To the Graduate Council:

I am submitting herewith a thesis written by Shelley R. Heard entitled "DECIDE: A Decision Expression Interpreter." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

J. R. B. Cockett

J. Robin B. Cockett, Major Professor

We have read this thesis
and recommend its acceptance:

Michael G. Thomason

Doyt L. Perry

Accepted for the Council:

Lew Minkel

The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville. I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without permission, provided that accurate acknowledgement of source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence. by the Head of Interlibrary Services when, in the option of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Shelley R. Heard

Date March 18, 1985

DECIDE: A DECISION EXPRESSION INTERPRETER

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Shelley R. Heard

June 1985

To my Parents:

Richard and Alice Heard

ACKNOWLEDGEMENTS

I would like to thank the many people who have helped me, both educationally and inspirationally, in my pursuit of a Master of Science Degree in Computer Science. I am appreciative of my thesis committee, Dr. J. Robin B. Cockett, Dr. Doyt Perry, and Dr. Michael Thomason, for their constructive criticism and guidance. I am especially grateful to Dr. Cockett for the many hours he patiently spent teaching me to think and program as a knowledge engineer.

Mike Dotson and Heather Drummond contributed to the writing of the early versions of the DECIDE program. Dr. Cockett helped design versions 1 and 2 of DECIDE and developed the code for the file handling routine and DECIDE language interpreter in version 3 of DECIDE.

Dr. Ed Schilling of the Department of Botany, UTK, acted as a critic and expert consultant in the implementation of the TAXON knowledge base.

Bates Stephens made CProlog version 1.2a operational on the VAX-11 780 in the winter of 1984. This allowed work on this thesis to be conducted in a research environment.

My friend and colleague Ann Wayburn gave technical advice in the art of thesis writing. My dear friend Noppawan Tanpipat was an inspiration to me.

I would like to thank the staff of the University of Tennessee Computing Center (UTCC), and especially Loretta Haack, for technical assistance. I also appreciate the financial and educational benefits I received during my employment at UTCC with both the Operations and the User Services groups.

I would also like to thank the staff of UTK's Computer Science Department for giving me advice and assistance in the preparation of this thesis. Ethel Wittenberg was especially helpful.

Dr. Kevin O'Kane and Vivien Goff were very helpful and instructive in the use of the VAX-11 780 computing facilities in the UTK Computer Science Department's Computer Laboratory at UTK. I appreciate their interest and support.

Dr. Doug Birdwell, of the UTK Electrical Engineering Department, and Dr. Cockett provided me with a Graduate Research Assistantship for my work on DECIDE version 3 and enhancements to the CProlog interpreter used to implement the program.

This work was partially supported by Martin Marietta Energy Systems, Inc. under subcontract no. 37B-07685C, project authorization S13, at the University of Tennessee, Knoxville.

ABSTRACT

Decision trees are a form of knowledge representation that have lost popularity with the successes achieved by rule-based systems. One drawback to decision tree-based knowledge systems is their lack of hierarchical structure, which is essential for the implementation and maintenance of large scale knowledge systems (Woods). Also, explanations are difficult to generate because it is hard to focus the attention of a program on the problem at hand when a decision tree methodology is used.

Dr. J. Robin B. Cockett's recent work on decision theory and decision expressions (Cockett 1983a, 1983b, 1984) lead to the development, by the author and Dr. Cockett, of a decision expression interpreter called DECIDE. An augmented decision tree was used to represent knowledge compactly. Decision expressions were used to encode the knowledge in these augmented decision trees. The DECIDE language allows knowledge to be programmed hierarchically as knowledge clusters. Clustering provides a basis for both file handling and explanation generation. The DECIDE program manages a knowledge base of these clusters. It guides the user through the decision process to solve the user's problem. DECIDE also provides the user with a friendly and helpful environment in which to work.

TABLE OF CONTENTS

CHAPTER PAGE

LIST OF FIGURES

INTRODUCTION

> The knowledge engineer . . . works intensively with an expert to acquire domain-specific knowledge and organize it for use by a program. Simultaneously she is matching the tools of the AI work bench to the task at hand - program organization, methods of symbolic inference. techniques for the structuring of symbolic information, and the like. If the tool fits, or nearly fits, she uses it. If not, necessity mothers AI invention, and a new tool gets created. She builds the early versions of the intelligent agent, guided always by her intent that the program eventually achieve expert levels of performance in the task. She refines or reconceptualizes the system as the increasing amount of acquired knowledge causes the AI tool to 'break' or slow down intolerably. She also refines the human interface to the intelligent agent with several aims: to make the system appear 'comfortable' to the human user in his linguistic transaction with it: to make the system inference processes understandable to the user . . . (Feigenbaum).

The DECIDE Project

This thesis is a survey of the DECIDE project which involved the development and implementation of an expert system building tool (DECIDE) based on a decision expression methodology developed by Dr. J. Robin B. Cockett. Its primary participants were Dr. Cockett and the author. Dr. Cockett directed the project and was primarily responsible for the overall design of the DECIDE. The author was responsible for the implementation of version 2 of DECIDE and a substantial knowledge base (TAXON) under this version. The author also helped design and implemented the user interface of version 3 of DECIDE.

The History of the Development of DECIDE

During the winter and spring quarters of 1983 Dr. Cockett directed a course entitled "Special Topics in Expert Systems" at the University of Tennessee, Knoxville. A major goal of the course was to design, develop and implement a simple expert system. Five things were needed to meet this goal: 1) a data structure for representing knowledge, 2) a syntax for codifying knowledge, 3) a program for interpreting this codified knowledge, 4) an interface between a user and this interpreter, and 5) a knowledge base to implement under this program as an expert system.

The Prototype

A decision tree was used as the data structure for representing knowledge. Knowledge codification was accomplished by first arranging knowledge in the form of a decision tree and then representing that tree with a series of decision expressions. The syntax for decision expressions was developed by Dr. Cockett based on his decision theory (Cockett 1983b). These decision expressions allowed the questions, answers, and conclusions of a single decision tree to be codified. A decision expression interpreter called DECIDE was developed to access this codified knowledge and provide an interface with the user. A botanical taxonomic key was chosen as a knowledge base to test the prototype.

The earliest version of DECIDE has written in Prolog by Mike Dotson, a student of Dr. Cockett's, during the winter of 1983. It provided a primitive user interface which would publish questions, read answers from the user and publish conclusions when they were reached. This prototype also allowed a user to backtrack in the decision tree. This was possible because of a built-in tracing facility which kept a complete history of the user's interaction within the program. Input from the user could be in the form of an answer to the question or a system request, such as for backtracking.

The author developed the first knowledge base for DECIDE which was based on an already published botanical taxonomic key to trees native to Tennessee. This was convenient since taxonomic keys are already written in the form of decision trees. However, because of memory limitations, only a small portion of the knowledge base could be implemented using the initial version of DECIDE.

This version of DECIDE had obvious inadequacies. Clearly a file management system was needed so that the entire knowledge base would not have to reside in memory. Otherwise, the amount of memory available would always limit the size of the expert system that could be implemented. Also, the user interface lacked many features that would be helpful to users. An explanation system was needed to help users understand why the system was asking a question and

how a conclusion was reached. Also, a problem intrinsic to all decision-tree-based systems would have to be resolved - this was the deterministic and inflexible nature of decision trees. Every question in an interaction had to be answered in order to proceed to the later questions and hence to obtain a conclusion. If a user could not answer a question then the user could not proceed. A good system would have to deal with this problem. Finally, before a large scale expert systems could be developed for this system, a scheme would have to be devised to segment knowledge into meaningful partitions or clusters. This scheme should work hand in hand with the file handling system to allow efficient access to the knowledge base.

Version 2

After the conclusion of the Expert Systems course the author began work on this Masters thesis, under the direction of Dr. Cockett, which had as its goal the implementation of an expert system based on the decision theory that was developing. This prompted the development of an improved version of DECIDE. Version 2 of DECIDE was written by the author under the direction of Dr. Cockett and was strongly influenced by Dotson's original program. It resolved several of the problems encountered in version 1. It also resulted in the implementation by the author of TAXON, an expert system whose domain was the entire

botanical key mentioned above. The source code for version 2 of DECIDE can be found in Appendix D of Heather Drummond's Masters Thesis (Drummond).

The author developed a file management system for DECIDE based on the strategy of requiring only a small portion of the knowledge base (decision tree) to be stored in memory at one time. The rest of the knowledge base would reside in files stored on disk. This meant that disk space, not memory, was the factor which limited knowledge base size. This allowed much larger knowledge bases to be implemented under DECIDE.

The author made several enhancements to the user interface of version 2 of DECIDE. One significant improvement was the flagging of system requests. This was done by preceding all system commands with the escape character. (The escape character will hereafter be denoted <ESC>). This removed any possible ambiguity between user responses intended as answers to questions and those which were intended as system commands.

A typographical error detection facility was also installed to help match a user's response to an answer to the current question or to one of the system commands. It performed three basic functions. First, it reduced all upper case letters in the input string to lower case so that the interpreter would not be case sensitive. Second, it treated each input string as a prefix and compared it to

members of the current question's answer list or to the system commands list to determine a unique match. Ambiguities were published if they existed and the question reasked. Third, if no match could be made an algorithm for detecting single typographical errors was invoked. If a unique match could be made using this algorithm that string was published for the user's information.

In addition to the backtracking command from version 1, several new system functions were implemented in version 2 to enhance the user interface. System commands were installed to allow some canned questions to be asked of DECIDE which might provide helpful information to the user. These new commands provided access to a help facility, a glossary program, a routine which published the possible answers to the current question, and a program allowing the user to write messages to a system log file.

In Version 1 of DECIDE if a user was unable to answer a question then the interaction could not continue. This was due to the decision tree methodology. As mentioned above, decision trees are deterministic and require that a choice be made at every node for a conclusion to be reached. Version 2 of DECIDE included a technique to help solve this problem. If a user could not answer the current question then the user could use the system command "<ESC>?" to indicate to the system that the user did not know the answer to the current question. This would transfer control of the

system to another decision tree. The intent of this scheme was to allow a decision node to be substituted by a decision tree. The conclusions of this decision tree would be the answers to the question that labeled the decision node. The system would thus allow the user to answer the question that he was previously unable to answer by answering simple questions.

The basic idea of node substitution was not new. Akers (1978) describes a similar process, which he calls node decomposition, for representing boolean functions in binary decision trees. In Moret (1980) node decomposition of K-ary decision trees is discussed. Moret describes such a decision tree model as a "functional hierarchy."

In version 2 the implementation of substitution did not enforce the relationship between the substituted question and the decision process which replaced it. The effect was only achieved by careful construction of the knowledge base by the knowledge engineer.

The use of this technique permitted a knowledge base to be organized as a hierarchy of decisions. Towards the top of this hierarchy would be the most general questions. Towards the bottom of the hierarchy questions would be more specific or detailed. Questions at one level could be replaced by decision trees at the next lower level.

One advantage of organizing the knowledge base in this fashion was that a user who is knowledgeable can bypass

large portions of the decision tree by answering the more general questions. However, the novice user could receive the additional interaction required in order to reach a conclusion.

It was soon realized that this organization was fundamental to the workings of an expert system based on this methodology. The eventual design of version 3 of DECIDE was heavily influenced by this realization.

Knowledge Bases Under Version 2

TAXON's was the first substantial knowledge base implemented using DECIDE. It was based on the taxonomic key published in "Key to Summer Trees of Tennessee" (Shanks 1963). TAXON was able to identify about two hundred plant species and subspecies. The decision tree in TAXON was arranged hierarchically. General questions, such as "To what group of genera does this plant belong?" and "To which genus does the plant belong?", were asked of the user but substitution decision trees were available for answering these general questions.

SALES was a limited legal expert system implemented under DECIDE version 2 by Heather Drummond as part of her Masters thesis at the University of Tennessee, Knoxville. SALES's knowledge domain is Article 2 of the Uniform Commercial Code, U. C. C. SALES would first determine if Article 2 of the U. C. C. applied to the user's situation.

If it did, SALES would then provided legal advice to the user concerning this situation (Drummond).

Drummond made two significant improvements to the DECIDE program. The first was a functional explanation system. This system would retrieve any questions that the user had answered "<ESC>?" to and which had not yet been resolved by a substitute decision and would tell the user that the current interaction would resolve that question. She also included a summary facility which was invoked by a system command. This facility would publish all the questions and answers stored in DECIDE's trace (Drummond).

Version 3

In the winter of 1984 the author and Dr. Cockett began the development of a third version of DECIDE. Version 2 had been written in parallel with the development of the theory it was designed to implement. Updating that program was achieved through a rather unsatisfactory patching of its code. This resulted in a program which was very inefficient and increasingly difficult to debug. Version 3 was designed to include developments in Dr. Cockett's decision theory that had not been implemented in version 2, including a better syntax for decision expressions. The system facilities were now written in the DECIDE language itself. This gave the DECIDE program control over all interactions,

whether they were generated from the knowledge base or from one of the system facilities.

As of this writing version 3 of DECIDE is operational. However, it continues to be embellishment. Improvements, which include a natural language text generator and the ability to monitor processes spawned from the DECIDE program, are being implemented by Dr. Cockett. Version 3 is currently being used to implement a knowledge-based expert system to aid in the design and analysis of control systems (Birdwell 1985a, 1985b).

An Overview of Chapters

Chapter 1 contains a discussion of knowledge representation techniques including production rules, decision trees, and an augmented decision tree developed by Dr. Cockett. Also discussed is the use of decision expressions developed by Dr. Cockett for encoding knowledge in the form of the augmented decision trees. This chapter defines many of the terms and concepts that are used in succeeding chapters.

Chapter 2 describes the techniques developed by Dr. Cockett and the author to organize and utilize knowledge bases using the DECIDE program. It is shown that knowledge can easily organized in a hierarchical fashion using the technique called knowledge clustering. The strong

association between file management, explanation generation, knowledge clustering is also described.

Chapter 3 discusses the organization and use of version 3 of the DECIDE program. The general features of the system are described including the utilities and system commands that enhance the user interface. Chapter 4 concludes the thesis.

Examples of interactions with the DECIDE systems are provided in Appendix A and Appendix B. The interaction in Appendix A is derived from the TAXON knowledge base implemented by the author under DECIDE version 2. The interaction in Appendix B is derived from a simple knowledge base developed under DECIDE version 3 to demonstrate the various capabilities of the system.

CHAPTER 1

KNOWLEDGE REPRESENTATION AND DECISION EXPRESSIONS

All expert systems have two fundamental components: a knowledge base and an inference engine (Negoita). The knowledge base (data) contains the rules and facts about a particular domain. The inference engine (program) uses these rules and information supplied by interaction with a user to draw conclusions. Although the real power of an expert system is derived from its expert knowledge (Hayes-Roth 1983) good methods of representing and organizing this knowledge are particularly important to the implementation of a large expert system (Hayes-Roth 1984). This chapter discusses briefly knowledge representation schemes in use today and the decision expression methodology used in the DECIDE system. Also, the syntax and terminology of decision expressions, as used in version 3 of DECIDE, are defined and examples of their use are discussed.

Knowledge Representation

Production Rules

The most popular method of representing knowledge is by use of production rules. A production rule is composed of a premise and a conclusion. These rules have the form

IF <premise> THEN <conclusion>

One of the best studied and evaluated production-rule-based system is MYCIN, which diagnoses and prescribes treatment for bacterial infections of the blood. MYCIN's knowledge base consists of about 200 production rules, each of which is in the form of an IF-THEN statement. Each rule has associated with it a "certainty factor" which indicates the strength of the rule. A typical MYCIN rule is:

IF the infection type is primarily-bacteremia,
the suspected entry point is the gastrointestinal tract,
and site of the culture is one of the sterile sites,

THEN there is evidence (0.7) that the organism is bacteroides (Davis 1977).

A premise is represented internally as a combination of predicate functions, each of which returns a value between -1.0 and +1.0. A value of -1.0 indicates 100% certainly that the condition is FALSE and a value of +1.0 indicates 100% certainty that the condition is TRUE. Values between -0.2 and +0.2 indicate uncertainty (Davis 1977). Multivalued analogues of the Boolean AND and OR operations are used to evaluate a premise. For each rule whose premise evaluates successfully (greater than +0.2) the associated conclusion is made with a calculated certainty of

<premise-value> * <certainly value>

The rules of MYCIN define an AND/OR goal tree. The root represents a goal or hypothesis and each branch in the tree represents a clause in the premise (Figure 1-1). Arrows in these trees are directed towards the root.

MYCIN implements backward chaining to achieve hypothesis testing. A hypothesis is picked as a goal and a depth first search of the corresponding AND/OR goal tree is performed. Each leaf node encountered in the search is evaluated either by interaction with a user or from inferences in the knowledge base. Backward chaining allows a production system to focus its attention on proving a single hypothesis at a time. The line of questioning is thus directed towards proving that hypothesis (Winston 1977).

Backward chaining also provides a method for explanations since the program knows what hypothesis it is testing, what information has been provided in defense of that hypothesis, and what facts in the chain are left to be proved. Inquiries to the system fall into two natural categories: WHY a question was asked, and HOW a conclusion was (or will be) reached (Davis 1977).

Knowledge systems based on production rules exhibit several problems and limitations. Davis (1977) notes that back chaining makes makes it very difficult to control the order in which questions are asked. He also notes that because the system was not designed to use "pure" predicates

(a) IF the infection type is primarily-bacteremia,
the suspected entry point is the gastrointestinal tract,
and site of the culture is one of the sterile sites,

THEN there is evidence that the organism is bacteroides (Davis 1977).

(b)

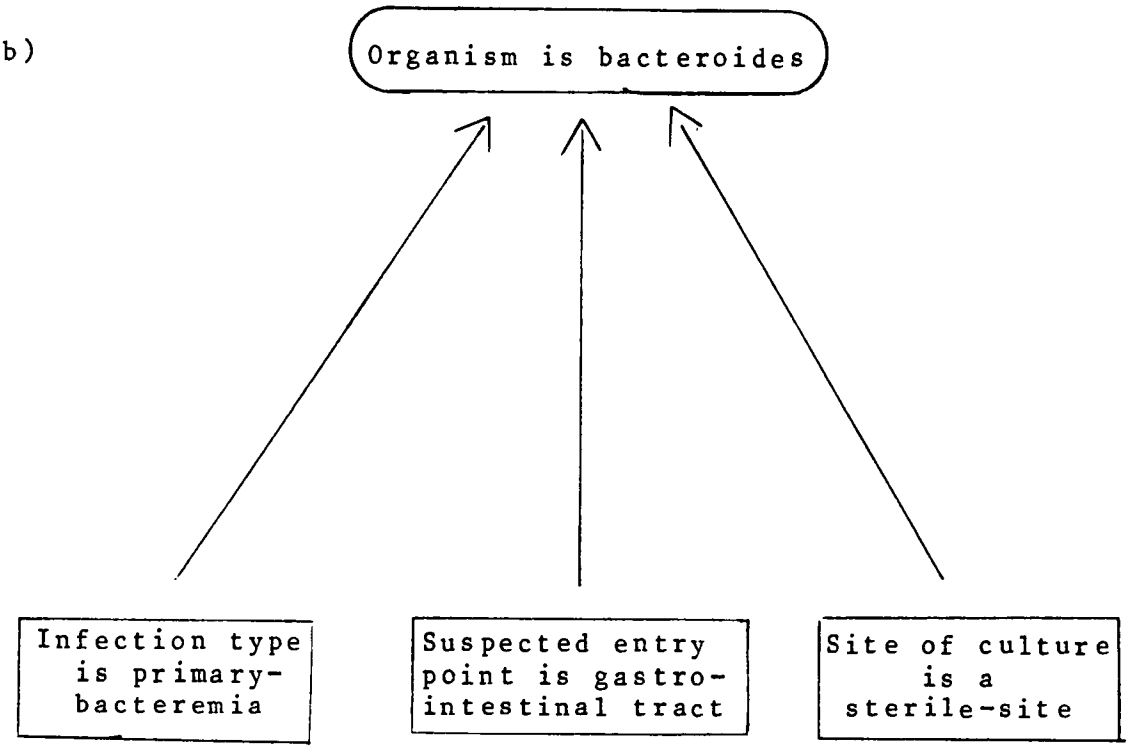

Figure 1-1: A Rule and AND/OR Goal Tree in MYCIN. An IF-THEN rule (a) from MYCIN (Davis 1977) and the AND/OR tree (b) that represents it. In this example three pieces of evidence (leaves) support a hypothesis (root). This simple tree has only one node, an AND node.

(those that return only TRUE or FALSE values) difficulties arise when such predicates are used. Most notably, explanations about such rules are often "murky."

Problems also arise when large numbers of rules are used. In systems with large numbers of rules it is difficult to predict the side effects of the addition of a new rule into the knowledge base (Winston 1977). Also, in a large system exhaustive invocation of rules is not feasible. The solution to this problem has been the use of meta-rules (rules about rules) which help prune the AND/OR tree for the goal being considered (Davis 1977). However, these meta-rules are themselves invoked exhaustively and potentially suffer from the same problems.

A final problem is that the attention of a production rule system is focused only on whatever hypothesis is been tested. Explanations available to the user (WHY and HOW above) tell the user what hypothesis is being considered but do not explain what the entire scope of the problem is. Also, if the hypothesis being tested is not what the user is considering, then the explanation system may not be helpful.

Decision Trees

Another, though less popular, form of knowledge representation is the decision tree. A decision tree is a labeled tree. All non-leaf nodes are decisions. These nodes are labeled with questions. Arrows that leave a node

are the answers to the question that labels that node. Leaf nodes are labeled by conclusions. As an example, Figure 1-2 shows a decision tree which describes the proper action to take at a traffic light.

Each node-arrow pair in a decision tree is a fact. For example. consider an arrow A that leaves node Q. Let Q be the question "how many legs does it have?" and let A be the answer "four." IF question Q is answered with answer A then a couplet <Q,A> is the fact "the animal has four legs."

Each path in a decision tree from root to leaf may be regarded as an IF-THEN rule. In this respect decision trees are like production rules. Unlike the AND/OR goal trees that represent production rules, the arrows in a decision tree are directed towards the leaves (conclusions). Many conclusions are represented in one decision tree whereas only one (the goal) is represented in the AND/OR tree.

Decision trees provide a useful model for applications which require discrete functions to be applied sequentially. Such applications include pattern recognition, taxonomy. logic design,fault detection and algorithm analysis (Moret).

Applying decision tree methodology to knowledge (expert) systems, has two immediate advantages over production rules. First, the use of production rules does not allow the knowledge programmer to easily dictate the order that questions are asked. Decision trees require that questions be ordered. Also, production-rule-based systems

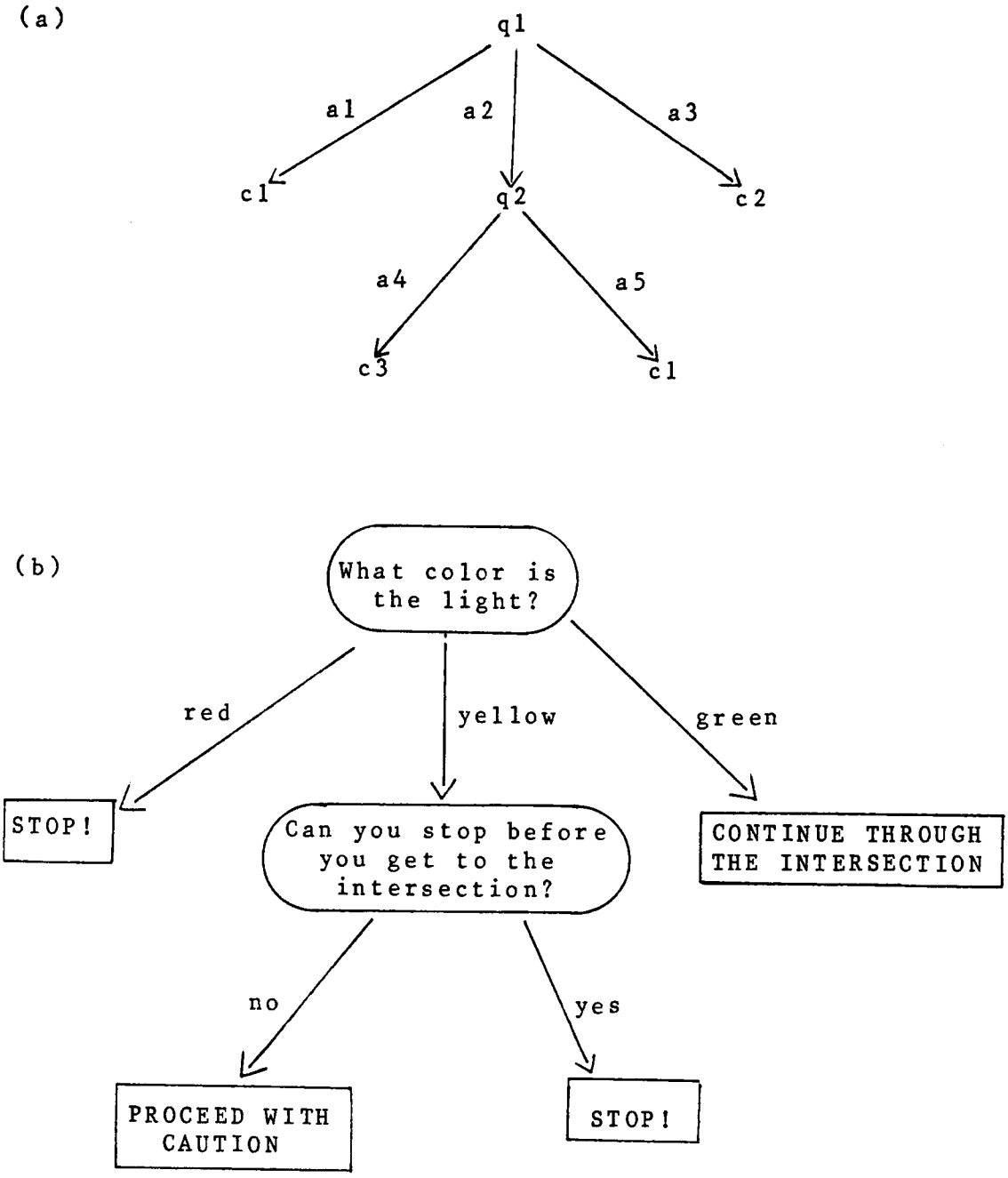

Figure 1-2: Decision Trees. (a) A model decision tree with questions {q1,q2}, answers {a1,a2,a3.a4,a5}, and conclusions {c1.c2.c3}. (b) A decision tree with the same structure as (a) but with text added to express its meaning. In this case the decision tree describes what action is appropriate to take at a traffic intersection.

sequentially test hypotheses until a correct one is found. This can cause questions to be asked which are not relevant to the final results. Decision trees can be arranged by the knowledge programmer to provide an acceptable (optimal or suboptimal) path to a correct conclusion.

The decision tree model has several problems which make it an unpopular form of knowledge representation. They are inherently complex. The fact that decision tree optimization is an NP-complete problem (Moret) is evidence of this complexity. It has been noted that knowledge in the form of even a medium sized decision tree is too difficult for humans to understand (Cohen). Another drawback to the use of decision trees is that certain types of questions are impractical or impossible to represent using the traditional decision tree model. This is because some questions have a large or infinite set of answers. It is tedious, if possible, to provide such a decision with a branch and an outcome for each answer. Therefore, there is a practical limit to the number of arrows that can leave a node in a decision tree. It is also difficult to generate meaningful explanations from decision trees. Decision trees also lack modularity, which makes effective management of a knowledge base difficult.

Historically decision trees have received very little attention as a form of knowledge representation. The most commonly cited reasons for disregarding the decision tree

model are those outlined in the preceding paragraph. It seems that the successes of rule-based systems has made production rules a paradigm of knowledge representation in the AI community. Many surveys on the state of knowledge representation all but ignore the decision tree model (Hayes-Roth 1982, 1984; Mylopolous; Weiss). Hayes-Roth (1984) describes the decision tree methodology as a "generally obsolete concern." In its treatment of knowledge representation, the Handbook of AI, volume I (Barr), ignores altogether the decision tree model.

In light of recent work on decision trees (Akers, Moret, Cockett 1983a. 1983b, 1984), the justifications for ignoring this representation scheme are called into question. Akers uses a hierarchically structured decision tree model for describing and testing large digital functions (Akers, summarized in Moret). Dr. Cockett and the author have shown, through the implementation of the DECIDE program, that a decision expression methodology is a practical and powerful tool for knowledge representation. These advances in decision tree methodology are the result of augmentations to the basic decision tree model which are described below.

Augmented Decision Trees

Dr. Cockett realized that if a decision tree methodology was to be used for representing large knowledge

systems, then the basic structure of the decision tree model would have to be augmented. The augmentations that he proposed, and that were later implemented in DECIDE version 3. were designed to accomplish four goals. First, the structure of the model would have to be hierarchical to provide a modular organization for the knowledge represented within it. Such organization such provide a basis for explanation generation and file handling. Second, the model should allow any question to be represented, even those with a large or infinite number of answers. Third, the model would have to be made more flexible to enhanced the expressive power of the system. This could be achieved by parametizing the decision tree. Finally, the model would have to be easily represented (codified) so that a program could easily access the knowledge. The methods and results of Dr. Cockett's work to achieve these goals are outlined below.

Hierarchical decision trees. In the DECIDE decision tree model hierarchical structure is achieved by applying the concept of decision substitution. Decision substitution allows any question in a decision tree to be replaced by a second decision tree whose conclusions match those of the replaced question. In the DECIDE system these substituted decision trees form modular clusters of knowledge. Each of these "knowledge clusters" has a governing question, that

is, the question that the decision tree replaces. The attention of the system is focused on answering unresolved governing questions. This provides a basis for an explanation system. Knowledge clustering also provides the basis for file handling in the DECIDE system. Each cluster is stored in a separate file and each governing question can reference the file that contains its cluster. Chapter 2 contains a more detailed description of knowledge clustering.

Answer types. As mentioned above, one of the problems that traditional decision trees have is the inability to easily represent questions that have a very large number of answers. Consider the simple though inefficient decision which decides if a number is divisible by 2. The question, "What is the number?", is asked. There is an infinite number of possibles answers to this question. Arrows labeled with 1,3,5,... lead to the conclusion that the number is not divisible by 2. Arrows labeled with 2,4,6,... lead to the conclusion that the number is divisible by 2. If the answer "5" is given, then the conclusion "5 is not divisible by 2" is reached. If the answer "4" is provided, then the conclusion "4 is divisible by 2" is reached (see Figure 1-3a).

This problem is solved in DECIDE's augmented decision tree model by allowing a single arrow in a decision tree to

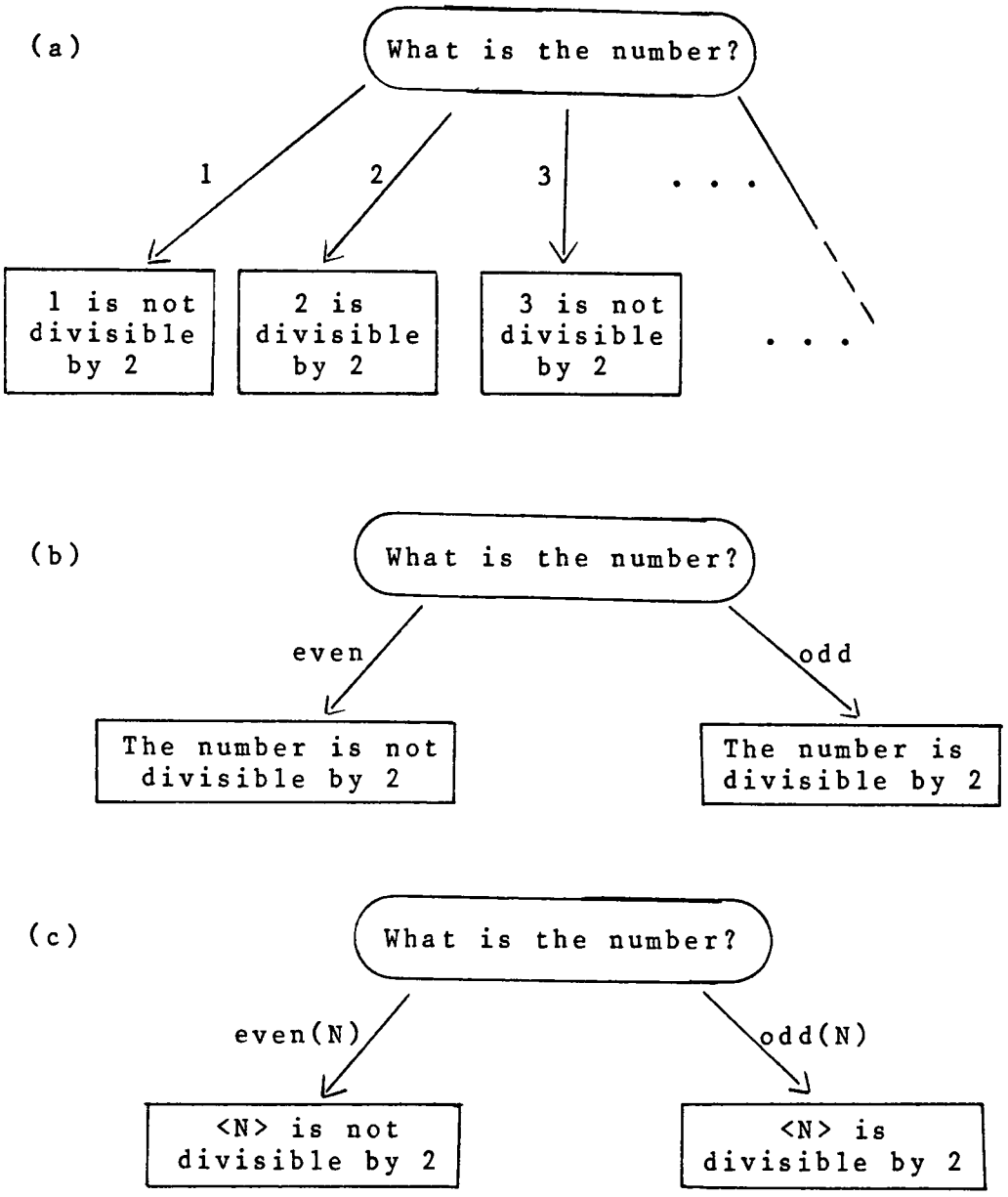

Figure 1-3: Answer Types and Answer Variables: (a) a decision tree with a branch for every answer {1,2,...}. (b) answer types "odd" and "even" used to represent sets of answers with common outcomes. (c) answer variable N is used to represent the member of an answer type and its value is assigned to the argument <N> of the outcomes. This gives the model (c) the same expressive power as the cumbersome and impractical model in (a).

represent a set of answers to a question which share a common outcome. This greatly reduces the width of such a decision tree. Such arrows are labeled with answer types. In the example above (Figure 1-3a), it is not the number itself that is important to making the immediate decision but rather which of the sets, {1,3,5,...} or {2,4,6...}, it belongs to. The answer types "odd" and "even" can be assigned to the members of the sets {1,3,5,...} and {2.4,6...}, respectively. Then the decision tree can be rewritten with only two arrows leaving the question. One arrow is labeled with the type "odd" and the other with the type "even" (see Figure 1-3b).

Sometimes the answer type may be important to the immediate decision, but the exact answer of the question is needed for context or for future decisions. Answer types are thus allowed to contain variables, called answer variables, to hold these values. The use of answer variables is described in the following section on parametized decision trees.

Parametized decision trees. Parametizing the decision tree model involves allowing nodes in the tree to have one or more arguments and allowing these arguments to be passed down the tree to succeeding nodes. In the DECIDE system such arguments are called context variables because they provide context to decisions and allow a particular

attribute of a decision to be represented as a variable. The use of context variables in decision trees allows general rules of knowledge to be represented. Assigning values to these variables allows the knowledge to be applied in a specific context. Also, the values of the context variables can be incorporated into the text of questions, conclusions, and explanations.

Assignment of values to context variables is achieved in two ways. First, the value can be passed from the node preceding it in the tree if that node contains the same context variable. Alternatively, the value of a context variable in a node can be derived from the answer type which label the arrow pointing to that node if that answer type contains a an answer variable of the same name as the context variable. Figure 1-3(c) shows how an answer variable is used to assign a value to a context variable. Figure 1-4 shows a complete augmented decision tree and demonstrates the use of decision substitution, context variables, and answer variables.

Decision Expressions

Dr. Cockett has developed a syntax for decision expressions for the express purpose of encoding knowledge in the form of the augmented decision tree model described above (Cockett 1983a. 1983b). These decision expressions can be used to represent large decision making processes

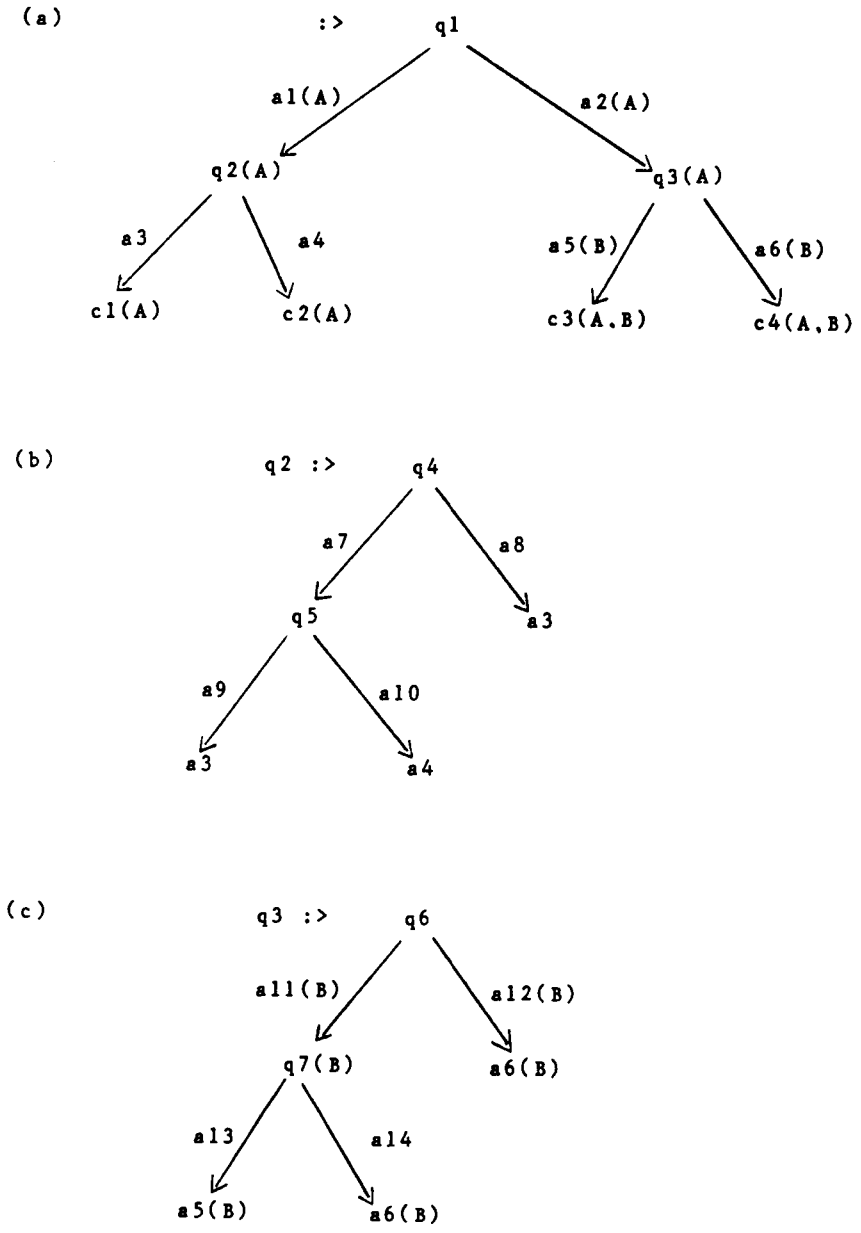

Figure 1-4: Augmented Decision Tree Model Used by DECIDE. (a), (b), and (c) are knowledge clusters. (a) is the initial knowledge cluster. (b) and (c) have governing questions q2 and q3 in (a), respectively.

with a relatively small amount of code. The following section is a description of this syntax, which the author and Dr. Cockett call the DECIDE language.

The DECIDE Language

The DECIDE language is based on a series of statements which manipulate decision expressions. Decision expressions are an algebraic notation for describing decision trees. The statements facilitate the encoding of knowledge represented with decision expressions.

The Syntax of Decision Expressions

The following definition of decision expressions formalizes their syntax[1].

Given a stock of local variables,

{x1,x2,...},

a stock of conclusions,

{c1.c2....},

a stock of question heads,

{q1,q2,...},

each of which has an answer set,

ans(qi) = {ai1,ai2,...,ain},

---

[1] A general syntax for decision expressions is found in (Cockett 1983b). The syntax and definitions described here pertain to the implementation of DECIDE version 3 and have been formalized in personal correspondence between Dr. Cockett and the author.

associated with it, then decision expressions can be defined recursively as follows:

(a) a local variable, xi, is a decision expression
(b) if ci is a conclusion then <:ci is a decision expression
(c) if e1,e2,...,en are decision expressions and q is a decision with answer set {a1,...,an} then

q:[e1<<a1,...,en<<an]

is a decision expression, where q is called a question head. The infix binary operator ":" associates the question head to a list of one or more branches. A branch is denoted ei<<ai where ai is an answer to the head question and ei is called an outcome.

The Syntax of Decision Statements

The DECIDE program processes knowledge in the form of decision statements. These statements perform three basic functions: decision tree root definition, local variable assignment, and decision substitution. The syntax of these statements describes the DECIDE language. With these statements large decision processes can be methodically codified with relative ease.

Root definition statement. Every decision process has a unique root R which is a decision expression. DECIDE uses the unary postfix operator ":>" to designate the root of a decision process:

:> R.

This is called a root definition statement.

Assignment statement. Local variables can be used to represent decision subtrees. In the DECIDE language a local variable x1 can be assigned to a decision expression of the form q:[e1<<a1,...,en<<an] by use of the infix binary operator ":=".

```
x1 := q:[e1<<a1,...,en<<an].
```

This operator is called the assignment operator and the type of statement shown above is called an assignment statement.

Decision substitution statement. The binary infix operator ":>" is used to denote decision substitution. Its left hand operand is a question head of a decision expression and its right hand operand is a decision expression representing the knowledge cluster (decision tree) that answers the governing question.

The use of decision substitution allows large decisions to be structured hierarchically. The more general questions are placed at the top of this hierarchy. Each of these can be replaced by a decision tree which has questions which are more specific, and these questions can in turn be replaced by decision trees, and so on. In an ideal implementation, the bottom of the hierarchy would have only trivial (easy to answer) questions. An example of a decision process represented as a set of DECIDE language statements is shown

in Figure 1-5. The decision statements in this example encode the decision trees shown in Figure 1-4.

Decision Substitution

A decision substitution for question head Q consists of a collection of decision statements such that

(a) there is a unique substitution statement, Q :> E

(b) for each local variable, C, occuring in the decision statements there is a unique assignment statement, C := E1

(c) there are no cyclic assignments

(d) every reachable conclusion in the statements is an answer of the question head Q

The collection of expressions associated with Q by the decision substitution statement are called a knowledge cluster. The question head Q is called the governing question of that knowledge cluster.

The Initial Decision Substitution

Every knowledge cluster in a DECIDE system knowledge base has a governing question which defines the domain of the knowledge which is contained in that cluster. At the very top of the hierarchy there is an implied governing question whose domain is all knowledge in the system. This questions might be stated as "What conclusion can the DECIDE system find for you?" Its answer set includes any conclusion that can be reached by the decision substitution for that

```
(a)     :> q1:[x2(A)<<a1(A),x3(A)<<a2(A)].
        x2(A) := q2(A):[<:c1(A)<<a3,<:c2(A)<<a4].
        x3(A) := q3(A):[<:c3(A,B)<<a5(B),<:c4(A,B)<<a6(B)].

(b)     q2(A) :> q4:[x5<<a7,<:a3<<a8].
        x5 := q5:[<:a3<<a9,<:a4<<a10].

(c)     q3(A) :> q6:[x7(B)<<a11(B),<:a6(B)<<a12(B)].
        x7(B) := q7(B):[<:a5(B)<<a13,<:a6(B)<<a14].
```

Figure 1-5: Decision expressions. The decision expressions above encode the augmented decision tree in Figure 1-4. (a), (b), and (c) are all knowledge clusters. (a) is the initial knowledge cluster. Questions q2 and q3 are governing questions for knowledge clusters (b) and (c), respectively.

governing question. This first substitution is called the initial decision substitution.

The initial decision substitution consists of a collection of decision statements such that

(a) there is a unique root definition statement (:> R), whose right hand operand (R) is called the initial knowledge cluster

(b) for each local variable, V, occuring in the decision statements there is a unique assignment statement, V := E1

(c) there are no cyclic assignments

(d) every reachable conclusion in the statements is an acceptable conclusion to the DECIDE system

The DECIDE Knowledge Base

A knowledge base for the DECIDE system consists of a collection of knowledge clusters. One and only one of these must be an initial knowledge cluster. All knowledge clusters in a knowledge base have governing questions in that knowledge base.

The three DECIDE statements (root definition, assignment, and substitution) provide all the expressive power needed to represent decision processes. Root definition defines the first decision. The use of local variables and assignment statements provide a means of breaking decision trees into their component parts, which simplifies the human encoding of these trees. Decision substitution statements provide a means of hierarchically

structuring a knowledge base, which improves human understanding of the knowledge and its organization.

In the next chapter the concept of "knowledge clustering" is discussed as a means of organizing a knowledge base. Knowledge clustering in the DECIDE system is based on the hierarchical structure of the decision tree model above. It is shown how this hierarchical organization is fundamental to DECIDE's file handling and explanation systems.

CHAPTER 2

KNOWLEDGE CLUSTERING IN DECIDE

Moret notes that a particularly important tool in the analysis of problems is decomposition, which tries to simplify a problem by breaking it into parts which are easier to solve. He further describes a means of hierarchically structuring decision tree by using node decomposition. Node decomposition allows a K-ary node (one with K arrows leaving it) to be replaced by a decision tree which has at least one leaf label matching each of the arrow labels of the node it replaces. This creates a functional hierarchy analogous to the use of subroutines in software (Moret).

In the DECIDE system decision substitution performs this same decomposition. Decision substitution allows a question to be replaced by a decision tree that can answer that question. Each decision tree constitutes a knowledge cluster and is the basic module for storing code in the DECIDE system's knowledge base. The concept of knowledge clustering, and its importance to both file handling and explanation generation, is discussed in this chapter.

Knowledge Clustering

Clustering is a technique of grouping related items into packets. This concept is certainly universal to any

organization scheme. Secretaries cluster files into folders to organize them. A text book is a cluster of chapters on a subject. Structured programming advocates the clustering of code that performs a specific function into a subroutine.

Usually a clusters is labeled to indicate what is contained within it. File folders have printed labels to describe their contents. Text books and their chapters have titles to describe the subject matter they contained. And each subroutine of a program has a name and parameters to describe what it does.

A knowledge cluster is a grouping of related facts and rules which are focused on a specific knowledge domain. Knowledge clustering is a technique for segmenting a knowledge base into related but independent knowledge clusters.

Structured Programming Analogy

Knowledge. like an algorithm, is procedural and as algorithms can be programmed, so can knowledge. Likewise, the same software engineering techniques that apply to computer programming (i.e. top-down design, modularity, abstraction) apply to knowledge programming.

Early pioneers of structured programming brought attention to the notion of abstraction. Wegner defines abstraction as follows:

> An abstraction is a characterization of the object by a subset of its attributes . . . If the attribute subset captures the "essential" attributes of the object, then the user need not be concerned with the object itself but only with the abstract attributes (Belady).

Dijkstra adds that

> there is an abstraction involved in naming an operation and using it on account of what it does while completely disregarding how it works (Belady).

It was mentioned earlier that clusters are ofter labeled to indicate their contents. An abstraction is used when subroutines are named (labeled) according to what they do. The analogy between subroutines and knowledge clusters would suggest that an abstraction can be created by the labeling of these structures as well. In fact this is precisely what the DECIDE system does -- it labels each knowledge clusters with a governing questions.

The Governing Question

DECIDE's knowledge base is a collection of interrelated knowledge clusters. Each of these knowledge clusters is a decision tree which is labeled by a governing question. A governing question defines the domain of the knowledge that its cluster contains and thus defines the context of all decisions in that cluster. The answer set of a governing question is the conclusion set of the knowledge cluster (decision tree) it labels. Each question in a knowledge cluster can itself be a governing question for another knowledge clusters.

DECIDE uses decision substitutions, as described in Chapter 1, to replace a governing question in a decision with a knowledge cluster which can be used to answer it. An entire expert system can thus be defined in the context of a single governing question. A knowledge cluster labeled by this governing question would have to contain a set of decisions which can produce the desired conclusion. Each of these questions could in turn be a governing question for other knowledge clusters.

This system allows knowledge to be organized in a top-down manner in much the same way as code is organized in a structured program. For example, the knowledge contained in the TAXON knowledge base (see introduction) can be encompassed by a single knowledge cluster which has the governing question "What species of Tennessee tree are you trying to identify?" The decision substitution for this question embodies the knowledge cluster. This substitution includes three decisions, shown below. The terms enclosed in angle brackets are context and answer variables instantiated from the answer set of the previous questions.

(1) "What group of genera does the tree belong to?" --> <GROUP>

(2) "What genus in group <GROUP> does the tree belong to?" --> <GENUS>

(3) "What species of genus <GENUS> does the tree belong to?" --> <SPECIES>

Each of these questions (1), (2) and (3) can itself be a governing question and a knowledge cluster (decision substitution) can exist for each of them.

Note that questions (2) and (3) are context dependent. That is, their meaning is determined by the value given to a context variable (Chapter 1) in the previous decision. In this example. if there are five groups and each group contains five genera, then 1 x 5 x 5, or 25, different questions can be generated from the set of three, each governing its own knowledge cluster.

An Intrinsic Explanation System

Knowledge clustering in DECIDE provides a very good model for explanation generation. The context of a question being asked is defined by the governing question of the knowledge cluster it belongs to. If that governing question is itself a question in another knowledge cluster, then an even broader context of the current question is known.

For example. in TAXON the question "Is the tree deciduous?" is in the knowledge cluster whose governing question is "what group of genera does the tree belong to?" This governing question is in turn a question in the knowledge cluster whose governing question is "What species of Tennessee tree are you trying to identify?" If a user is asked "Is the tree deciduous?" and the user replies "Why is this question important?", then the system can respond with

an explanation such as follows:

> It is important to know if the tree is deciduous in order to determine what group of genera it belongs to. It is important to know what group of genera it belongs to in order to determine what species of Tennessee tree you are trying to identify.

Such an explanation provides the user with a clear understanding of the immediate and long term goals of the system.

File Handling and Knowledge Clusters

Dr. Cockett designed the file handling scheme for the DECIDE system (Cockett 1983b). Each knowledge cluster in the knowledge base is stored in a separate file. Its governing question is a question in another knowledge cluster (file).

Decision substitution is used as a model for the file management of a DECIDE knowledge base. A knowledge cluster does not need to be considered if the governing question for that cluster can be answered directly. If the cluster is needed, then the file which contains that cluster can be referenced by the governing question which labels it. This is done by using decision substitution statements. For example.

```
x1 := q1:[x2<<a1....].
q1 :> file(q1.dat).
```

The knowledge cluster that contains the solution to question q1 is found in the file, q1.dat. In this file another

decision substitution statement, whose head question is also q1, defines the substitution for the governing question.

```
<q1.dat>
q1 :> q2:[<:a1<<b1,...].
.
.
.
```

There is no reason to keep all of the knowledge clusters that comprise a knowledge base in working memory. Only the clusters on which focus is directed need be considered. Decision substitution is used as a cue for the file management of a DECIDE knowledge base. The DECIDE language syntax (Chapter 1) allows for a knowledge cluster to be loaded into memory when a decision substitution is made. When an answer is returned to the knowledge cluster's governing question that cluster is no longer needed and may be removed from working memory. The only knowledge clusters that reside in working memory are those that have unanswered governing questions.

Hierarchical structure is very important to any knowledge representation scheme. The traditional decision tree model lacks this structure and is thus a poor model for knowledge representation. The augmented decision tree model described in Chapter 1, in conjunction with the concept of knowledge clustering, provides this hierarchical structure to DECIDE's knowledge. Chapter 3 describes the design of

CHAPTER 3

THE DECIDE PROGRAM

The Overall Design of the DECIDE System

Throughout its development the DECIDE system's general scheme for expert system implementation has remained basically the same. This model is comprised of four components: a knowledge base, a decision expression interpreter, a user interface, and an acquired knowledge store. A schematic diagram of this design is shown in Figure 3-1.

The Knowledge Base

DECIDE's knowledge base is a collection of knowledge clusters encoded as decision statements (as discussed in Chapter 1). It supplies knowledge in the form of these statements to the inference engine. This knowledge base in DECIDE version 3 is not dynamic, that is to say, knowledge acquired during a session with DECIDE will not change the contents of the knowledge base for future sessions. The system thus cannot learn for experience.

The Acquired Knowledge Store

The acquired knowledge store is collected from the user interaction and is saved in a separate data base. This

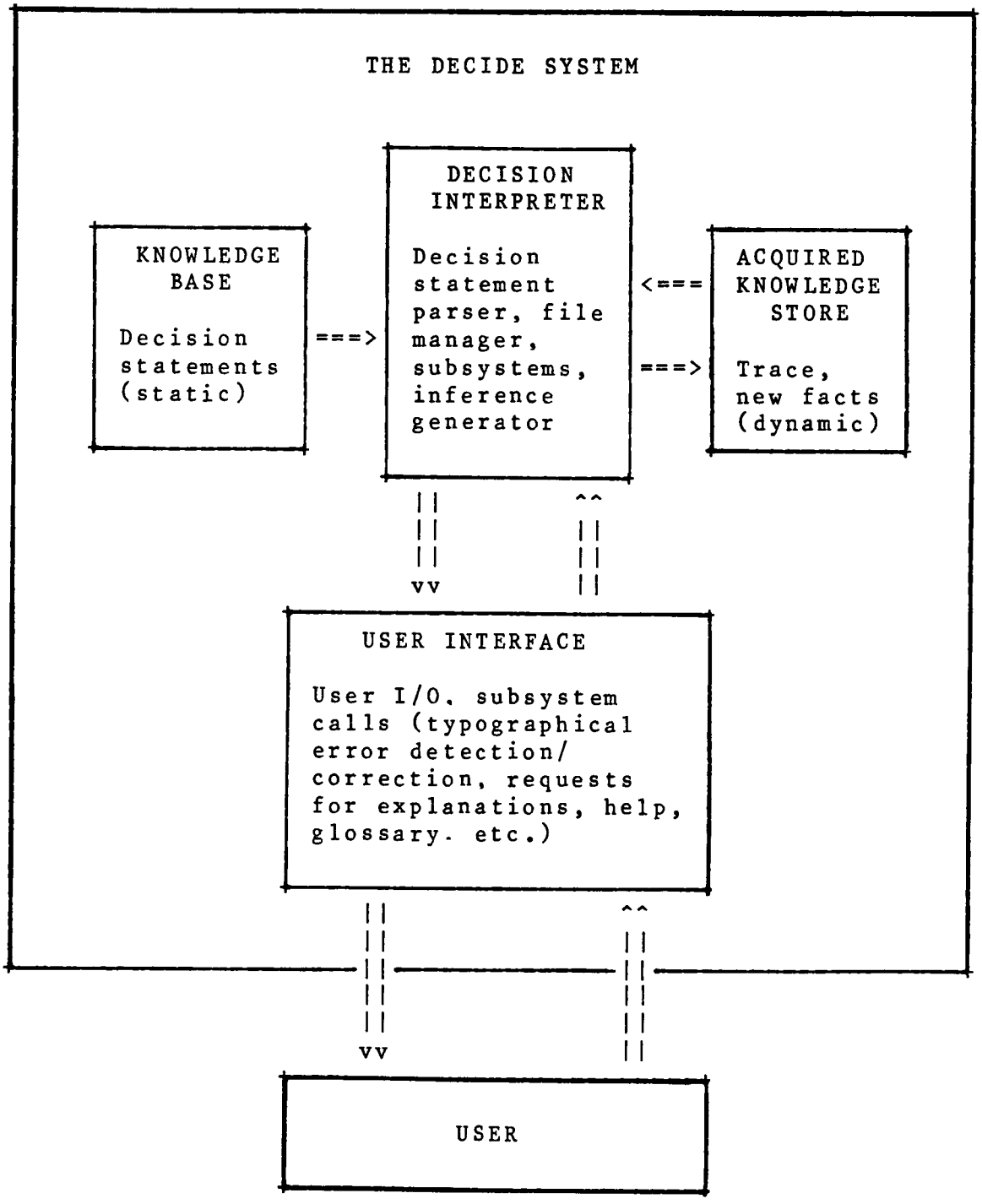

Figure 3-1: The DECIDE System. Arrows indicate the flow of information within the system.

knowledge includes a history or trace of all previous interactions with the system. The trace is a stack of structures each containing a decision expression and the answer assigned to its head question. These question-answer pairs are actually facts which are available to the inference engine for later decisions. The trace also describes the path taken through the decision tree. This is important to the implementation of backtracking and explanation generating systems.

The Decision Expression Interpreter

The decision expression interpreter and the user interface comprise the body of the DECIDE program. The decision expression interpreter performs several duties. It acts as a decision statement parser which breaks these statements down into their components: expressions and operators. It is responsible for performing the duties of local variable assignment and decision substitution as indicated by the operators. It procures answers to questions in order to traverse the decision tree and publishes conclusions when they are reached.

The decision interpreter also is responsible for knowledge management and file handling. Knowledge management in this case refers to the procurement and use of information that may be useful to the problem at hand. Information is gathered from decisions in the knowledge

base. When a question in a knowledge cluster is asked, the decision interpreter must find an answer for it in order to proceed to the next question or conclusion in the decision tree. The interpreter has three potential sources of an answer: internal knowledge, the user, and decision substitution. If it cannot produce an answer from information in its acquired knowledge store then it must ask the user for the answer. If the user cannot answer the question then the system searches for a substitution for that decision. If one does not exist then the system cannot proceed to a conclusion. This implies that a well designed knowledge base contains decision substitutions for all questions that a user might have difficulty answering.

File handling is closely tied to the knowledge clustering scheme of knowledge organization. Ideally each file in the physical knowledge base contains the decisions that make up a single cluster. The knowledge cluster interface is thus the file interface as well.

The decision interpreter contains several routines and subsystems that perform various tasks to enhance user interactions. Some of these are executed automatically. Automatic tasks include a token (answer) completion utility, a typographical error detection and correction filter, a conclusion publishing routine and a text formatter. Other tasks can be invoked by a request from the user. These include an answer set publisher, a system to explain the

context of a question, a conclusion verifier, a help routine, a glossary facility, a backtracking facility, and a system for sending messages from the user to the system manager.

The User Interface

The user interface provides a communication link between the user and the decision interpreter. It allows the user to participate in a decision process by answering questions. Access to the various subsystems in the decision expression interpreter is also provided. The user can invoke such a subsystem by giving a system command.

One of the major problems with large software systems is their user interfaces often lack uniformity and consistency (Smith). DECIDE's user interface behaves very uniformly because all interactions with the system, including those with subsystems, are managed by the decision interpreter. This is possible because all subsystems are written as decisions in the DECIDE language. Each subsystem is itself a knowledge cluster. This also allows subsystems to be invoked during interactions with other subsystems.

Text Generation and Answer Recognition

Internally the DECIDE system represents questions, answers, and conclusions with symbols (i.e. q1, a1, c1) which have little if any meaning to the user. The program

must associate text with these symbols so that a meaningful interaction can be generated. Also, text provided by the user must be matched to answers of the current question.

Version 3 of DECIDE, which is written in Prolog, uses Prolog grammar rules to generate text and to define answer types. (For a good description of Prolog grammar rules see (Clocksin)). There are five types of text generation used in DECIDE. Ask text associates text with questions. Explain text is used to publish explanations (see WHY command below). Enter text is used to allows text to be published when a decision substitution is made (see decision substitution request below). Exit text is used to publish conclusions or subconclusions when the system returns form a decision substitution call (see decision substitution request below). Glossary text is used to publish definitions or explanations of terms found in the systems glossary (see GLOSSARY subsystem below).

Answer types, as discussed in chapter 1, allow any member of a defined set of answer to be associated with a single outcome. It is thus important to provide a means of establishing membership to a given set of answers. Grammar rules prove to be a just such a means. Answer types in the DECIDE system are defined with grammar rules which can recognize any member of the set of answers represented by that type (see ANSWERS command below).

The remainder of this chapter is devoted to describing the various subsystems that the decision interpreter provides. First the automatically invoked systems are discussed. Then the user requested tasks are described.

The Automatic Subsystems of DECIDE

The CONFIRM and EXACT subsystems. The CONFIRM subsystem is responsible for overseeing all interactions that occur during the running of the program. Each time a question is asked an answer must be provided to perpetuate the interaction. This answer may be provided by an internal process, from a query to the user, or by a decision substitution. If the reply is from an internal process or a returned conclusion from a substitution then the system can assume that it matches exactly one answer from the current question's answer set. If the source of the response is the user, however, this assumption cannot be made. In this case CONFIRM has the task of deciding which answer the user intended to match. After the match is made, CONFIRM maps the reply to the next question in the interaction.

The first thing CONFIRM does when it receives an reply form the user is determine if the user's reply in an answer to the current question or a system command. All system commands begin with the <ESC> character and are thus easily recognized. When a response is given by the user, its first character is examined. If this character is the <ESC>

character, then the system takes the rest of the string as a possible system command. If the first character is not the <ESC> character, then the reply is treated as a possible answer to the current question.

If a user's reply to a question does not match exactly with an answer to that question or one of the system commands then another subsystem, EXACT, is called to determine whether or not the reply can be matched.

The EXACT subsystem is invoked by the CONFIRM subsystem when a user's reply to a question does not match exactly an answer of that question. There are four conclusions that this subsystem can reach. First, the user's reply is a proper prefix to only one of the answers. Second, the user's reply is a single typographic error of a prefix to only one answer which can be confirmed by the user as matching the answer exactly. Third, the user's reply is an ambiguous prefix. Fourth, the user's reply is not recognized as an answer to the current question.

The first thing that EXACT does is to determine if the reply is a proper prefix to only one answer of the current question. If it is, the matched answer is returned to CONFIRM and entire answer is published in parentheses to broadcast this conclusion to the user. However, if the reply is a prefix of more than one of the answers, then the system will conclude that the reply is ambiguous.

For example, suppose "big", "bigger" and "biggest" are the only answers for the current question. If the user answers "big", CONFIRM recognizes the exact match without considering that "big" is a prefix of the other two answers. IF the user's reply is "bigges" then EXACT is called to analyze the reply. EXACT returns the conclusion that "biggest" was the intended answer because "bigges" is uniquely its prefix. This conclusion is then verified to the users by publishing "(biggest)". The interaction will then proceed as if the user had typed the entire string "biggest". If the user's reply is "bigge", then EXACT concludes that the user's reply is ambiguous, since it is a prefix of both "bigger" and "biggest".

A second conclusion that EXACT may reach is that the user has mistyped a reply to a question in such a way that the intended answer may be guessed. Four types of typographical errors are considered:

1) single letter substitution - for example, for the answer "yes" the reply "xes" is a typo where the letter "x" has been typed instead of "y".
2) single letter deletion - for example, for the answer "yes" the reply "ys" is a typo where the letter "e" has been omitted.

3) single letter insertion - for example, for the answer "yes" the reply "yexs" is a typo where the letter "x" has been inserted erroneously into reply.

4) letter pair reversal - for example, for the answer "yes" the reply "yse" is a typo where the letter pair "es" have been reversed.

Furthermore, a user may mistype a prefix to an answer. For example, the reply "yx" matches the answer "yes". Here "yx" matches "ye" by letter substitution and "ye" is a prefix of "yes". Once such a match is made the system asks the user if the guessed answer is indeed what was intended. If the user answers "yes" to this question, the conclusion returned to CONFIRM is that the guessed answer is an exact match to the current question. If the user responds that the system's guess is wrong, then the conclusion returned to CONFIRM is that the user's reply is unrecognized.

If a reply from the user is a typo which matches more than one answer of the current question, then EXACT concludes that the reply is unrecognized. That is, when an ambiguity exists EXACT does not try to guess which one is intended by the user. For example, if the answers to a question are "yes" and "no" and the user replies, "ne", then the systems finds that both answers can be matched using the typographic checking described above. "ne" matches "no"

using single letter substitution ("e" for "o"), but it also matches "ye" using single letter substitution ("y" for "n"). "ye" is a prefix of yes so an ambiguity exists.

The last case to consider is that the user's reply to a question cannot be handled by the prefix and typographical matching procedures above. For example, if the answers to the current question are "yes" and "no" and the user replies "elephant", the EXACT system concludes that the user's response is unrecognized.

The CONCLUDE subsystem and the HOW command. Before a conclusion is accepted by the DECIDE system it is passed to a subsystem called CONCLUDE. This subsystem publishes the conclusion for the user to examine and pauses for the user to indicate whether or not the conclusion is acceptable. If the user types a carriage return (hereafter denoted <CR>), then the conclusion is accepted and passed back to the governing question as one of its answers.

DECIDE pauses at this point in the interaction for two reasons. First, it allows the user an opportunity to ask how the conclusion was reached. This is done by the user typing "HOW", which signals CONCLUDE to generate an explanation. Second, a pause allows the user to use the backtrack command if the conclusion is unsatisfactory. This is important because knowledge cluster boundaries are also file boundaries. Upon accepting a conclusion from a

knowledge cluster that cluster is evacuated from working memory. To backtrack after the evacuation would be more costly.

This explanation that is generated by CONCLUDE summarizes the way that questions were answered in the current knowledge cluster and then restates the conclusion reached form the interaction. The example below demonstrates how this system works.

```
What type of animal is it?
|: <ESC>?

We must now determine what type of animal is being
considered.
Does the female of the species give milk?
|: YES

Then the animal is a mammal.
|: <CR>

The conclusion, the animal is a mammal, has been
accepted.

What type of mammal is it?
|:
```

Here, the user did not know which type of animal was being considered and answered "I don't know." By answering "yes" to the question, "Does the female of the species give milk?", the conclusion was made that the animal is a mammal. Before accepting this conclusion, the user is given a chance to accept or reject the outcome of the interaction. By typing a <CR> the user indicated to the CONCLUDE system that the conclusion that system provided was satisfactory. The

system then continued as if the user had answered "mammal" to the original question. If the user had responded "how" instead then the following explanation would have been published.

> If the answer to the question "does the female of the species give milk?" is yes, then the answer to the question "what type of animal is it?" is mammal.

The User Requested Subsystems of DECIDE

Each subsystem that can be invoked by a user's request has a system command associated with. At any time in an interaction with DECIDE the user may invoke one of these commands. The answering of the current question is postponed until the command is finished executing. A list of the system commands is shown in Table 3-1. When one of these commands is issued to DECIDE, it must be prefixed with the <ESC> character to distinguish it from answers to the current question.

The decision substitution request. When the answer to a question is not known, then the user can respond with the command "<ESC>?". This is the user's way of saying "I don't know". It is now up to the system to decide how best to help the user answer this question. The system first looks in the knowledge base to see if a decision substitution statement for the current question can be made. If one can, then the appropriate knowledge cluster is fetched to decide

Table 3-1. The System Commands of DECIDE. Users of the DECIDE system may issue one of the system commands below at any time during a session. Each command must be preceded with the <ESC> character to distinguish it from answers to the current question.

Command	Function
?	Request for decision substitution
ANSWERS	Print answers to current question
BACKTRACK	Backtrack to preceding question
COMMENT	Write a comment to the system log file
GLOSSARY	Access glossary of terms
HELP	Print list of system commands
QUIT	Abort execution of session with DECIDE
RESTART	Restart current decision from root
WHY	Publish context of current question

the question. When a conclusion is encountered in the knowledge cluster, it is returned as an answer to its governing question and the interaction continues as if that answer had been supplied directly by the user. The example given in the discussion of the CONCLUDE subsystem demonstrates such a call.

The ANSWERS command. Each question that is asked has a set of answers which is associated with it. This set includes answers encoded in the decision expression branches, synonyms of these answers, and answers that can be generated from associated grammar rules in the knowledge base. The example shown below represents a question q and its answer set.

```
q1 := q1:[q2<<non_neg_int,
          q3<<one,
          q4<<many].

non_neg_int --> ['<',any,'non-negative', integer,'>'].
non_neg_int --> [X], {integer(X), X >= 0}.

synonym(lots,many).
synonym(several,many).
```

The answer set for question q1 includes the words "one", "many", "several" and "lots" and the set of all non-negative integers which can be processed by the particular Prolog interpreter.

The ANSWERS subsystem publishes the answers to the current question for the user to examine. It is invoked by

the command "<ESC>answers". If the user inquires about the answers for the question above (q1), then the system would respond as follows.

The possible answers are:

< any non-negative integer >
one
many

Note that synonyms are not published. The reason for this is that matching replies with synonyms requires more computing overhead than matching pointers. The system thus trains the user to use "best" answers. Also note that the pointer "integer" is not given as an answer. This is because the pointer is not intended to be an answer but instead it represents a set of answers which are described in the grammar rules. One grammar rule for "integer" generates the string "< any non-negative integer >". This is used only for text generation for the ANSWERS subsystem. The answers it refers to are the integers {1,2,3,...} as described by the other grammar rule.

The BACKTRACK command. The backtracking facility is invoked with the command "<ESC>BACKTRACK". It allows a user to backtrack to the previous question. Backtracking erases from DECIDE's trace all facts up to and including the last user answered question. If the user backtracks into a

previously evacuated knowledge cluster that cluster is restored from disk.

The COMMENT command. At times during a session with an expert system a user may wish to make a comment to the system manager. The COMMENT subsystem was designed to permit this. At any point in the interaction the command "<ESC>comment" can be used to invoke the COMMENT subsystem. The subsystem asks the question "What comment would you like to make to the system manager?" and then reads the user's comment. A record is then built containing three items: the question that was asked when the COMMENT subsystem was invoked, the file name of the knowledge cluster containing that question, and the user's comment text. This record is then written to a system log file which can be examined at a later time by the system manager. The following interaction demonstrates the use of the comment command.

```
What tipe of animal is it?
|:<ESC>COMMENT
What comment would you like to make to the system
manager?
> You have misspelled the word "type" as "tipe" in
> this question! <

What tipe of animal is it?
|:
```

The GLOSSARY command. The GLOSSARY subsystem allows the user to access the definitions for any glossary term in the current question. If there is only one glossary term in

the question, its definition is published when the glossary subsystem is invoked. If there is more than one term in the question, then the user is asked which of these term is to be defined. Glossary terms are flagged in the ask_text for questions as the argument of a structure, g/1. For example, consider the ask_text for the question q1, below.

```
ask_text(q1) --> [to,which,g(genus),does,the,
                  plant,belong,'?'].
```

The term "genus" is flagged as a glossary term. Associated with each glossary term is a definition, which is stored in the knowledge base as shown below.

```
gloss_text(genus) --> [genus,'-','pl.',genera,':',
                       a,group,of,closely,related,
                       species].
```

When the ask_text for a question is displayed on the user's terminal DECIDE's text formatter displays the term in bold or highlighted script. This makes the glossary terms easy to identify.

The HELP command. The help subsystem can be invoked by using the command "<ESC>HELP". The list of the system commands is published, each of which is flagged as a glossary term. Thus if the user wishes to know more about a particular command the glossary subsystem can be used to retrieve this information.

The WHY command. The question context explanation system of DECIDE is invoked by the system command <ESC>WHY. This subsystem is designed to explain to the user the relevance of the current question being asked. The default response is generated by appending the governing question for the current knowledge cluster to the string "Because answering this question will help you resolve:" This is demonstrated in the following example.

```
What type of mammal is it?
|: <ESC>?

Going into detail to determine what type of mammal is it.
Does it eat meat?
|: <ESC>WHY

Because answering this question will help you resolve:
what type of mammal is it.

Does it eat meat?
|:
```

Each question for which a detail subsystem exists can have explain_text associated with it. This text is used to replace the default explanation text so that a more natural and perhaps more informative explanation can be presented to the user. Suppose in the example above that q1 is the label for the question "What type of mammal is it?" In the knowledge base there is the predicate:

```
explain_text(q1) --> [answering,this,question,will,
                      help,us,determine,which,type,
                      of,mammal,you,are,trying,to,
                      identify,'.'].
```

The resulting interaction would then look like this:

```
What type of mammal is it?
|: <ESC>?

Going into detail to determine what type of mammal is it.
Does it eat meat?
|: <ESC>WHY

Answering this question will help us determine which type of
mammal you are trying to identify.

Does it eat meat?
|:
```

QUIT and RESTART commands. The quit command is executed by typing <ESC>QUIT. This causes the termination of the current session with DECIDE and returns the user back to the operating system's monitor. The restart command is executed by typing "<ESC>RESTART" and is used to start a new session with DECIDE. It erases the entire acquired knowledge store before beginning the decision making process at the root of the decision tree.

Current Work on the DECIDE Program

The description of the DECIDE system above represents the state of the program in the fall of 1984 when the author left the project. Since that time. Dr. Cockett has continued work on the system under a subcontract from Martin Marietta Energy Systems, Inc. at the University of Tennessee. His work involves the application of the DECIDE program to implement an expert system as a front end to a

The resulting interaction would then look like this:

```
What type of mammal is it?
|: <ESC>?

Going into detail to determine what type of mammal
is it.  Does it eat meat?
|: <ESC>WHY

Answering this question will help us determine which
type of mammal you are trying to identify.

Does it eat meat?
|:
```

The QUIT and RESTART commands. The quit command is executed by typing <ESC>QUIT. This causes the termination of the current session with DECIDE and returns the user back to the operating system's monitor. The restart command is executed by typing "<ESC>RESTART" and is used to start a new session with DECIDE. It erases the entire acquired knowledge store before beginning the decision making process at the root of the decision tree.

Current Work on the DECIDE Program

The description of the DECIDE system above represents the state of the program in the fall of 1984 when the author left the project. Since that time, Dr. Cockett has continued work on the system under a subcontract from Martin Marietta Energy Systems, Inc. at the University of Tennessee. His work involves the application of the DECIDE program to implement an expert system as a front end to a

control systems design and simulation program called CASCADE. This work is described in two publications (Birdwell 1985a, 1985b), which include example interactions with the system.

CHAPTER 4

CONCLUSION

The implementation of DECIDE as a decision expression interpreter was a success. An augmented decision tree model for representing large decision processes was developed and used in the development of Dr. Cockett's decision expression syntax. The DECIDE language proved to be a very powerful tool for expressing large and complex decisions. The DECIDE decision interpreter provided a friendly interface between the knowledge base and the user. A technique for organizing decision trees into hierarchical identities provided a well organized structure for knowledge and allowed DECIDE to focus attention on specific problems in the knowledge domain. This in turn provided both an efficient file handling scheme and a good explanation system for explaining questions and conclusions that had been lacking in traditional decision models.

Knowledge bases implemented under DECIDE are structured by the use of knowledge clustering and decision substitution. These two techniques provide a strong analogy to the methodologies used in structured programming and the same benefits are shared in both systems.

A limitation exists on the type of knowledge the DECIDE system can manage. The knowledge must be structured hierarchically. The system thus lends itself to

applications such a taxonomy, where humans have already organized the knowledge. TAXON is just such an example. This is not to say that other types of knowledge cannot also be structured in this way. However there is a considerable amount of effort involved in formalizing a knowledge domain and structuring it hierarchically. However, once this is done the encoding of the knowledge in the DECIDE language should not be difficult.

DECIDE is continually being improved. Dr. Cockett is currently implementing an natural language text generator to the user interface. This work has already lead to the system's ability to publish well composed conclusions and explanation.

There are several ways that DECIDE can be improved. Support software for writing knowledge bases quickly and with a minimum of errors is needed badly. Currently, efficient knowledge programming is a result of very much practice. This process needs to be facilitated. Improving the capabilities of the natural language interface to allow English to symbolic translation would also be desirable. At present the DECIDE system has no ability to learn beyond the bounds of a single session. Research on learning algorithms for decision tree methodologies is needed to determine if such algorithms are practical.

Finally, the author regrets that there was not enough time in the duration of this thesis to implement the TAXON

knowledge base under version 3 of DECIDE. It would interesting to compare the performances of version 2 and version 3 of DECIDE using the same knowledge base.

BIBLIOGRAPHY

BIBLIOGRAPHY

Aho, A. V., J. D. Ullman (1972). The Theory of Parsing, Translation, and Compiling Volume I: Parsing. Prentice-Hall Inc., Englewood Cliffs, New Jersey.

Aikins, J. S. (1983). Prototypical knowledge for expert systems. Artificial Intelligence, . 20, pp. 163-210.

Akers, S. B. (1978). Binary decision diagrams; IEEE Trans. on Computers, 27(6).

Barr. A., E. A. Feigenbaum (ed.) (1981). The Handbook of Artificial Intelligence, vol. I; William Kaufman, Inc.

Belady, L. A., B. Leavenworth (1980). Program Modifiability in Software Engineering (Freeman); Academic Press.

Birdwell, J. D., M. Athans, A. A. Bly, J. R. B. Cockett, L. Hatfield, M. T. Heath, R. W. Heller, A. J. Laub, R. W. Rochelle, J. P. Stovall, B. Cabral, G. Chen, K. Fung, S. Heard, C. McCullough, A. Mak (1985a). CASCADE: experiments in the development of knowledge-based computer-aided systems and control analysis and design environments. (abstract) IEEE Control Systems Society 2nd Symposium on Computer-Aided Control System Design (CACSD).

Birdwell, J. D., A. J. Laub, M. Athans, L. Hatfield (1985b). Expert systems techniques in a computer-based control control system analysis and design environment. Plenary paper. 3rd IFAC Symposium on Computer Aided Design and Engineering Systems, Lyngby, Denmark July 31 - August 1, 1985.

Buchanan, B. G., E. H. Shortliffe (1984). Rule Based Expert Systems; Addison-Wesley Publishing Company.

Clancey, W. J. (1983). The epistemology of a rule-based expert system -- a framework for explanation. Artificial Intelligence, 20, pp. 215-251.

Clocksin W., C. Mellish (1981). Programming in Prolog; Springer Verlag, New York.

Cockett, J. R. B.; (1983a. October). Rewriting Decision Expressions. (Publication No. CS-83-55). (Available from the Computer Science Department, University of Tennessee, Knoxville).

_____. (1983b, November). File Handling for Detail, Extent, and Sub-Tasks in the Implementing of Decision Processes. (Publication No. CS-83-56). (Available from the Computer Science Department, University of Tennessee, Knoxville).

_____. (1984, October). The Theory of Discrete Decision Processes. (Publication No. CS-84-58). (Available from the Computer Science Department, University of Tennessee, Knoxville).

Cohen, P. R., E. A. Feigenbaum (ed.) (1984). The Handbook of Artificial Intelligence; William Kaufman, Inc.; vol. III.

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

Davis, R. (1980a). Meta-Rules: Reasoning about control. Artificial Intelligence, 15, pp. 179-222.

_____. (1980b). Content references: Reasoning about rules. Artificial Intelligence, 15, pp. 223-239.

Drummond, H. (1984). Knowledge Engineering for SALES: A Limited Legal Expert System. (Unpublished Masters thesis, Department of Computer Science, University of Tennessee, Knoxville).

Feigenbaum, E. A. (1979). Themes and case histories of knowledge engineering in expert systems in The Micro-Electronic Age; Michie, D., ed., Edinburgh University Press.

Freeman, H., P. M. Lewis II (1980). Software Engineering; Academic Press.

Gill, Arthur (1976). Applied Algebra for the Computer Sciences; Prentice Hall, Inc., Englewood Cliffs, New Jersey.

Hayes-Roth, F., D. A. Waterman, D. B. Lenat (eds.) (1983). Building Expert Systems; Addison-Wesley Publishing Company, Inc., Reading, Massachusetts.

Hayes-Roth (1984). The knowledge-based expert system: A tutorial; Computer, 17(8), pp. 11-28; Sept. 1984.

McCalla, G., N. Cercone (1983). Knowledge-representation. Computer 16(10), pp. 12-18.

Michie, D. (ed.) (1979). Expert Systems in the Micro-Electronic Age; Edinburgh University Press.

Mittal S., B. Chandrasekaran, J. Sticklen (1984) Patrec: A knowledge-directed database for a diagnostic expert system; Computer 17(9), pp. 51-58.

Moret, B. M. E. (1982). Decision trees and diagrams; ACM Computing Surveys, 14(4).

Mylopolous, J., S. Shibahara, J. K. Tsotos (1983). Building knowledge-based systems: the PSN experiment, Computer 16(10), pp. 83-89.

Negoita, V. N. (1985). Expert Systems and Fuzzy Systems; Benjamin Cummings Publishing Co., Inc.

Shanks, R. E., A. J. Sharp (1963). Summer Key to Tennessee Trees; The University of Tennessee Press, Knoxville. Tennessee.

Smith, R. G., G. M. E. Lafue, E. Schoen, S. C. Vestal (1984). Declarative task description as a user-interface structuring mechanism. Computer 17(9), pp. 29-38.

Weiner, J. L. (1980). BLAH, a system which explains its reasoning. Artificial Intelligence, 15, pp. 19-48.

Weiss S. M., C. A. Kulikowski (1984). A Practical Guide to Designing Expert Systems; Rowman and Allanheld.

Winston, P. H. (1977). Artificial Intelligence; Addison-Wesley Publishing Company, Inc., Reading, Massachusetts.

Winston, P. H., K. A. Prendergast (1984). The AI Business; The MIT Press, Cambridge, Massachusetts.

Woods, W. A. (1983). What's important about knowledge representation?; Computer 16(10), pp. 22-27.

APPENDIXES

APPENDIX A

INTERACTIONS WITH TAXON UNDER VERSION 2 OF DECIDE

```
/* NOTE: The dollar sign "$" preceding system commands */
/* in the following interaction represents the <ESC>   */
/* character.  User input follows the prompt "|:".     */

/*                      SESSION A1                     */

-------------------------------------------------------------
-------------------------------------------------------------

To which of the following groups does the tree belong:
     A:  Trees with needle or scale-like leaves
     B:  Broad-leaf evergreens
     C:  Trees with thorns on their stems
     D:  Trees with two or more leaves at a node
     E:  Trees with alternate compound leaves
     F:  Trees with alternate simple leaves
     (Type '?' if you do not know.)
|: $ANS
This question may be answered with one of the following:
a
group a
b
group b
c
group c
d
group d
e
group e
f
group f

To which of the following groups does the tree belong:
     A:  Trees with needle or scale-like leaves
     B:  Broad-leaf evergreens
     C:  Trees with thorns on their stems
     D:  Trees with two or more leaves at a node
     E:  Trees with alternate compound leaves
     F:  Trees with alternate simple leaves
     (Type '?' if you do not know.)
|: a
Group A:  Trees with needle or scale-like leaves.
```

```
To which genus does this plant belong?
 (Type '?' if you don't know.)
|: pinus
genus Pinus
Leaves characteristically 5 in a bundle.
|: y
species P. strobus L.

-------------------------------------------------------------

/*                      SESSION A2                           */
/*                                                           */
/*   Another session with TAXON with more detailed           */
/*   interaction.                                            */

-------------------------------------------------------------
-------------------------------------------------------------

To which of the following groups does the tree belong:
    A:  Trees with needle or scale-like leaves
    B:  Broad-leaf evergreens
    C:  Trees with thorns on their stems
    D:  Trees with two or more leaves at a node
    E:  Trees with alternate compound leaves
    F:  Trees with alternate simple leaves
    (Type '?' if you do not know.)
|: $?
Deciduous tree (leaves of one year falling before the
 subsequent leaves expand)
|: yen
You mistyped your reply!
yes?
Please try again.
Deciduous tree (leaves of one year falling before the
 subsequent leaves expand)
|: yes
Thorn-bearing trees; branches and sometimes trunks bearing
 various kinds of thorns.
|: y
Group C - Trees with thorns on their stems
To which genus does this plant belong?
 (Type '?' if you don't know.)
|: $a
```

```
This question may be answered with one of the following:
robina
xanthoxylum
aralia
gleditsia
pyrus
crataegus
maclura
bumelia

To which genus does this plant belong?
 (Type '?' if you don't know.)
|: $?
Leaves simple.
|: y
Leaves entire.
|: y
Petrioles 2-5 cm. long; leaves acuminate, usually rounded
 at the base; the multiple fruit spherical, 10-15 cm. in
 diameter.
|: y
species Maclura pomifera (Raf.) Scheid. (Osage orange,
hedge-apple)

------------------------------------------------------------

/*                        SESSION A3                        */
/*                                                          */
/*  Another session with TAXON under DECIDE version 2.      */


------------------------------------------------------------
------------------------------------------------------------

To which of the following groups does the tree belong:
    A:  Trees with needle or scale-like leaves
    B:  Broad-leaf evergreens
    C:  Trees with thorns on their stems
    D:  Trees with two or more leaves at a node
    E:  Trees with alternate compound leaves
    F:  Trees with alternate simple leaves
    (Type '?' if you do not know.)
|: $?
Deciduous tree (leaves of one year falling before the
subsequent leaves expand)
|: n
Evergreen tree (leaves persisting into the second year or
 longer; leaves broad, hard,leathery. or needle-like or
 scale-like)
|: y
```

```
Trees with needle-like leaves or small scale-like leaves.
|: y
Group A - Trees with needle or scale-like leaves
To which genus does this plant belong?
 (Type '?' if you don't know.)
|: $?
Leaves needle-like, 2, 3, or 5 in a bundle with a sheath at
 the base, 5 to several cm. long.
|: y
genus Pinus
Leaves characteristically 5 in a bundle.
|: n
Leaves 2 or 3 in a bundle.
|: y
Leaves 2 in a bundle; or both twos and threes.
|: y
Cones usually asymmetrical, often more than 7 cm. in length,
 with very stout prickles.
|: yes
species P. pungens Lamb.

-------------------------------------------------------------

/*                      SESSION A4                          */
/*                                                          */
/*   A seesion with TAXON under DECIDE version 2            */
/*   This session reaches the same conclusion as            */
/*   SESSION A3. but in fewer interactions.                 */

-------------------------------------------------------------
-------------------------------------------------------------

To which of the following groups does the tree belong:
    A:  Trees with needle or scale-like leaves
    B:  Broad-leaf evergreens
    C:  Trees with thorns on their stems
    D:  Trees with two or more leaves at a node
    E:  Trees with alternate compound leaves
    F:  Trees with alternate simple leaves
    (Type '?' if you do not know.)
|: a
Group A:  Trees with needle or scale-like leaves.

To which genus does this plant belong?
 (Type '?' if you don't know.)
|: pinus
genus Pinus
Leaves characteristically 5 in a bundle.
|: n
Leaves 2 or 3 in a bundle.
|: y
```

Leaves 2 in a bundle; or both twos and threes.
|: y
Cones usually asymmetrical, often more than 7 cm. in length,
with very stout prickles.
|: y
species P. pungens Lamb.

APPENDIX B

AN EXAMPLE INTERACTION UNDER VERSION 3 OF DECIDE

```
/* NOTE: The dollar sign "$" preceding system commands */
/* in the following interaction represents the <ESC>   /
/* character.  User input follows the prompts "|:" and */
/* ">".  Input for the former is delimited by a <CR>;  */
/* input for the later is delimited by "<" followed    */
/* immediately by a <CR>.                              */

------------------------------------------------------------
------------------------------------------------------------

 Hello and welcome to version 3.0 of DECIDE.


 This program tries to answer the question "What do you want
 to know?".  If what you want to know is in the domain of
 one of the DECIDE expert systems then you should not have
 much trouble discovering it.

 Have fun and good luck.

                    Robin Cockett and Shelley Heard


 What is your terminal type?
|: vt100
 Which expert system would you like to use?
|: $a
 ( answers )
 The possible answers are:
      animals
      taxon
      test
 Which expert system would you like to use?
|: animals
 Going into detail to determine which animal is being
 identified.  First it is necessary to determine the broad
 classification of the animal.  Does the female of the
 species give milk?
|: y
 ( yes )
 The answer to the question:
        what type of animal is it?
 is mammal
```

```
|: how
 If the answer to the question:
       does the female of the species give milk?
 is yes
 then the answer to the question:
       what type of animal is it?
 is mammal.  It is now necessary to determine which mammal
 is being considered.  What type of mammal is it?
|: $?
 First the type of mammal being considered is determined.
 Does this animal eat meat?
|: yen
 Did you mean ' yes '?
|: y
 ( yes )
 The answer to the question:
       what type of mammal is it?
 is carnivore
|: how
 If the answer to the question:
       does this animal eat meat?
 is yes
 then the answer to the question:
       what type of mammal is it?
 is carnivore.
|: continue
 It remains to determine which carnivore is being
 considered.  Is it tawny in colour?
|: $help

   System Commands:
     answers
     comment
     glossary
     help
     quit
     system
     unknown or ?
     vms
     why
   Note: all system commands are prefixed by the <ESC>
   character.

   To continue type a carriage return.
|: $g
 ( glossary )
 Which term would you like defined?
```

```
|: comment
 Comment - allows the user to write a message to the system
 manager.

   System Commands:
     answers
     comment
     glossary
     help
     quit
     system
     unknown or ?
     vms
     why
   Note: all system commands are prefixed by the <ESC>
   character.

   To continue type a carriage return.
|: continue
 Is it tawny in colour?
|: $comment
 What comment would you like to make to the system manager?
> In Tennessee we spell it "color"!
> It might please the locals if you use American spellings.<

 Is it tawny in colour?
|: yes
 Is it covered with dark spots?
|: no
 The answer to the question:
       which carnivore is it?
 is lion
|: how
 If the answer to the question:
       is it tawny in colour?
 is yes
 and the answer to the question:
       is it covered with dark spots?
 is no
       then the answer to the question:
       which carnivore is it?
 is lion.
|: continue
 The answer to the question:
       which mammal is it?
 is lion
```

```
|: continue
 The conclusion lion has been accepted.
 The animal is a lion.
|: continue
 The conclusion lion has been accepted.

          Leaving the Expert System.  Hope you had fun.
          Cheers!
```

VITA

Shelley Read Heard was born on April 29, 1956 in Savannah, Ga. He is the son of Mr. and Mrs. Richard W. Heard. He attended high school at Blue Ridge School in Dyke, Va., where he graduated in June 1974. He then attended Vanderbilt University from 1974 to 1976, before transferring to Southwestern at Memphis, where he obtained a Bachelor of Science degree in Biology in June 1979. After working one year in Savannah, he returned to Tennessee to enter the University of Tennessee, Knoxville as a graduate student of Computer Science. He received his Master of Science degree in Computer Science in June 1985.