually designed by a different person as an essentially parallel activity, while the rest of the design was "lumped together" by the designer as the third major activity. The fact that the specialists can be fairly easily identified, and that the plans for each specialist are also identifiable and small in number strongly confirms that this is a class 3 activity. On examination we could see that this organization tends to localize dependencies, and allows for parallel design activity -- something of which most designers are not able to take advantage!

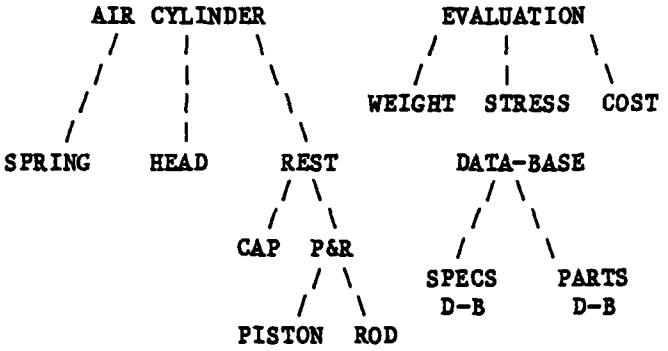



Figure 3: Partial AIR-CYL Structure

### 4.3. Design Agents

Currently we are considering two-level plans, where a plan consists of a set of tasks, possibly with some in parallel; a task being of a set of steps, with each step corresponding to a single design decision. We are developing a language for expressing design agents which will, when complete, allow a designer to write a design problem-solver with only minimal assistance from the knowledge engineer. Our group has designed and implemented a language for building diagnostic systems[24].

```
PLAN
   NAME                Air Cylinder Design Plan
   TYPE                Design
   USED BY             Air Cylinder  SPECIALIST
   USES                Spring Head Rest SPECIALISTS
   QUALITY             Reliable BUT Expensive
   FINAL CONSTRAINTS   Design details OK?
   TO DO
        Validate and Process Requirements
        ROUGH DESIGN  Air Cylinder
        PARALLEL DESIGN  Spring AND Head
        TEST  Head and Spring Compatible?
        DESIGN  Rest
```

Figure 4: A Plan

For example, in figure 4 we show a plan, where "Validate and Process Requirements" is the name of a task, "Head and Spring Compatible?" is the name of a set of constraints, and "Rest" is the name of a specialist. Note that this is a design plan. Some specialists will also have rough-design plans. A task will consist of the sequential use of a number of steps, and a step consists of obtaining required information followed by calculations and a decision about the value of a single attribute.

```
STEP
   NAME                        Piston Seal Seat Width
   USED BY                     Piston Seal
   COMMENT                     Written by DCB, Sept. 83
   ATTRIBUTE NAME              Seal Seat Width
   FAILURE HANDLER
     FOR DECISION FAILURE      Piston Seal Seat Width FH
   FAILURE SUGGESTION          INCREASE Piston Thickness
   REDESIGN                    NOT POSSIBLE
   TO DO
     KNOWNS     FETCH Piston Thickness
                FETCH Piston Material
                FETCH Minimum Thickness
                     OF Piston Material
                FETCH Spring Seat Depth
     DECISION   Available IS
                  (Piston Thickness
                    MINUS DOUBLE Minimum Thickness)
                Seal Seat Width IS 0.156
                COMMENT Using one size only
                TEST Available > Seal Seat Width?
                STORE Seal Seat Width
```

Figure 5: A Step

Figure 5 shows a step to decide on the width of the seat for the piston seal, where "Piston Seal" is the name of a task, "Seal Seat Width" is what is being decided, "INCREASE Piston Thickness" is what the step will suggest if it's not possible to make a decision, "Piston Seal Seat Width FH" is the name of the failure handler that will analyze the problem and invoke the redesigner (not possible for this step), "Piston Thickness" is an attribute that should already have been decided, and "Available > Seal Seat Width" is the name of a constraint.

### 4.4. Example of AIR-CYL operation

The Air-cylinder plan given above calls for the use of the Head specialist. Part of the Head design plan is as follows:-

```
TO DO
     ....
     Air Cavity
     Air Inlet
     ....
```

The compressed air rushing into the AC through the Air Inlet is buffered slightly for better Piston activation by a chamber, called the Air Cavity, that is cut into the Head. The Air Cavity is designed to fit into the available space in the Head so as not to intrude on any other cut or component. The Air Inlet enters the Cavity through the top surface of the head, and is restricted by the size of the Air Cavity amongst other things. If the cavity is too small the inlet will not be able to enter it in a suitable way.

The Air-cylinder design plan, after requirements checking and rough design, will ask the head specialist to design the Head. A Head design plan will be selected -- let's presume for its "low cost" quality -- and design activity will proceed according to the plan. Eventually the Air Cavity will get designed by the Air Cavity task, and, when that has completed successfully, the Air Inlet task will be activated.

A major factor in determining the size of the cavity is how much material should be left between it and other cuts to allow for strength in operation and for manufacturability. Let us suppose that the low cost material chosen for the Head needs 1/8th of an inch clearance. The Air Inlet task discovers that the Air Cavity is too small for it to correctly position the inlet, and, after looking to see if some local changes can be made (ie. to the inlet), a step, and subsequently the task, will fail. This will cause the plan to fail. Amongst the suggestions of how to make the design succeed will be one that suggests that the Head's material be changed to one which requires a smaller clearance. The next plan to be activated will be the "strength/medium-cost" plan, which will use a harder more expensive material. This time the plan will succeed as clearance need only be 1/16th of an inch and the Air Cavity can be larger, enabling the Air Inlet to be positioned correctly.

An edited and annotated trace of the AIR-CYL system in its present version can be found in Appendix A.

## 5. Handling Failures

### 5.1. Our Philosophy

#### 5.1.1. Restricted Knowledge

Much of the work on failure handling in the literature considers all relevant knowledge to be available at failure time. If one views the problem solver's complete internal model as the "state-of-the-world", then, as one has complete knowledge of the form of the model and one knows that it completely captures the

state of the world, it is easy to do any kind of model manipulations that one desires[25]. If the state-of-the-world model is in fact an incomplete one, possibly augmented by "observations" from outside the model, then this kind of freedom of manipulation is not available. If, in addition, the model is structured in some way, so that at some failure point only part of the whole model is available (ie. that pertaining to just the most local problem-solving) then the manipulations are still further constrained. Our view is that the models of data and control in human problem-solving are structured and probably incomplete, thus restricting the kinds of information available for handling failures and the manipulations possible.

The structure of the design problem-solving system (ie. specialists, plans, tasks and steps) provides the context in which to structure failure handling. We will assume that at any point in the structure only the minimum knowledge is available locally about the problem-solving task. An agent knows about the following:-

- The agent that asked it to act
- The information passed to it by the calling agent
- The sub-agents it is able to use
- The information returned to it by the sub-agents that it has already called
- All the information about its own state
- The partial design as produced by prior agents

In addition, the specialist knows which plans are appropriate, which one has been selected, and how it is progressing. We will restrict the information passed to an agent from above to that which does not provide history but merely makes requests, provides requested information, gives suggestions, and passes constraints. The information received from sub-agents is restricted to reports of success or failure, and suggestions, with a minimum of information about what took place at the lower levels of the problem-solving structure, except where required by failure reporting.

### 5.1.2. Social Metaphor

We will continue to use the social metaphor when discussing the system -- that is, we can learn about possible behaviors of an agent in the system by considering it to be a person working in a design team organized with the same

structure as we are suggesting for the class 3 design system[2]. By using this idea, and the minimum-knowledge restriction discussed above, we hope to establish what is essential for failure handling in this kind of design activity.

This metaphor has proven very useful in our group's other work on problem solving, especially for diagnosis, where efficient knowledge structuring and control strategies can be observed in the medical community[17]. We feel that there is much to be gained by applying this strategy to the design domain too.

#### 5.1.3. Local Decisions

In this work we are trying to make sure that all design agents detect their own failure, are able to determine what went wrong (at least superficially), attempt to see if they can fix it locally, do so if they can, and report failure only if all attempts fail. Agents which have some control over other agents can use those agents in their attempt to correct the detected problem.

#### 5.1.4. Domain Driven

In general, any wholesale adoption of an AI mechanism will often lead one astray. For example, use of complete global dependency structures and pure dependency directed backtracking[3] is inappropriate in this design domain, as it would be unconstrained use of a mechanism in a way that did not reflect the structure of the problem-solving activity. This should not be interpreted to mean that we consider that belief revision behavior does not occur in humans, rather that analysis of the domain should lead one to it, and that the mechanisms are surface forms of richer problem-solving behavior.

### 5.2. Redesign

#### 5.2.1. How Redesign Occurs

Each kind of agent can have different kinds of reasons for failing. For example, a step finds that a decision violates some constraint, a task discovers that a step's failure can't be mended locally, a plan can fail if it is discovered that it's not applicable to the situation to which it is being applied, while a

---

[2] A discussion of this metaphor can be found in[26] and other papers in that issue.

[3] A clear and concise introduction to and bibliography for techniques such as these, referred to as "belief revision", can be found in[27].

specialist can fail if all of its plans fail.

For every kind of failure a message giving details is generated and passed back to the calling agent. The message includes, wherever possible, suggestions about what might be done to alleviate the problem. As there are usually many kinds of problems that can occur, an agent will first look at the message to decide what went on below. This is done by the Failure Handler associated with the agent. Much of the failure analysis is provided by the system, but for some cases, for example for constraint failures, the user (that is the person using the plan language to write a design system) has to supply some details. For some conditions immediate failure can be specified, for others an attempt to redesign might be made.

The agent also has associated with it a Redesigner -- except in the cases where redesign is not possible, and then this is specified in the declaration of the agent. Here too, much of the redesign activity will be provided by the system, but in some places the user needs to be quite specific. An adequate language for flexible control of redesign has yet to be developed and it remains one of our research goals. Consequently, for the most part redesign activity will initially be fairly inflexible. We hope, for example, to incorporate "advice" about how to proceed -- such as which suggestions should be ignored even though they appear relevant.

### 5.2.2. Design vs. Redesign

Design and Redesign are different problem-solving activities. We are also concerned with re-design — ie. design again. For example, if during failure handling so much has changed since the last attempt that it makes no sense to even try redesign, then design should be attempted again.

To see that design (or re-design) and redesign are clearly different consider the step. A step chooses the value for one attribute. For design a step is more-or-less pure calculation in most cases. This corresponds to a specific style of deciding a value that a person seems to have -- use known values, make a decision (calculation), check to see of it's ok, make a report. This is the style used during design and re-design.

In the case of redesign it seems that the step acts quite differently. The suggestions it receives guide the process of deciding on a value. Suggestions will, in some way, be 'clashed' against each other to produce a value or range of values. If a range is produced, then local knowledge is used to select a value from the range. The other possible result is that the suggestions are incompatible and that there is no value.

## 5.3. Agent Failure

### 5.3.1. Constraint Failure

Behind almost all failures is constraint failure. A constraint will collect information about the state of the design and will test some relationship between these attributes. A constraint failing will make suggestions about what could be causing the problem and will indicate the values that caused the failure, and what values would have caused success. These suggestions are entirely made on the basis of the form of the constraint, ie. entirely local. It is up to the agent in which the constraint is embedded to interpret these suggestions, and to make its own more informed suggestions.

### 5.3.2. Step Failure

A failing step will, via its failure handler, determine what to do next. If redesign is appropriate and possible then the redesigner for that step will be given any available information about the failure (eg. details from the failing constraint). Some of this redesign action will be provided by the system, and some must be specified by the expert system implementor. If the attempt succeeds then the step will return to its caller without any sign that a local failure occurred. If it fails, then the step will make its suggestions to the calling agent and include it with other suggestions it was given. There are interesting issues here about how much effort a step should expend before admitting failure, and how to reasonably capture that process in the system.

### 5.3.3. Task Failure

A task checks conditions on entry and exit, and executes a sequence of steps, with explicitly checked constraints. Any constraints that are tested between steps monitor the progress of the task. As currently implemented the task is an ordered sequence of steps with no possible variation. As the task represents the designer's concentrated effort in one local context (within the broader context of the selected plan) this fixed sequence is reasonable.

If one of the task's steps fails it will receive a failure message and a collections of suggestions. It knows which suggestions apply to which steps below it. Knowing which steps have already been activated it can therefore decide which of the steps referred to in the suggestions was active most recently. That step will then be asked to do a redesign. If it can, then there is a good chance that the failing step can now succeed, as long as the steps after the redesigning step but before the failing step are not disturbed by the redesign. They will be asked, and if there is a problem then some strategy will

be selected. This a matter for further research. Clearly failures can produces failures and so on, until the original purpose becomes obscured. Also, clearly, a human is not able to deal with such nested failures. The simplest strategy is to fail if an attempt to correct a failure itself fails. If an attempt to follow a suggestion fails, then another one is tried until all of the suggestions have been tried by the task. In that situation the task will fail, and itself make suggestions, passing on those to which it couldn't respond.

### 5.3.4. Plan Failure

A plan is currently implemented as an ordered sequence of actions. To be successful, all of the actions must succeed. Note that this doesn't imply that nothing failed at a lower level, but merely that eventually every action in the plan returned success. Other plans in other specialists may have failed on the way to this success. The details of what happened below is of essentially no concern to this specialist. We are still investigating the degree to which it is reasonable and possible to attempt failure handling at the plan level.

A plan in a specialist represents a method for designing that part, area, or function of the device. The plans provide alternative ways of achieving the same portion of the design, all expressed at the same level of detail. At lower levels other specialists will also make choices. If a plan fails, all the decisions made by the plan up to the point of failure must be retracted. That is, the "drawing" is returned to a previous version. On real-life drawings (blueprints) changes are accumulated on the drawing until enough exist to warrant producing a new version. In the design system, prior to every major activity, the current state of the data-base is given a version identifier. If the activity fails, the version need not be changed, otherwise the new version is adopted for active use, and the old one is stored away to provide a record of the history of the design.

### 5.3.5. Specialist Failure

Specialist failure on entry is a sign that conditions are in some way inappropriate for this specialist to act. For example, a specialist may only be able to design air-cylinders smaller than a breadbox. On entry to a specialist, prior to plan selection, one would expect an examination of the gross conditions of applicability, and if those are not met a failure message would be sent to the calling agent.

Specialist failure during plan selection can occur for a variety of reasons — there are no plans appropriate for this situation; all appropriate plans have been used during this call of the specialist. The first type of failure occurs

on entry to the specialist, after it has passed the applicability tests, if no plans are appropriate. The second kind of plan selection failure implies that any local attempts to remedy the problems causing plan failure also failed. Clearly the calling agent should be informed of this special situation, as it represents the complete failure of the specialist.

## 5.4. Other Research

### 5.4.1. DESI/NASL

In McDermott's DESI/NASL system[25, 9] failure handling is treated as just another task for the system to handle. The system is prevented from backing up over a previously made decision about a value. McDermott argues that, in general, backing up to a choice point using a universal mechanism is not appropriate, and that consideration must be given to all the actions and choices made since that point and prior to the failure. The failing primitive task produces a description of the failure, and a failure task is set up to attempt to deal with the situation. As well as that description, the task/subtask structure, the control trace and the recorded data dependencies are available for use. New subtasks may be added, old subtasks restarted, old subtasks re-expressed, or heuristics abandoned. This is a very powerful failure handling mechanism. We feel that it is too unconstrained and not structured enough, and more like a general programming mechanism than a specific theory of failure handling.

### 5.4.2. BUILD

In the BUILD planning system Fahlman[28] adopts an approach similar to McDermott's. After being frustrated by Micro-PLANNER's chronological backtracking Fahlman wrote the system in CONNIVER using control structures that allowed "the BUILD-PLACE-MOVE sequence to proceed in a headlong manner, with very little pre-checking of conditions", and reported that "trouble is met in a variety of ways when it arises". [p.117].

In the BUILD system, every function which makes a major choice includes the declaration of a "Gripe Handler". If the subgoal selected fails in some way the most local gripe handler is called with a failure message reporting the problem. The gripe handler has access to the full environment of the failure situation, and can also examine the bindings of the choice function. Should the decision be made that failure is due to some decision at a higher level it may complain to the next higher gripe handler, otherwise the problem is handled locally. The problems with this method of failure handling are the same as for the DESI/NASL

system. That is, there are no constraints on the backtracking done. The BUILD system in some ways is even less constrained than McDermott's system as it appears to be able to backtrack in any way over any prior decision. The failure handling behavior of the AIR-CYL system could probably be implemented in the mechanisms provided by Fahlman and McDermott.

### 5.4.3. EL/ARS

The EL/ARS system for electronic circuit analysis[29, 10] is a system that uses dependency records to keep track of failure situations. It avoids failure situations in further processing by using all of the recorded "NOGOOD" situations to affect the choice of actions. As not all NOGOOD situations are relevant for any one problem-solving situation this method is missing some essential structure and can be inefficient. The use of 'pure' dependency structures in a global way is unconstrained use of a technique, producing unstructured problem-solving activity.

### 5.4.4. TROPIC

Latombe's design system TROPIC[30, 7] uses dependency directed backtracking to the most recently contributing choice point. Failure information is attached to the chronological trace of control flow at a point singled out at backup time as being responsible for the failure. This is clearly better than pure chronological backtracking, and the controlled use of failure recording is to be preferred over the EL/ARS method. However, there is no control over the range of the backtracks made, as the system maintains a global dependency structure.

## 5.5. Summary

It appears that this extended analysis of failure handling within the framework of a class 3 design system has led us to a modified form of dependency-directed backtracking controlled by suggestions and failure-handling advice. This control regime is complicated by the notion of attempting repair of failures by doing redesign. The whole activity is made to work by using a data-base to represent versions of the drawing that record the state of the design, and by a variety of messages that produce data and control flow.

# 6. Theoretical and Practical Research Issues

### 6.1. Improvements to AIR-CYL

We have learnt much during the last year as a result of our efforts in implementing a prototype system[1] and the AIR-CYL expert system for a nontrivially complex mechanical engineering object. This implementation effort is not complete, but we now have a better understanding of the strengths of and problems with the approach. We feel that while the idea of design refinement captures the essence of design problem solving, at least in its relatively routine aspects, there are several important aspects of problem solving and plan specification that need continued investigation. In addition to those aspects discussed below, we hope to improve the interface with the system to allow others to use it. Eventually we expect to provide a graphical interface to show the development of the design as it progresses.

### 6.2. Plan Language

In our implementation, plans have shown a significant tendency to incorporate selection schemata and constraint-checking. In addition there is a distinct difference between specification checking, rough design, design and redesign activity. Thus there is a need to investigate the form and content of the plan structures that would be useful in class 3 design problems. In order for this notion of design plans to find much practical application in CAD it is necessary that generic representations for plans and their coordination be obtained. Updating the expert system to reflect changing products and any increase in a company's expertise will not be a practical undertaking unless such representations are available.

### 6.3. Problem-Solving

#### 6.3.1. Direction of Refinement

It may be the case that we can alter plans depending on experience, or select plans in a different order. This will have to be investigated closely to see exactly what factors influence such changes. Clearly if one plan consistently leads to failure then it should be treated as suspect, but it may merely mean that its conditions for selection have not been adequately captured. With parallel sections of design it may be useful to hold up one section until part of the other has succeeded. This would be useful if one subsection is particularly difficult to design, as it would save wasting effort on the other parallel section.

### 6.3.2. Plan Selection

The issue of how plans get selected is far from resolved. While it is clear that many pieces of information are involved we do not yet know how they are combined during selection, or what role suggestions, or the history of the design so far play in this selection. Plans themselves have information attached to them that may be used during the selection. For example, a plan may be known for its qualities, such as producing a subcomponent cheaply, or tending to succeed more than it fails.

### 6.3.3. Rough Design

There is one aspect of rough-design that we have not yet tried to capture and that is the "rough" value. For example, when designing we often start by assigning an attribute a small range of values -- "it's between 3 and 4 inches". Values in a design system may need to have qualities of "precision", "accuracy", and "confidence". Accuracy is captured at present by using a form that includes a value and its tolerances -- (LNGTH 2.3 0.001 0.02). Precision is to do with how precisely one is able to state the range of possible values -- less than 4. While confidence has to do with how allowable it is to alter that value -- "a good starting value for this is 3 inches + or - 1/10th, but if you want you can change it if it doesn't look right". Clearly these all interact in various ways, and will pose problems for the system -- for example, what's "roughly 3 but less than 4" plus (LNGTH 2.3 0.001 00.02)?

### 6.3.4. Relaxation of Requirements

One possible way to deal with failures is to attempt to relax one or more of the requirements. Clearly some requirements can be "softer" than others, and asking the user for some relaxation may clear the way to a successful design. If a lot of effort has been expended on a design by machine and human this makes a lot of sense. It may be possible for the system to choose requirements to relax, but a lot of special knowledge would be necessary to implement that. Even knowing when to ask for a relaxation will be difficult. This is a matter for future research.

### 6.3.5. Performance Degradation

We suspect that as designs get to be only just class 3 the performance of the system will degrade in interesting ways. If there are many dependencies between attributes then the system can be expected to fail more often. By observing this behavior, and the ease with which a design system for an object can be captured in the plan language, we hope to be able to make some observations about categories of design smaller than the classes that we have outlined.

### 6.3.6. Automatic Construction

Another interesting possibility for the system is the automatic ordering of steps and tasks. At present one has to be aware of the exact order that the designer uses in order to be sure that all required values are available before each agent acts. Due to the commonly observed difficulty in getting ones expert to recall exactly what he or she does, it will often be possible to get only a partial ordering. In fact, provided the dependencies are immediately available, or can be deduced, the agents could be presented to the system for it to decide on a reasonable order, in the case that the designer is is doubt or doesn't care.

## 6.4. Limitations of Approach

Of course, we are quite aware that there are bound to be other examples of Class 3 design tasks that cannot be brought under the plan refinement paradigm in a natural way. There is a distinction between plan refinements in the abstract and a particular generic class of plans which we know how to represent and refine and handle failures with. Thus, even if it is true in principle that design is a process of choosing and refining plans in the abstract, our ability to write expert systems for design is very much a function of the generic classes of plans which we are able to describe and manipulate. We would like, as a result of our research, to be able to characterize the kinds of design problems for which the plan refinement approach will lead to effective expert systems.

## 6.5. Functional Understanding

One of the projects that we are working on in our expert systems research concerns the representation of how a device functions[31]. It ought to be possible to cleanly derive a diagnostic structure from this representation. One of the tests of someone understanding a device is that person's ability to reason about any malfunction in that device. What is the relationship of this functional representation to the design representations that we are studying? One would expect that the designer's view of how and why he thinks that the design is satisfactory, from the viewpoint of meeting the functional specifications, should have close correspondences to the functional understanding of the diagnostician. Making these relationships clear will be an important theoretical undertaking. It will show how understanding how a device works, being able to design a device to fulfill certain functional requirements, and being able to trouble-shoot a device all share certain representations.

## 7. Concluding Remarks

We have presented an approach to building expert systems for a particular class of design activity in the domain of mechanical components. Much work remains to be done in this area before we understand what design is and how best to build systems to do it. However we feel that by using a hierarchically structured system with plan selection we are capturing the essential qualities of routine design, while discovering many interesting and difficult issues.

**Acknowledgment:** This work was supported by AFOSR grant #82-0255. We would also like to acknowledge the cooperation of the AccuRay Corporation and Pete Schmitz.

## 8. References

1. Brown, D. C. and Chandrasekaran, B., ``An approach to expert systems for mechanical design,'' Trends and Applications '83, IEEE Computer Society, NBS, Gaithersburg, MD, May 1983, pp. 173-180.

2. Fischer, G. and Bocker, H-D., ``The nature of design processes and how computer systems can support them,'' European Conference on Integrated Computing Systems, Sept 1982, pp. , Stresa, Italy

3. Grinberg, M. R., ``A knowledge-based design system for digital electronics,'' Proceedings 1st Annual National Conference on AI, AAAI, August 1980, pp. 283.

4. Mitchell, T. M., ``An Intelligent Aid for Circuit Redesign,'' Proceedings of AAAI Conference, AAAI, 1983, pp. 274-278.

5. Stefik, M. and Conway, L., ``Towards the principled engineering of knowledge,'' The AI Magazine, Vol. 3, No. 3, 1982, pp. 4, AAAI

6. McDermott, J., ``R1: a rule-based configurer of computer systems,'' Artificial Intelligence, Vol. 19, No. 1, Sept 1982, pp. 39-88.

7. Latombe, J-C., ``Failure processing in a system for designing complex assemblies,'' Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI, 1979, pp. 508-515.

8. Latombe, J-C., editor, Artificial Intelligence and Pattern Recognition in CAD, North-Holland, 1978, Proceedings IFIP Wkg. Conference, Grenoble, Fr., March 1978

9. McDermott, D.V., ``Circuit Design as problem solving,'' in AI and Pattern Recognition in CAD, Latombe, J-C, ed., North-Holland, 1978, pp. 227-245.

10. Sussman, G.J., ``Electrical Design -- a problem for AI research,'' Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI, 1977, pp. 894-900.

11. Eastman, C. M., ``The representation of design problems and maintenance of their structure,'' in AI and Pattern recognition in CAD, Latombe, J-C., ed., North-Holland, 1978, pp. 335-357.

12. Fenves, S.J. and Norabhoompipat, T., ``Potentials for AI applications in

Structural Engineering, Design and Detailing,'' in Artificial Intelligence and Pattern Recognition in CAD, Latombe, ed., N-H, 1978, pp. 105-119.

13. Freeman, P. and Newell, A., ``A model for functional reasoning in design,'' Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI, 1971, pp. 621.

14. Brown, D. C. and Chandrasekaran, B., ``Design considerations for picture production in a natural language graphics system,'' Computer Graphics, Vol. 15, No. 2, July 1981, pp. 174-207, SIGRAPH-ACM

15. Mittal, S. and Chandrasekaran, B., ``A conceptual representation of patient databases,'' Journal of Medical Systems, Vol. 4, No. 2, 1980, pp. 169-185.

16. Chandrasekaran, B., ``Towards a taxonomy of problem-solving types,'' AI Magazine, Vol. 4, No. 1, 1983, pp. 9-17.

17. Gomez, F. and Chandrasekaran, B., ``Knowledge organization and distribution for medical diagnosis,'' IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-11, No. 1, January 1981, pp. 34-42.

18. Lenat, D. B., ``Automated theory formation in mathematics,'' Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI, 1977, pp. 833-842.

19. Lenat, D. B., ``The nature of heuristics,'' Artificial Intelligence, Vol. 19, No. 2, 1982, pp. 189-249.

20. Hayes-Roth, B., ``Human Planning Processes,'' Tech. report R-2670-ONR, RAND Corp., Santa Monica, CA., 1980.

21. Chandrasekaran, B., Mittal, S. and Smith, J. W., RADEX - Towards a computer-based radiology consultant, North Holland Pub. Co., Amsterdam, Holland, 1980, pp. 463-474.

22. Simon, H. A., The Sciences of the Artificial, MIT Press, 1981, 1st Edition, 1969

23. Roberts, R. B. and Goldstein, I. P., ``The FRL Manual,'' AI memo 409, MIT, 1977.

24. Bylander, T., Mittal, S. and Chandrasekaran, B., ``CSRL: A language for expert systems for diagnosis,'' Proceedings of the International Joint Conference on Artificial Intelligence, August 1983, pp. 218-221.

25. McDermott, D.V., ``Flexibility and efficiency in a computer program for designing circuits,'' AI-TR 402, MIT, June 1977.

26. Chandrasekaran, B., ``Natural and social system metaphors for distributed problem solving: Introduction to the issue,'' IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-11, No. 1, January 1981, pp. 1-5.

27. Doyle, J. and London, P., ``A selected descriptor-indexed bibliography to the literature on Belief Revision,'' ACM SIGART, No. 71, April 1980, pp. 7.

28. Fahlman, S.E., ``A planning system for robot construction *tasks*,'' AI-TR 283, MIT, May 1973.

29. Stallman, R. and Sussman, G.J., ``Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis,'' Artificial

Intelligence, Vol. 9, 1977, pp. 135-196, also in MIT AI Lab memo. 380 -- 1976

30. Latombe, J-C., ``Artificial Intelligence in Computer-aided Design : the TROPIC system,'' Tech. report 125, Stanford Research Institute, February 1976.

31. Moorthy, V. S. and Chandrasekaran, B., ``Representation for functional understanding of devices and compilation of diagnostic systems from them,'' Proceedings of the MEDCOMP-83 conference on Computers in Medicine, MEDCOMP, 1983, pp. ???, To appear

## 9. Appendix A

### Annotated Trace of AIR-CYL

This is a trace generated by the system. It has been edited for brevity and presentation in this format. The trace was only turned on for the Specialists. The trace is of a successful design with no selection of alternative plans. It shows the attributes designed by the system.

```
***** AIR-CYL Air-cylinder Design System *****
*** Version date: (4 26 84)
*** Todays date:  (5 4 84)
*** User: DCBROWN

*** Requirements input
The following options are available :-
   1 --- to use a set of standard test/demo requirements
   2 --- to have requirements read from your disk file
   3 --- to type them all in yourself
Note: you will be able to make alterations
*** Please type the number of your option >>>????>1

* Standard test/demo requirements to be used
     From file DCB:AC-Requirements-Test.LSP
Requirements:001

!!! Note:
There are about 20 values given as requirements,
including the maximum operating temperature and
pressure, and the size of the envelope in which
the air-cylinder must fit.

* Do you wish to make alterations to the requirements?
     Please answer YES or NO or QUIT >>>????>n

* No alterations to be made
*** Requirements Input Complete

--- Entering Specialist
    ...AirCylinder...Mode = Design

!!! Note:
The first part of the design plan is to check
the requirements and then do a rough-design.
```

```
----- Entering Specialist
      ...AirCylinder...Mode = RoughDesign

------- Entering Specialist
        ...Head...Mode = RoughDesign

------- Leaving Specialist
        ....Head...Result= Success Msg

------- Entering Specialist
        ...Rest...Mode = RoughDesign

--------- Entering Specialist
          ...Cap...Mode = RoughDesign

--------- Leaving Specialist
          ....Cap...Result= Success Msg

--------- Entering Specialist
          ...Tube...Mode = RoughDesign

!!! Note:
At any point the system can ask the user for
a value, or ask for a value to be checked.
This is useful in situations where it is hard to
capture the judgment being made, or in
development situations where that part of
the design has yet has yet to be codified.

???? ASK-USER: Tube rough volume is 4.051
               Is this OK for this design?
Answer is ????>yes

--------- Leaving Specialist
          ....Tube...Result= Success Msg

--------- Entering Specialist
          ...PistonAndRod... Mode = RoughDesign

--------- Leaving Specialist
          ....PistonAndRod...Result= Success Msg

--------- Entering Specialist
          ...Bumper... Mode = RoughDesign

--------- Leaving Specialist
          ....Bumper...Result= Success Msg

------- Leaving Specialist
        ....Rest...Result= Success Msg

------- Entering Specialist
        ...Spring... Mode = RoughDesign

------- Leaving Specialist
        ....Spring...Result= Success Msg

----- Leaving Specialist
      ....AirCylinder...Result= Success Msg
```

```
!!! Note:
Now do the design.

----- Entering Specialist
      ...Spring... Mode = Design

???? ASK-VALUE: Spring Wire Diameter
Value is ????>.215

???? ASK-VALUE: Number of Coils
Value is ????>11

SpringMaterial          ---- NIL
SpringOD                ---- 0.985
SpringID                ---- 0.77
SpringWireDiameter      ---- 0.215
SpringFreeLength        ---- NIL
SpringCompressedLength  ---- NIL
SpringInstalledLength   ---- NIL
SpringLoad              ---- NIL
SpringNumberOfCoils     ---- 11
SpringDeflectionPerCoil ---- NIL

----- Leaving Specialist
      ....Spring...Result= Success Msg

----- Entering Specialist
      ...Head... Mode = Design

???? ASK-USER:
Head Air Cavity volume is 0.323
Is this OK for this design?
Answer is ????>ok

!!! Note:
The LNGTH form below is a way of expressing
tolerances. The first figure is the value,
the second the +ve tolerance, while the third
is the -ve tolerance.

HeadWidth      ---- 1.5
HeadDepth      ---- 0.97
HeadHeight     ---- 1.5
HeadMaterial   ---- StainlessSteel
HeadScrewSize  ---- (LNGTH 0.19 5.e-3 5.e-3)
HeadCenterCenterDistance
               ---- (LNGTH 0.625 5.e-3 5.e-3)
HeadMountingHoleDiameter
               ---- (LNGTH 0.206 3.e-3 0.0)
HeadCounterSinkDiameter
               ---- (LNGTH 0.37 1.e-2 1.e-2)
HeadMaxHtoFDistance
               ---- (LNGTH 0.31 5.e-3 5.e-3)
HeadMountingHolesToFaceDistance
               ---- (LNGTH 0.2455 2.5e-3 2.5e-3)
HeadWiperSeatDepth          ---- 0.175854
HeadWiperSeatDiameter       ---- 0.459841
HeadWiperType               ---- UCup
HeadAirHoleToSideDistance   ---- 0.75
HeadAirHoleToFaceDistance   ---- 0.701
```

```
HeadAirHoleDepth                ---- 0.2105
HeadAirHoleDiameter             ---- 0.374
HeadAirCavityID                 ---- 0.534
HeadAirCavityOD                 ---- 1.089
HeadAirCavityDepth              ---- 0.4565
HeadTRHCenterCenterDistance     ---- 1.115
HeadTRHDepth                    ---- NIL
HeadTRHDiameter                 ---- 0.19
HeadBearingThickness            ---- 4.85e-2
HeadBearing1Length              ---- 0.4765
HeadBearing2Length              ---- 0.182646
HeadSealSeatWidth
          ---- (LNGTH 0.125 5.e-3 5.e-3)
HeadSealSeatToFaceDistance      ---- 0.3585
HeadSealSeatDiameter
          ---- (LNGTH 0.5 3.e-3 0.0)
HeadTubeSeatID
          ---- (LNGTH 1.21 6.e-3 3.e-3)
HeadTubeSeatOD
          ---- (LNGTH 1.359 1.e-2 1.e-2)
HeadTubeSeatDepth               ---- 6.25d-2

!!! Note:
The system contains a table of standard decimal values
and is able to take the nearest higher or lower value,
or just the nearest.  For example, a value of 2.4936 can
be stored as 2.5, or as 2.4844 (ie. 31/64ths).

----- Leaving Specialist
      ....Head...Result= Success Msg

!!! Note:
Once the Head specialist is completed the
Rest specialist can start.

----- Entering Specialist
      ...Rest... Mode = Design

------- Entering Specialist
        ...PistonAndRod... Mode = Design

PistonDiameter
     ---- (LNGTH 1.212 4.e-3 0.0)
PistonMaterial                  ---- Brass
PistonThickness                 ---- 0.34375d
PistonRodHole                   ---- 0.25d
PistonSpringSeatDepth           ---- 3.9e-2
PistonSpringSeatID              ---- 0.754375
PistonSpringSeatOD              ---- 1.00062
PistonSealType                  ---- UCup
PistonSealSeatDiameter
     ---- (LNGTH 0.885 0.0 1.e-3 ThreeDP)
PistonSealSeatWidth
     ---- (LNGTH 0.156 1.e-2 1.e-2 ThreeDP)
PistonSealSeatPosition          ---- 9.4e-2
PistonBreakawayCutDiameter      ---- 0.729
PistonBreakawayCutDepth         ---- 3.4e-2
PistonNotchCount                ---- 4
PistonNotchWidth                ---- 7.8e-2
PistonNotchDepth                ---- 7.8e-2
```

```
PistonBrazeSeatDepth          ---- 7.8e-2
PistonBrazeSeatDiameter       ---- 0.390625d

RodDiameter ---- (LNGTH 0.312 0.0 2.e-3)
RodLength              ---- 4.095
RodThreadLength        ---- 1.031
RodThreadType          ---- UNF24
RodMaterial            ---- StainlessSteel
RodPistonSeatDiameter  ---- 0.247
RodPistonSeatLength    ---- 0.31
RodEndOfRodToHead      ---- 2.781

------- Leaving Specialist
        ....PistonAndRod...Result= Success Msg

------- Entering Specialist
        ...Cap... Mode = Design

CapMaterial          ---- StainlessSteel
CapHeight            ---- 1.5
CapWidth             ---- 1.5
CapDepth             ---- 0.625
CapInternalDepth     ---- 0.499
CapInternalDiameter  ---- 1.089
CapTubeSeatDepth     ---- 6.25d-2
CapTubeSeatID
           ---- (LNGTH 1.21 6.e-3 3.e-3)
CapTubeSeatOD
           ---- (LNGTH 1.359 1.e-2 1.e-2)
CapAirHoleDiameter   ---- 0.374
CapAirHoleCenterToBackDistance
                     ---- 0.313
CapAirHoleDepth      ---- 0.2055
CapBackFaceThickness ---- NIL
CapTRtoTRDistance    ---- 1.115
CapTRDiameter
           ---- (LNGTH 0.203 5.e-3 5.e-3)
CapTRDepth           ---- 0.3125
CapTRRecessDepth     ---- 0.3125
CapTRRecessRadius    ---- 1.003
CapLargeChamferWidth ---- NIL
CapLargeChamferAngle ---- NIL
CapSmallChamferWidth ---- NIL
CapSmallChamferAngle ---- NIL

------- Leaving Specialist
        ....Cap...Result= Success Msg

------- Entering Specialist
        ...Tube... Mode = Design

TubeMaterial       ---- StainlessSteel
TubeLength         ---- 3.5
TubeID             ---- 1.214
TubeOD             ---- 1.344
TubeChamferLength  ---- NIL
TubeChamferAngle   ---- NIL

------- Leaving Specialist
        ....Tube...Result= Success Msg
```

```
------- Entering Specialist
        ...Bumper... Mode = Design

BumperMaterial          ---- StainlessSteel
BumperLength            ---- NIL
BumperID                ---- 0.390625d
BumperOD                ---- 0.69
BumperFlangeDiameter    ---- 1.059
BumperFlangeThickness   ---- 6.25e-2

------- Leaving Specialist
        ....Bumper...Result= Success Msg

------ Leaving Specialist
       ....Rest...Result= Success Msg

--- Leaving Specialist
    ....AirCylinder...Result= Success Msg

*** Design attempt succeeds
*** Version date: (4 26 84)
*** Todays date:  (5 4 84)
*** User: DCBROWN
```

# APPENDIX C

## CSRL: A LANGUAGE FOR EXPERT SYSTEMS FOR DIAGNOSIS

To appear in the Special Issue of Intn'l Jrnl. of Computers and Mathematics on "practical artificial intelligence systems."

# CSRL: A Language for Expert Systems for Diagnosis*

Tom Bylander, Sanjay Mittal**, and B. Chandrasekaran
Artificial Intelligence Group
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210 USA

## Abstract

We present CSRL (Conceptual Structures Representation Language) as a language to facilitate the development of expert diagnosis systems based on a paradigm of "cooperating diagnostic specialists." In our approach, diagnostic reasoning is one of several generic tasks, each of which calls for a particular organizational and problem solving structure. A diagnostic structure is composed of a collection of specialists, each of which corresponds to a potential hypothesis about the current case. They are organized as a classification or diagnostic hierarchy, e.g., a classification of diseases. A top-down strategy called establish-refine is used in which either a specialist establishes and then refines itself, or the specialist rejects itself, pruning the hierarchy that it heads. CSRL is a language for representing the specialists of a diagnostic hierarchy and the diagnostic knowledge within them. The diagnostic knowledge is encoded at various levels of abstractions: message procedures, which describe the specialist's behavior in response to messages from other specialists; knowledge groups, which determine how data relate to features of the hypothesis; and rule-like knowledge, which is contained within knowledge groups.

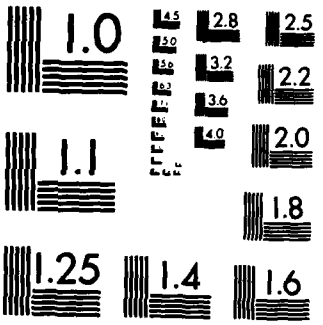
---

*This an expanded version of a paper of the same title which was presented at the 1983 International Joint Conference on Artificial Intelligence

**Currently at Knowledge Systems Area, Xerox PARC, 3333 Coyote Hill Rd., Palo Alto, CA 94304 USA

AD-A146 890 DISTRIBUTED KNOWLEDGE BASE SYSTEMS FOR DIAGNOSIS AND INFORMATION RETRIEVAL(U) OHIO STATE UNIV RESEARCH FOUNDATION COLUMBUS B CHANDRASEKARAN AUG 84 2/2

UNCLASSIFIED AFOSR-TR-84-0864 AFOSR-82-0255 F/G 9/2 NL




OPY RESOLUTION TEST CHART

# CSRL: A Language for Expert Systems for Diagnosis*

Tom Bylander, Sanjay Mittal**, and B. Chandrasekaran
Artificial Intelligence Group
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210 USA

## 1 Introduction

Many kinds of problem solving for expert systems have been proposed within the AI community. Whatever the approach, there is a need to acquire the knowledge in a given domain and implement it in the spirit of the problem solving paradigm. Reducing the time to implement a system usually involves the creation of a high level language which reflects the intended method of problem solving. For example, EMYCIN [1] was created for building systems based on MYCIN-like problem solving [2]. Such languages are also intended to speed up the knowledge acquisition process by allowing domain experts to input knowledge in a form close to their conceptual level. Another goal is to make it easier to enforce consistency between the expert's knowledge and its implementation.

CSRL (Conceptual Structures Representation Language) is a language for implementing expert diagnostic systems that are based on our approach to

---

*This an expanded version of a paper of the same title which was presented at the 1983 International Joint Conference on Artificial Intelligence.

**Currently at Knowledge Systems Area, Xerox PARC, 3333 Coyote Hill Rd., Palo Alto, CA 94304 USA

diagnostic problem solving. This approach is an outgrowth of our group's experience with MDX, a medical diagnostic program [3], and with applying MDX-like problem solving to other medical and non-medical domains. CSRL facilitates the development of diagnostic systems by supporting constructs which represent diagnostic knowledge at appropriate levels of abstraction.

First, we will overview the relationship of CSRL to our overall theory of problem solving types and the diagnostic problem solving that underlies CSRL. We then present CSRL, illustrating how its constructs are used to encode diagnostic knowledge. Two expert systems under development in our laboratory which use CSRL are then briefly described. Based on our experience with these systems, we point out where improvements in CSRL are needed.

## 2 Classificatory Diagnosis

The central problem solving of diagnosis, in our view, is classificatory activity. This is a specific type of problem solving in our approach, meaning that a special kind of organization and special strategies are strongly associated with performing expert diagnosis. In this section, we will briefly review the theory of problem solving types as presented by Chandrasekaran [4], and the structure and strategies of the diagnostic task [5].

### 2.1 Types of Problem Solving

We propose that expert problem solving is composed of a collection of different problem solving abilities. The AI group at Ohio State has been working at identifying well-defined types of problem solving (called generic tasks), one of which is classificatory diagnosis. (For the purposes of this discussion, we will use "diagnosis" in place of "classificatory diagnosis" with the understanding that the complete diagnostic process includes other

elements as well.) Other examples include knowledge-directed data retrieval, consequence finding, and a restricted form of design.

Each generic task calls for a particular organizational and problem solving structure. Given a specific kind of task to perform, the idea is that specific ways to organize and use knowledge are ideally suited for that task.

Even when the specification of a problem is reduced to a given task within a given domain, the amount of knowledge which is needed can still be enormous (e.g., diagnosis in medicine). In our approach, the knowledge structure for a given task and domain is composed of specialists, each of which specialize in different concepts of the domain. Domain knowledge is distributed across the specialists, dividing the problem into more manageable parts, and organizing the knowledge into chunks which become relevant when the corresponding concepts become relevant during the problem solving.

Decomposing a domain into specialists raises the problem of how they will coordinate during the problem solving process. First, the specialists as a whole are organized, primarily around the "subspecialist-of" relationship. Each task may specify additional relationships that may hold between specialists. Second, each task is associated with a set of strategies which take advantage of these relationships and the problem solving capabilities of the individual specialists. The choice of what strategy to follow is not a global decision, but chosen by the specialists during problem solving.

## 2.2 The Diagnostic Task

The diagnostic task is the identification of a case description with a specific node in a pre-determined diagnostic hierarchy. Each node in the hierarchy corresponds to a hypothesis about the current case. Nodes higher in the hierarchy represent more general hypotheses, while lower nodes are more specific. Typically, a diagnostic hierarchy is a classification of malfunctions of some object, and the case description contains the manifestations and background information about the object. For example, the Auto-Mech expert system [6] attempts to classify data concerning an automobile into a diagnostic hierarchy of fuel system malfunctions. Figure 1 illustrates a fragment of Auto-Mech's hierarchy. The most general node, the fuel system in this example, is the head node of hierarchy. More specific fuel system malfunctions such as fuel delivery problems are classified within the hierarchy.

PUT FIGURE 1 HERE

Each node in the hierarchy is associated with a specialist which contains the diagnostic knowledge to evaluate the plausibility of the hypothesis from the case description. From this knowledge, the specialist determines a confidence value representing the amount of belief in the hypothesis. If this value is high enough, the specialist is said to be established.

The basic strategy of the diagnostic task is a process of hypothesis refinement, which we call establish-refine. In this strategy, if a specialist establishes itself, then it refines the hypothesis by invoking its subspecialists, which also perform the establish-refine strategy. If its confidence value is low, the specialist rejects the hypothesis, and performs no further actions. Note that when this happens, the whole hierarchy below

the specialist is eliminated from consideration. Otherwise the specialist suspends itself, and may later refine itself if its superior requests it.

With regard to figure 1, the following scenario might occur. First, the fuel system specialist is invoked, since it is the top specialist in the hierarchy. This specialist is then established, and the two specialists below it are invoked. Bad fuel problems is rejected, eliminating the three subspecialists of bad fuel from consideration. Finally, the fuel mixture specialist is established, and its subspecialists (not shown) are invoked.

An important companion to the diagnostic hierarchy is an intelligent data base assistant which organizes the case description, answers queries from the diagnostic specialists, and makes simple inferences from the data [7]. For example, the data base should be able to infer that the fuel tank is not empty if the car can be started. The diagnostic specialists are then relieved from knowing all the ways that a particular datum could be inferred from other data.

There are several issues relevant to diagnostic problem solving which we will not address here. The simple description above does not employ strategies for bypassing the hierarchical structure for common malfunctions, for handling multiple interacting hypothesis, or for accounting of the manifestations. Also, additional control strategies are required when many nodes are in a suspended state. For discussion on some of these topics, see Gomez and Chandrasekaran [5]. Test ordering, causal explanation of findings, and therapeutic action do not directly fall within the auspices of the classificatory diagnosis as defined here, but expertise in any of these areas would certainly enhance a diagnostic system. Fully resolving all of these

issues and integrating their solutions into the diagnostic framework are problems for future research.

### 2.3 Differences from other Approaches

The usual approach to building knowledge based systems is to emphasize a general knowledge representation structure and different problem solvers which use that knowledge. One difference in this approach is that the organization of knowledge is not intended as a general representation for all problems. Rather it is tuned specifically for diagnosis. By limiting the type of problem to be solved, a specific organizational technique (classification hierarchy) and problem solving strategy (establish-refine) can be used to provide focus and control in the problem solving process.

Another difference is that the specialists in the hierarchy are not a static collection of knowledge. The knowledge of how to establish or reject is embedded within the specialists. Each specialist can then be viewed as a individual problem solver with its own knowledge base. The entire collection of specialists engages in distributed problem-solving.

## 3 CSRL

CSRL is a language for representing the specialists of a diagnostic hierarchy and the diagnostic knowledge within them. The diagnostic knowledge is encoded at various levels of abstractions. Message procedures describe the specialist's behavior in response to messages from other specialists. These contain the knowledge about how to establish or refine a specialist. Knowledge groups determine how selected data relate to various features or intermediate hypotheses that are related to the specialist. The selected data may be the values of other knowledge groups, so that a single knowledge group

can "summarize" the results of several others. Knowledge groups are composed of rule-like knowledge which match the data against specific patterns, and when successful, provide values to be processed by the knowledge group.

### 3.1 Specialists

In CSRL, a diagnostic expert system is implemented by individually defining each specialist. The super- and subspecialists of the specialist are declared within the definition. Figure 2 is a skeleton of a specialist definition for the *Bad Fuel node* from figure 1. The declare section specifies its relationships to other specialists. The other sections of the specialist are examined below.

PUT FIGURE 2 HERE

Since CSRL is designed to use only a simple classification tree, many choices concerning the composition of the hierarchy must be made. This is a pragmatic decision, rather than a search for the "perfect" classification tree. The main criteria for evaluating a classification is whether enough evidence is normally available to make confident decisions. To decompose a specialist into its subspecialists, the simplest method is to ask the domain expert what subhypotheses should be considered next. Usually the subspecialists will differ from one another based on a single attribute (e.g., location, cause). For further discussion on this and other design decisions in CSRL, see Bylander and Smith [8].

### 3.2 Message Procedures

The messages section of a specialist contains a list of message procedures, which specify how the specialist will respond to different messages from its superspecialist.* "Establish", "Refine", "Establish-Refine" (combines Establish and Refine), and "Suggest" are predefined messages in CSRL; additional messages may be defined by the user. Below, we will examine how Establish and Refine procedures are typically constructed.

Message procedures are the highest level of abstraction for diagnostic knowledge within specialists. Just as in general message passing languages, messages provide a way to invoke a particular kind of response without having to know what procedure to invoke. Strategies for diagnosis, such as establish-refine, are usually easy to translate into a message protocol. However, CSRL does not provide any way to specify and enforce message protocols.

Figure 3 illustrates the Establish message procedure of the BadFuel specialist. "relevant" and "summary" are names of knowledge groups of BadFuel. "self" is a keyword which refers to the name of the specialist. This procedure first tests the value of the relevant knowledge group. (If this knowledge group has not already been executed, it is automatically executed at this point.) If it is greater than or equal to 0, then BadFuel's confidence value is set to the value of the summary knowledge group, else it

---

*A specialist is not allowed to send messages to its superspecialist. However, other message passing routes are allowed. Specifically, a specialist may send a message to itself, across the hierarchy, and to indirect subspecialists. In the latter case, each interconnecting specialist is sent a "Suggest" message and decides within its Suggest message procedure whether or not to pass the original message downwards.

is set to the value of the relevant knowledge group. In CSRL, a confidence value scale of -3 to +3 is used (integers only). A value of +2 or +3 indicates that the specialist is established. In this case, the procedure corresponds to the following diagnostic knowledge.

> First perform a preliminary check to make sure that BadFuel is a relevant hypothesis to hold. If it is not (the relevant knowledge group is less than 0), then set BadFuel's confidence value to the degree of relevancy. Otherwise, perform more complicated reasoning (the summary knowledge group combines the values of other knowledge groups) to determine BadFuel's confidence value.

PUT FIGURE 3 HERE

Figure 4 shows a Refine procedure which is a simplified version of the one that BadFuel uses. "subspecialists" is a keyword which refers to the subspecialists of the current specialist. The procedure calls each subspecialist with an Establish message.* If the subspecialist establishes itself (+? tests if the confidence value is +2 or +3), then send it a Refine message.

PUT FIGURE 4 HERE

CSRL has a variety of other kinds of statements and expressions so that more complicated strategies can be implemented. For example, a "Reset" statement deletes the confidence value and the knowledge group values of a specialist. This might be used when additional tests are performed, making it necessary to recalculate the confidence value. Also, messages can be

---

*For convenience, many of CSRL's control constructs mimic those of INTERLISP; however, these constructs are executed by the CSRL interpreter, not by using LISP EVAL. LISP code is allowed within message procedures, but only by within a construct called "DoLisp". This is not intended to let specialists have arbitrary code, but to allow interaction with other LISP-implemented systems.

parameterized and message procedures can declare local variables.

### 3.3 Knowledge Groups

The kgs section of a specialist definition contains a list of knowledge groups, which are used to evaluate how selected data indicate various features or intermediate hypotheses that relate to specialist's hypothesis. A knowledge group can be thought of as a cluster of production rules which map the values of a list of expressions (boolean and arithmetic operations on data) to some conclusion on a discrete, symbolic scale. Different types of knowledge groups perform this mapping differently, e.g., directly mapping values to conclusions, or having each rule add or subtract a set number of "confidence" units.

Knowledge groups are intended for encoding the heuristics that a domain expert uses for inferring features of a hypothesis from the case description. The main problem is that this inference is uncertain -- there is rarely a one-to-one mapping from data to the features of the hypothesis. The way that this is handled in CSRL is borrowed from the uncertainty handling techniques used in MDX [9].

Each feature or intermediate hypothesis is associated with a knowledge group. The data that the domain expert uses to evaluate the feature is encoded as expressions in the knowledge group. These are usually queries to a separate data base system. Each combination of values of the expressions is then mapped to a level of confidence as determined by the domain expert. This set of knowledge groups becomes the data for another knowledge group, which determines the confidence value of the specialist from the confidence values

of the features.* By examining the results of test cases, the knowledge groups are relatively easy to debug since the attention of the domain expert can be directed to the specific area of knowledge which derived the incorrect result.

As an example, figure 5 is the relevant knowledge group of the BadFuel specialist mentioned above. It determines whether the symptoms of the automobile are consistent with bad fuel problems. The expressions query the user (who is the data base for Auto-Mech) for whether the car is slow to respond, starts hard, has knocking or pinging sounds, or has the problem when accelerating. "AskYNU?" is a LISP function which asks the user for a Y, N, or U (unknown) answer from the user, and translates the answer into T, F, or U, the values of CSRL's three-valued logic. Each set of tests in the if-then part of the knowledge group is evaluated until one matches. The value corresponding to this "rule" becomes the value of the knowledge group. For example, the first rule tests whether the first expression is true (the "?" means doesn't matter). If so, then -3 becomes the value of the knowledge group. Otherwise, other rules are evaluated. The value of the knowledge group will be 1 if no rule matches. This knowledge group encodes the following diagnostic knowledge:

> If the car is slow to respond or if the car starts hard, then BadFuel is not relevant in this case. Otherwise, if there are knocking or pinging sounds and if the problem occurs while accelerating, then BadFuel is highly relevant. In all other cases, BadFuel is only mildly relevant.

PUT FIGURE 5 HERE

Figure 6 is the summary knowledge group of BadFuel. Its expressions are

---

*Actually, any number of knowledge group levels can be implemented.

the values of the relevant and gas knowledge group (the latter queries the user about the temporal relationship between the onset of the problem and when gas was last bought). In this case, if the value of the relevant knowledge group is 3 and the value of the gas knowledge group is greater then or equal to 0, then the value of the summary knowledge group (and consequently the confidence value of BadFuel) is 3, indicating that a bad fuel problem is very likely.

PUT FIGURE 6 HERE

### 3.4 Comparison with Rule-Based Languages

There is nothing in CSRL that is not programmable within rule-based languages such as OPS5 [10] or EMYCIN [1]. The difference between CSRL and these languages is that CSRL makes a commitment to a particular organizational and programming style. CSRL is not intended to be a general purpose representation language, but is built specifically for the classificatory diagnosis problem. It is possible to program in a rule-based language so that there is an implicit relationship between rules so that they correspond to knowledge groups and specialists. R1, although not a diagnostic expert system, is an excellent example of how one creates implicit grouping of rules in such a system [11]. The central idea underlying CSRL is to make these relationships explicit. The expert system implementor is then relieved from trying to impose an organization on a organization-less system and is free to concentrate on the conceptual structure of the domain. Also, there is a greater potential to embed explanation and debugging facilities which can take advantage of the expert system organization.

### 3.5 The CSRL Environment

The current version of CSRL is implemented in INTERLISP-D and LOOPS, an object-oriented programming tool. Each specialist is implemented as a LOOPS class, which is instantiated for each case that is run. The LOOPS class hierarchy is used to specify default message procedures and shared knowledge groups, making it easy to encode a default establish-refine strategy, and letting the user incrementally modify this strategy and add strategies as desired. A graphical interface displays the specialist hierarchy, and through the use of a mouse, allows the user to easily access and modify any part of the hierarchy. Additional facilities for debugging and explanation are being implemented.

## 4 Expert Systems that use CSRL

### 4.1 Auto-Mech

Auto-Mech is an expert system which diagnoses fuel problems in automobile engines [6]. This domain was chosen to demonstrate the viability of our approach to non-medical domains, as well as to gain experience and feedback on CSRL.* The purpose of the fuel system is to deliver a mixture of fuel and air to the air cylinders of the engine. It can be divided into major subsystems (fuel delivery, air intake, carburetor, vacuum manifold) which correspond to initial hypotheses about fuel system faults.

Auto-Mech consists of 34 CSRL specialists in a hierarchy which varies from four to six levels deep. Its problem solving closely follows the establish-refine strategy. Before this strategy is invoked, Auto-Mech collects some

---

*Auto-Mech was developed using an early version of the language.

initial data from the user. This includes the major symptom that the user notices (such as stalling) and the situation when this occurs (e.g., accelerating and cold engine temperature). Any additional questions are asked while Auto-Mech's specialists are running. The diagnosis then starts and continues until the user is satisfied that the diagnosis is complete. The user must make this decision since the data that Auto-Mech uses are very weak at indicating specific problems and, more importantly, Auto-Mech is unable to make the repair and determine whether the problem has been fixed.

A major part of Auto-Mech's development was determining the assumptions that would be made about the design of the automobile engine and the data that the program would be using. Different automobile engine designs have a significant effect on the hypotheses that are considered. A carbureted engine, for example, will have a different set of problems than a fuel injected engine (the former can have a broken carburetor). The data was assumed to come from commonly available resources. The variety of computer analysis information that is available to mechanics today was not considered in order to simplify building Auto-Mech.

### 4.2 Red

Red is an expert system whose domain is red blood cell antibody identification [12]. An everyday problem that a blood bank contends with is the selection of units of blood for transfusion during major surgery. The primary difficulty is that antibodies in the patient's blood may attack the foreign blood, rendering the new blood useless as well as presenting additional danger to the patient. Thus identifying the patient's antibodies and selecting blood which will not react with them is a critical task for nearly all red blood transfusions.

The Red expert system is composed of three major subsystems, one of which is implemented in CSRL. The non-CSRL subsystems are a data base which maintains and answers questions about reaction records (reactions of the patient's blood in selected blood samples under a variety of conditions), and a overview system, which assembles a composite hypothesis of the antibodies that would best explain the reaction record [13]. CSRL is used to implement specialists corresponding to each antibody that Red knows about (about 30 of the most common ones) and to each antibody subtype (different ways that the antibody can react).

The major function of the specialists is to rule out antibodies and their subtypes whenever possible, thus simplifying the job of the overview subsystem, and to assign confidence values, informing overview of which antibodies appear to be more plausible. The specialists query the data base for information about the test reactions and other patient information, and also tell the data base to perform certain operations on reaction records.

An interesting feature of Red is how it handles the problem of interacting hypotheses. It is possible for the patient's blood to have practically any number or combination of antibodies, making it very hard for a single specialist to determine how well it will fit with other specialists in a composite hypothesis. In Red, each specialist is encoded to assume that it is independent — it looks at the data as if no other specialist can account for the same data. The knowledge of how the specialists can interact is left to the overview subsystem. This would be problematic if few specialists could rule themselves out, but so happens that in this domain, it is rare to have more than a few antibodies that cannot be independently ruled out. Thus Red's CSRL subsystem makes overview's problem solving computationally feasible since

it considerably reduces the amount of search that would otherwise be necessary.

## 5 Needed Improvements in CSRL

The largest flaw in CSRL is that there is no strategy that determines when diagnosis should stop. Currently, the default procedures simply ask the user if the current diagnosis is satisfactory. Some notion of what it means to account for the data needs to be added to the language. The work on Red's overview system is a step in this direction, but there needs to be more integration of overview and CSRL (currently overview starts after the specialists are finished), and a better understanding of what kinds of interactions can occur between two hypotheses. Progress in this area would also help increase the focus of the diagnosis, i.e., the diagnosis could concentrate on accounting for the most important manifestation(s).

Another problem is the meaning of the confidence value of a specialist. In MDX, this value was directly associated with the amount of belief in the specialist. However in both Auto-Mech and Red, this meaning had to be slightly altered to fit the purposes of the expert system. In Auto-Mech the confidence value is used to indicate whether the hypothesis was worth pursuing. In Red it is used to indicate the specialist's plausibility given the independence assumption mentioned earlier. It is not possible in either expert system to confirm a specialist without outside help. In Auto-Mech a repair or highly specific test must be performed while in Red all the specialists must be considered together. This does not create a problem for the process of establish-refine problem solving, but makes it difficult to explain what the confidence value means. Any explanation facility must understand the assumptions that are being made to make coherent explanations.

## 6 Conclusion

We believe that the development of complex expert systems will depend on the availability of special purpose languages with organizational and problem-solving tools that match the conceptual structure of the domain. CSRL represents an initial step in this direction. It provides facilities to organize diagnostic knowledge in accordance with the structure of the domain. In particular, CSRL's constructs facilitate the encoding of rule-like and strategic knowledge into appropriate abstractions: knowledge groups, message procedures, and specialists.

## ACKNOWLEDGMENTS

We would like to acknowledge Jack Smith and Jon Sticklen for many fruitful discussions concerning CSRL's design. Many improvements in the language are due to Mike Tanner and John Josephson, who implemented the CSRL specialists in Auto-Mech and Red. The language development is funded by a grant from the Battelle Memorial Laboratories University Distribution Program, and experimentation and application in different domains is supported by AFOSR grant 82-0255, and NSF grant MCS-8103480.

## REFERENCES

1. W. van Melle. A Domain Independent Production-Rule System for Consultation Programs. Proc. Sixth International Conf. on Artificial Intelligence, Tokyo, 1979, pp. 923-925.

2. E. H. Shortliffe. Computer-Based Medical Consultations: MYCIN. Elsevier, New York, 1976.

3. B. Chandrasekaran and S. Mittal. Conceptual Representation of Medical Knowledge for Diagnosis by Computer: MDX and Related Systems. In Advances in Computers, Academic Press, New York, 1983, pp. 217-293.

4. B. Chandrasekaran. "Towards a Taxonomy of Problem Solving Types." AI Magazine 4, 1 (Winter/Spring 1983), 9-17.

5. F. Gomez and B. Chandrasekaran. "Knowledge Organization and Distribution for Medical Diagnosis." IEEE Trans. Systems, Man and Cybernetics SMC-11, 1 (January 1981), 34-42.

6. M. C. Tanner and T. Bylander. Application of the CSRL Language to the Design of Expert Diagnosis Systems: The Auto-Mech Experience. to appear in Proc. of the Joint Services Workshop on Artificial Intelligence in Maintenence, 1984.

7. S. Mittal and B. Chandrasekaran. "Conceptual Representation of Patient Data Bases." J. of Medical Systems 4, 2 (1980), 169-185.

8. T. Bylander, and J. W. Smith. Using CSRL for Medical Diagnosis. Proc. Second International Conf. on Medical Computer Science and Computational Medicine, IEEEComputerSociety, Glouster, Ohio, 1983.

9. B. Chandrasekaran, S. Mittal, and J. W. Smith. Reasoning with Uncertain Knowledge: The MDX Approach. Proc. Congress of American Medical Infomatics Association, San Francisco, 1982.

10. C. L. Forgy. OPS5 Users Manual. Tech. Rept. CMU-CS-81-135, Carnegie-Mellon University, 1981.

11. J. McDermott. "R1: A Rule-based Configurer of Computer Systems." Artificial Intelligence 19, 1 (1982), 39-88.

12. J. W. Smith, J. Josephson, C. Evans, P. Straum, and J.Noga. Design For a Red-Cell Antibody Identification Expert. Proc. Second International Conf. on Medical Computer Science and Computational Medicine, IEEE Computer Society, Glouster, Ohio, 1983.

13. J. Josephson, B. Chandrasekaran, and J. W. Smith. The Overview Function in Diagnostic Problem Solving. Technical Paper, AI Group, Dept. of Computer and Information Science, The Ohio State University, 1984.

FIGURE 1

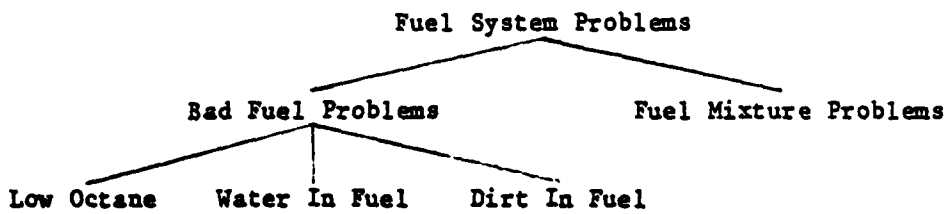



FIGURE 2

```
(Specialist BadFuel
    (declare (superspecialist FuelSystem)
             (subspecialists LowOctane WaterInFuel DirtInFuel))
    (kgs ...)
    (messages ...))
```

FIGURE 3

```
(Establish (if (GE relevant 0)
               then (SetConfidence self summary)
             else (SetConfidence self relevant)))
```

FIGURE 4

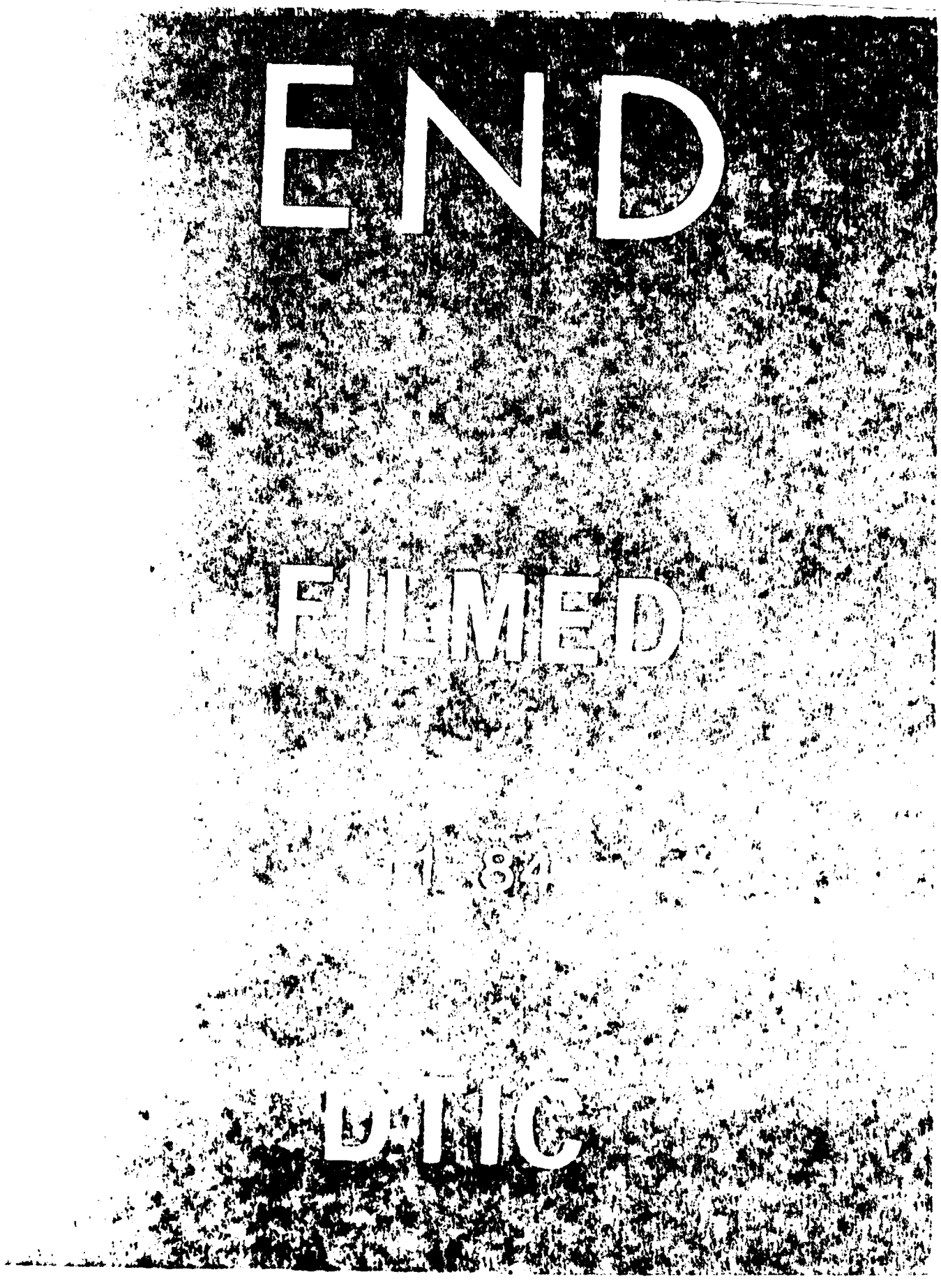
```
(Refine (for specialist in subspecialists
           do (Call specialist with Establish)
              (if (+? specialist)
                  then (Call specialist with Refine))))
```

FIGURE 5

```
(relevant Table
     (match (AskYNU? "Is the car slow to respond")
            (AskYNU? "Does the car start hard")
            (And (AskYNU? "Do you hear knocking or pinging sounds")
                 (AskYNU? "Does the problem occur while accelerating"))
       with (if T ? ?
                then -3
              elseif ? T ?
                then -3
              elseif ? ? T
                then 3
              else 1)))
```

FIGURE 6

```
(summary Table
    (match relevant gas
      with (if 3 (GE 0)
               then 3
             elseif 1 (GE 0)
               then 2
             elseif ? (LT 0)
               then -3)))
```


END
FILMED
8-84
DTIC