To the Graduate Council:

I am submitting herewith a dissertation written by Juan Antonio Herrera-Ball entitled "Theoretical Foundations and Algorithms for the Generation of Optimal Decision Trees." I have examined the final copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Electrical Engineering.

R. C. Gonzalez

R. C. Gonzalez, Major Professor

We have read this dissertation
and recommend its acceptance:

J. R. B. Corbett

Michael G Thomason

Donald W. Bouldin

Robert E Bolz Heimer

Accepted for the Council:

Lewis Winkel

Vice Provost
and Dean of The Graduate School

# THEORETICAL FOUNDATIONS AND ALGORITHMS FOR THE GENERATION OF OPTIMAL DECISION TREES

A Dissertation

Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Juan Antonio Herrera-Ball

March 1988

Copyright © Juan Antonio Herrera-Ball, 1988
All rights reserved

En memoria de mis abuelas:

Lola † y Mercedes †

Con cariño a mis padres:

Juan y Gladys

## ACKNOWLEDGEMENTS

I wish to thank Dr. R. C. Gonzalez for the trust he has placed in me and my work. Throughout my college education, Dr. Gonzalez has been a source of inspiration, and his knowledge and experience continues to influence my professional career.

My gratitude also goes to Dr. J. R. B. Cockett for his guidance and assistance in the research, writing and proofreading of this dissertation. Most importantly, I thank Dr. Cockett for his enduring sense of commitment and his invaluable wisdom.

In addition, I would like to thank Dr. R. E. Bodenheimer, Dr. D. W. Bouldin and Dr. M. G. Thomason for serving on my dissertation committee.

I am very grateful to the many individuals at Perceptics Corporation which provided the support and resources which facilitated this investigation. In particular, I am grateful to Dr. J. A. Vrba, Manager, A.I. Applications Division, for providing helpful comments and constructive criticism on early portions of the manuscript.

Finally, my thanks to K. Smith for serving as an inspiration when the going got tough.

This work was supported by Perceptics Co. through a contract with the University of Tennessee. I gratefully acknowledge the financial support given to me by the Venezuelan government in the form of a CONICIT graduate scholarship. Finally, I express my gratitude to Dr. Gonzalez for research assistantships that allowed me to work on very challenging projects.

## ABSTRACT

Theoretical foundations for the generation of optimal decision trees are presented. The approach used is based on recent developments in the area of discrete decision theory and on an existing syntactic algorithm for the reduction of decision expressions. An important characteristic of these developments is the treatment of decision trees as (co)algebraic terms. An overview of relevant results from previous investigations as well as a description of the reduction algorithm is included.

A set of algorithms for finding the irreducible forms of a decision tree are developed. The methods presented perform the syntactic optimization of decision expressions by finding minimal terms with respect to a preorder relation. This preorder relation is associated with a class of cost functions called reasonable cost criteria. Any expected testing cost is a member of this class.

The principal technique introduced is a search strategy called separation planning, which, when combined with the reduction method, yields an algorithm for generating all the optimal forms of a decision tree. The manipulations and algorithms are first presented for the simple case involving decision trees in a discrete decision theory without identities. They are then extended to the case when arbitrary identities are present. Several issues regarding the manipulation of terms in decision theories with arbitrary identities are also discussed. An existing implementation of these algorithms is described and several examples are presented.

## TABLE OF CONTENTS


CHAPTER PAGE

1 INTRODUCTION ..... 1

2 DECISION THEORY FUNDAMENTALS ..... 9

2.1 Theoretical Preliminaries ..... 10

2.2 Decision Tree Manipulations ..... 15

2.3 Decision Tree Equality ..... 18

2.4 Optimality Criteria ..... 22

2.5 Irreducible Terms ..... 26

2.6 The Basic Reduction Algorithm ..... 31

3 FINDING ALL THE IRREDUCIBLES ..... 35

3.1 Separation Planning ..... 37

3.1.1 Separators ..... 38

3.1.2 Separator Plans ..... 43

3.1.3 The Separation Planning Algorithm ..... 65

3.2 Generating All the Irreducible Forms of a Decision Tree ..... 71

4 DECISION THEORIES WITH IDENTITIES ..... 78

4.1 Theoretical Preliminaries ..... 79

4.2 The Algebra of Characterizing Terms ..... 86

4.3 Equality of Terms ..... 96

5 DECISION CLUSTERS ..... 105

5.1 Functional Identities and Decision Clusters ..... 107

5.2 Manipulation of Decision Clusters ..... 113

CHAPTER PAGE

5 (continued)

5.2.1 Migration or Reachable Idempotence ..... 113

5.2.2 Reachable Repetition, Distribution and Transposition ..... 117

5.3 Retrieval Properties of Decision Clusters ..... 124

5.4 Retrieval Complete Knowledge Clusters ..... 130

**6 EXPERIMENTAL RESULTS ..... 138**

6.1 Software Implementation ..... 138

6.2 Generation of Characterizing Terms from Rules ..... 143

6.3 Examples ..... 145

**7 CONCLUSIONS ..... 169**

**LIST OF REFERENCES ..... 171**

**VITA ..... 176**

## LIST OF TABLES

**TABLE** **PAGE**

6.1 Example of days in which a decision to play golf or not has been made in terms of the weather..... 157

## LIST OF FIGURES

**FIGURE** **PAGE**


4.1 Lattice of characterizing identities. ..... 95

5.1 A decision cluster and its components: deduction nodes, antecedent subtree and conclusion subtrees. ..... 115

5.2 Algorithm for finding an irreducible form of a decision cluster..... 123

5.3 Invariance of q-retrieval under idempotence: (a) Original tree; (b) Term after reachable transposition; (c) Resulting term after applying idempotence..... 126

5.4 The lattice of identities functional on some decision $q$. ..... 135

5.5 Minimally retrieval complete term $e'$ obtained from the characteristic term $e_0$. ..... 135

6.1 Block diagram for the Decision Tree Designer (DTD). ..... 140

6.2 Features used to describe the characters. ..... 149

6.3 Rules for recognizing the characters 1, 6, 9, 0, A, C, P, U. ..... 150

6.4 Decision cluster for recognizing the characters 1, 6, 9, 0, A, C, P and U based on number of holes (#holes), hole position (hole-pos), number of bays (#bays) and bay direction (bay-dir). ..... 154

6.5 Stable irreducible forms corresponding to the character recognition decision cluster..... 155

6.6 Irreducible forms with *outlook* at the root..... 159

6.7 Irreducible forms with *temperature* at the root. ..... 161

6.8 Irreducible forms with *humidity* at the root. ..... 162

FIGURE PAGE

6.9 Irreducible forms with *windy* at the root. ..... 164

6.10 Rules for sequential testing procedure. ..... 166

6.11 An optimal sequential testing procedure. ..... 166

# CHAPTER 1

# INTRODUCTION

Decision trees are widely recognized as one of the most computationally efficient means of realizing a function dependent on many variables [31]. In this role, a decision tree can be regarded as a deterministic algorithm for evaluating a discrete function, wherein the action which the algorithm performs next depends on the outcome of some previous test. Decision trees have been used extensively for implementing automated decision-making processes in areas where fast processing speed is essential such as computer vision, pattern recognition and intelligent control [17, 18, 21, 23, 27, 32, 39, 40, 49]. More recently, they have been used in the design of expert systems [44, 45, 46].

Most often, however, decision trees are used to represent classification problems. In this application, they have often been criticized due to the fact that, compared to rule-based methodologies, decision trees are more difficult for humans to understand and manipulate, they rapidly become unmanageable with increasing size, and they are not modular [15]. Decision trees, however, are very explicit about the sequencing of the computation needed to implement them on sequential machines. Moreover, they can implement any given control structure, something that standard rule-based methodologies alone cannot always guarantee [13].

The broad applicability of decision trees has resulted in the need for designing decision trees that are computationally efficient. Unfortunately, the generation of optimal decision trees for most problems of practical significance is a cumbersome task typically requiring considerable development and testing effort.

An optimality criterion for a decision theory is a cost function that associates a real value to each term of the theory to indicate the cost of evaluation, storage or testing [31]. Thus, optimizing a tree is equivalent to finding a term with minimal cost among the class of terms that are semantically equivalent to that decision tree.

The task of heuristically optimizing a tree is quite formidable even for the most simple applications. It is well known that the problem of constructing optimal decision trees from either a decision table or a set of examples (rules) is NP-complete on the number of decisions [25]. Some authors have applied standard search algorithms to find, from all the possible trees, one that is optimal with respect to some optimality criterion (usually, the expected testing cost). Depending upon the particular search technique used they fall under one of two major categories: *dynamic programming* algorithms and *branch-and-bound* methods.

Dynamic programming algorithms [1, 19, 28, 29, 34] build an optimal solution from the bottom up by combining the solution to successively larger subproblems. For a decision tree involving $n$ decisions, each with $k$ possible outcomes, these algorithms require a number of operations proportional to $n \cdot k \cdot (k+1)^{n-1}$. Furthermore, the process requires enough storage to hold a lattice consisting of $(k+1)^n$ nodes and $n \cdot k \cdot (k+1)^{n-1}$ edges, in addition to enough storage to hold the resulting tree. In fact, this technique is so prohibitively expensive in terms of storage that in practice it has been feasible only for relatively small problems (12 decisions or less).

Branch-and-bound techniques [2, 37, 38], on the other hand, always develop a partial solution which is potentially less expensive than any other, as determined by a lower bound function, until one solution has been completely developed. The problem with these methods is that they may result in a nearly-exhaustive search of all the possible trees. In

fact, the number of partial solutions developed at level $x$ in a decision tree is bounded by $[n\text{-}(x\text{-}1)]2^{x-1}$ [2]. Thus, the algorithm runs in exponential time regardless of the input size.

Other researchers have reported methods based on the reduction of Boolean functions to canonical or minimal formulas. These methods usually require the solution to known NP-hard problems like determining the prime implicants of a function [4, 43], establishing a minimal disjoint set cover for the solution [30] or the synthesis of P-functions [42]. In general, these techniques are of little practical value since they introduce NP-hard problems as part of the solution.

Given that they use an expected testing cost as the criterion for optimality, many of the optimization techniques that have been reported assume the *a priori* knowledge of the probability of occurrence of each decision rule in the problem. Since that type of data is usually based on estimates, some authors have instead chosen to study the use of heuristics to search for sub-optimal solutions. Some have tried to quasi-optimize decision trees by likening the problem to one of refining an initial uncertainty about the function's value into a certitude by minimizing an entropy formulation of the function [35, 47, 48]. This family of greedy algorithms uses some heuristic measure, e.g. the activity of a decision variable [33], to perform a local optimization in a fashion similar to branch-and-bound but without backtracking. It has been shown that, although this type of heuristics is asymptotically optimal on the average, there are problems for which the result can have an expected testing cost arbitrarily larger than the optimal solution [32]. Finally, some authors have used *ad hoc* criteria in the minimization of completely specified Boolean functions [3, 22]. However, these methods require the generation of all the prime implicants or the minimization of the disjunctive form of the function, both of which are NP-hard problems.

The techniques mentioned above fail to utilize the inherent structure of decision trees. Furthermore, the information required by these methods is frequently not available and

some unwarranted assumptions must be made about the form of the criterion. As a result, these algorithms can only give sub-optimal solutions.

As the result of this investigation, a series of optimization techniques that make use of the information inherent in the structure of decision trees, and which also improve on the complexity of existing algorithms were developed. In addition, these techniques have been implemented successfully on conventional computer systems and have been useful in providing a methodology for the automatic generation of optimal decision trees.

The approach followed uses the (co)algebraic properties of decision trees developed by Cockett [7, 9, 10]. This formulation provides a calculus of manipulations for decision expressions. The ability to syntactically manipulate decision trees not only makes them a viable and flexible form for the representation of knowledge, but it also opens the possibility of performing syntactic optimization. Therefore, trees which have been designed heuristically for a particular problem may be optimized without requiring any knowledge of the semantics. This is an essential prerequisite for performing automatic optimization.

The reduction algorithm presented in [12] has these properties in addition to providing an improvement over the complexity of previous approaches. For a $k$-ary tree with $n$ decisions, the storage required by the reduction algorithm is bounded by $k \cdot T$, where $T$ is the size (number of nodes) of the tree[1]. This worst-case upper bound is much smaller than that of dynamic programming methods which require enough storage for $(k+1)^n$ nodes and $n \cdot k \cdot (k+1)^{n-1}$ edges. The reduction algorithm runs in time proportional to $n^3 \cdot T$ [12]. Thus, for a small number of decisions it runs in time comparable to that of dynamic programming. However, for large numbers of decisions, the reduction algorithm is

---

[1] The size of the tree is in turn bounded by $\sum_{i=0}^{n-1} k^i = \frac{k^n - 1}{k - 1} < k^n$.

considerably better than dynamic programming as the exponential term of the latter takes over the complexity of the algorithm. Furthermore, the reduction algorithm, unlike dynamic programming, takes advantage of the fact that the size of the tree is generally smaller than the upper bound of $k^n$. The improvement over branch-and-bound methods is even more significant, since, due to the large number of subproblems generated, the time and storage required by those methods is exponential regardless of input size.

The algebraic formulation of decision theories gives rise to several significant forms. One of these, the *irreducible form*, plays a critical role in the topic of this work. In fact, one of the goals of this research effort was to develop a set of algorithms for generating all the irreducible forms of a decision expression.

The notion of an irreducible tree was first introduced in [8]. An irreducible decision expression is an efficient computational form as every irreducible can be shown to be optimal with respect to some expected cost criterion [12]. Conversely, given an expected testing cost, the solution to the optimization problem is an irreducible tree. The optimization problem can thus be solved by developing techniques to rewrite decision trees into irreducible forms.

The process of optimizing a decision tree with respect to a broad class of cost criteria, by reducing it to an irreducible form, is non-deterministic [12]. In general, there are many decision trees which are optimizations of a given tree. Thus, in the absence of any other costing information, it is very useful to generate all the optimal forms of the tree so that the problem of selecting the tree or set of trees best suited for a particular application can be narrowed down to a small number of sensible choices.

A set of algorithms for the syntactic manipulation of decision trees and the generation of all the optimal forms of a decision tree was developed as part of this work. In order to find all the irreducible forms of a decision tree in an efficient way, however, an awkward

problem had to be overcome: there is a rather trivial manner in which trees can be computationally equivalent, called *transposition equivalence*. Obviously, it is important to avoid producing any of the many different transposes of the same optimal tree when generating irreducibles. To solve this problem, a technique called *separation planning* was developed by the author and is introduced here. In fact, separation planning provides the basis of all the optimization algorithms developed during the course of this investigation. Separation planning also facilitates the search for a particular class of irreducible forms by narrowing down the size of the search space.

Frequently, there exists a number of dependencies between the decisions in a decision tree. As a consequence of these dependencies, the outcome of some set of decisions may have implications concerning the possible outcomes of another set of decisions. These dependencies complicate the optimization problem since they introduce an additional set of identities which semantically equivalent trees must always satisfy. A set of primitive operations to manipulate decision trees while maintaining the overall semantic integrity of the system were also developed during this research effort and are reported here.

It should be noted that, when viewed as an alternative representation for knowledge, decision trees have the characteristic of embedding both logic and control information within their structure. Unfortunately, this has the adverse effect of making it difficult to separate the knowledge from its use. The development of expert systems technology has served to demonstrate the many advantages of keeping the specification of the knowledge separate from its procedural implementation. Yet, the use of a standard inference engine to generate the procedural implementation of rule-based systems has been shown to be inadequate, since they alone cannot provide the means for implementing arbitrary control sequencings [13].

The solution to some of these problems seems to point to a combination of both production rule and decision tree methodologies, where the correspondence between both representations is fully exploited. In this role, each of the resulting decision expressions (knowledge clusters) is considered as an independent module of knowledge which combines both logical and procedural information and satisfies a set of arbitrary decision identities. Each decision cluster can then be considered as a functional specification for some decision problem. A set of rules for manipulating decision clusters syntactically, while preserving the semantics of the knowledge embedded in them is presented here. The desire to combine both rule and decision tree based methodologies was the principal motivation for the implementation developed by the author using the theoretical results and algorithms presented here. An overview of that implementation is included later in this dissertation.

The rest of this dissertation is organized as follows. Chapter 2 provides an overview of the theoretical formulation of discrete decision theories developed by Cockett and presents the basic terminology and the notation which is used throughout the dissertation. Chapter 3 introduces the main results developed by the author regarding the theoretical foundations of this work. This chapter also introduces several important algorithms, including the separation planning technique, and an algorithm for finding all the irreducible forms of a decision tree. Chapter 4 deals with decision trees with identities and discusses a series of results regarding characterizing terms (a very useful form for representing dependencies between decisions). This chapter discusses some results originally developed by Cockett and introduces some useful extensions developed by the author. Chapter 5 presents more results developed by the author, from which the extensions of the results of Chapter 3 were developed, to handle the manipulation and optimization of decision trees in the presence of arbitrary decision identities. In addition, Chapter 5 introduces the notion of decision

clusters for representing complete modules of logical and procedural knowledge. Chapter 6 describes an implementation of the techniques developed here, and presents several examples that serve to illustrate some applications of the results of this research effort. Finally, Chapter 7 gives an overview of what was accomplished by this investigation and provides some topics for further research.

# CHAPTER 2

## DECISION THEORY FUNDAMENTALS

The purpose of this chapter is to provide an overview of the theoretical formulation of discrete decision theories. This discussion provides the reader with the basic terminology and the notation which will be used throughout this work. In addition, a survey of previous results related to decision theories is included.

The chapter is organized in six sections. The first section introduces some preliminary concepts regarding decision theories and discusses a special set of algebraic decision identities. These identities are complete in the sense that they allow the derivation of any equality which holds in a non-trivial decision theory. The theoretical formulation adopted in this work is based on the algebraic formulation that was initially developed by Cockett in [7-11]. This formulation uses decision identities consisting of idempotence, distribution, and repetition to provide a calculus for manipulating decision expressions.

The second section presents several decision tree manipulations in the form of rewriting rules. These rewriting rules first appeared in [7, 8]. For a more recent discussion on the subject see [10]. The third section deals with different aspects of decision tree equality and introduces a fourth decision identity, transposition. In that section, the issue of equality is limited to decision theories without identities.

The fourth section describes the class of optimization criteria which is used in this work. The criteria described use a preorder relation on the terms of a decision theory. Since semantically equivalent trees may have syntactically different forms, the reasonable

preorder is used to determine which one is "better." Minimals in this preorder are most important since they give the solution to the optimization problem.

The fifth section introduces the concepts of irreducibility and irreducible terms. Several significant forms of decision expressions are discussed. The irreducible form, however, is the most important one for this research. An irreducible decision expression is an efficient computational form as every irreducible can be shown to be optimal with respect to some expected cost criterion [8, 12].

Finally, the sixth section presents the basic decision tree reduction algorithm along with a discussion of its features. A description of this algorithm first appeared in [12]. Given that there may exist many irreducible forms of a decision tree, the reduction algorithm provides a particular method for rewriting a decision tree into an irreducible form.

## 2.1 Theoretical Preliminaries

A decision tree can be regarded as a term in a decision theory. A decision theory is a coalgebraic theory in that instead of interpreting its function symbols as operations:

$$f\colon A * \ldots * A \rightarrow A,$$

they are interpreted as *cooperations*, that is, maps of the form:

$$g\colon A \rightarrow A + \ldots + A,$$

where "+" means the disjoint union. Whereas the notation commonly used for operations is the postfix notation

$$\langle x_1 , \ldots , x_n \rangle \cdot f,$$

the notation used here for cooperations will be that of

$$g \cdot \langle x_1 \,' \ldots ' \, x_n \rangle,$$

where apostrophes are used instead of commas to reflect the duality between the two concepts and the dot is dropped for convenience.

The manipulation of cooperations is exactly the same as that for operations as they are dual concepts. However, their effect is rather different. In particular, cooperations partition the base set into subsets, which correspond to the members of the disjoint union. This effect is similar to what might be expected of a decision.

One of the advantages of formulating decision trees in algebraic terms as a decision theory is that the manipulations of decision trees then become the result of the identities governing decisions and any further identities peculiar to the theory.

**Definition 2.1**

A *decision theory* $D = (Q, E)$ consists of a set $Q$ of cooperation symbols and a set, $E$, of identities.

Each $q \in Q$ with arity $n \geq 2$, has a codomain $\{a_1, \ldots, a_n\}$ associated with it.

The identities of E are of the form $t_i = t_j$ where $t_i$ and $t_j$ are terms of $D$.

The terms of the decision theory are defined recursively as follows:

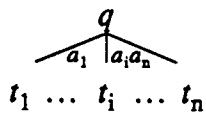
(i) There is a stock of variables $x_1, x_2, \ldots$, each of which is a term.

(ii) If $t_1, \ldots, t_n$ are terms and $q$ is a cooperation symbol of arity $n$, then

$$q \, \langle t_1 \,' \ldots ' \, t_n \rangle$$

is a term.

Thus, given a decision $q \in Q$, with arity $n \geq 2$ and codomain $\{a_1, \ldots, a_n\}$, the notation $q \, \langle t_1 \,' \ldots ' \, t_n \rangle$ is used to denote the following decision tree:




where $q$ is called the *root decision.*

Every decision theory must satisfy the following decision identities for every $q, q_1, q_2 \in Q$:

**[D.1]** (Idempotence)

$$q\langle x\,'\ldots\,'x\rangle = x$$

**[D.2]** (Distribution)

$$\begin{aligned} q_1\langle x_1\,'\ldots\,'x_{i-1}\,'q_2\langle y_1\,'\ldots\,'y_m\rangle\,'x_{i+1}\,'\ldots\,'x_n\rangle \\ = q_2\langle q_1\langle x_1\,'\ldots\,'x_{i-1}\,'y_1\,'x_{i+1}\,'\ldots\,'x_n\rangle\,'\ldots\,' \\ q_1\langle x_1\,'\ldots\,'x_{i-1}\,'y_m\,'x_{i+1}\,'\ldots\,'x_n\rangle\rangle \end{aligned}$$

**[D.3]** (Repetition)

$$\begin{aligned} q\langle x_1\,'\ldots\,'x_{i-1}\,'q\langle y_1\,'\ldots\,'y_n\rangle\,'x_{i+1}\,'\ldots\,'x_n\rangle \\ = q\langle x_1\,'\ldots\,'x_{i-1}\,'y_i\,'x_{i+1}\,'\ldots\,'x_n\rangle \end{aligned}$$

A decision theory must always satisfy the decision identities [D.1]-[D.3]. In addition, any further identities, $E$, which are peculiar to that particular decision theory, must also be satisfied. As it was shown in [9], if the cooperations in a coalgebraic theory satisfy [D.1], then every interpretation of the theory must also satisfy [D.1]-[D.3], even though [D.2] and [D.3] cannot be algebraically derived from [D.1]. The main result of [9] was the completeness of the decision identities idempotence, distribution and repetition. In other words, It was shown that [D.1]-[D.3] are sufficient to derive any identity which holds for any decision theory, even when identities over and above [D.1]-[D.3] are present.

The cooperation symbols in $Q$ will be referred to as decisions. Intuitively, making a decision entails determining the answer to a given question. Although a decision is always associated with some question-answer pair, it is not uncommon to use both question and decision interchangeably. Thus, a decision will not always be distinguished from its question.

The following is an example of a decision tree.

**Example 2.2**

Consider the following problem. George has three daughters: Sue, Pam and Joy.

- Sue and Joy have brown hair.
- Pam's hair is red.
- Sue always wears a skirt, while Joy never does.
- Pam may or may not wear a skirt.

One day, the owner of a candy store found out that one of George's daughters had stolen some candy from his store. The owner of the store knew George, the girl's father, but not the girl's name. So, he went to George and ask for his money. George paid the owner, and asked him the girl's name, but the owner did not know.

How can George find out which of his daughters should be punished?

The problem can be described by the following rules.

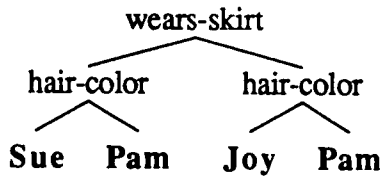
**Rule 1:** If the girl wears a skirt and has brown hair then her name is Sue.

**Rule 2:** If the girl does not wear a skirt and has brown hair then her name is Joy.

**Rule 3:** If the girl has red hair then her name is Pam.

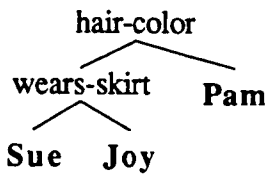
These rules are *prime* in the sense that none of the conditions in the rules can be omitted without changing the meaning of the problem.

From this set of rules, one can generate a decision tree representation of the problem by using backward chaining. The result is the following decision tree:




The first question in the tree tries to determine whether the girl is wearing a dress. If the codomain (answer set) for this decision is {*yes*, *no*}, then a *yes* answer means that the first argument (left-most branch) of the tree is followed. There, the second question -- What color is the girl's hair? -- is encountered. Assuming a codomain {*red*, *brown*}, a *brown* answer would yield the result: the girl's name is *Sue*. Comparing this decision tree to the set of rules reveals that both are equivalent representations of the problem.

However, the result of backward chaining on the rules is not the most efficient tree. In fact, changing the order in which the decisions are asked results in a better decision tree:




As it will be shown later, the latter decision tree representation is the only solution to the problem which is optimal with respect to a *reasonable cost criterion.* ■

## 2.2 Decision Tree Manipulations

All the intuitive facts about decision theories, which are expected to hold, actually do hold in decision theories. In particular, as it was shown by Cockett in [7], if every cooperation satisfies the decision identities [D.1]-[D.3], then every term of the theory does so as well.

A term, $w$, will be denoted by $w\langle z_1 \,'\, \ldots \,'\, z_n\rangle$, where $z_1, \ldots, z_n$ are the variables of the term. Substitution will be indicated by the replacement of the appropriate argument. So for instance, given the terms $w, t_1, \ldots, t_n$, the notation $w\,\langle t_1 \,'\, \ldots \,'\, t_n\rangle$ will denote the substitution given by

$$w\,\langle t_1 \,'\, \ldots \,'\, t_n\rangle = w\,\langle z_1 \,'\, \ldots \,'\, z_n\rangle\,[z_1 \leftarrowtail t_1, \ldots, z_n \leftarrowtail t_n],$$

where $z_i \leftarrowtail t_i$ means *replace the variable* $z_i$ *with the term* $t_i$.

The following set of rewriting rules can be derived from the application of the decision identities [D.1]-[D.3] to the terms of a decision theory. The proofs for the following statements are omitted here but can be found in [10].

*Repeat Reduction Rewriting Rule*: If $D = (Q, E)$ is a decision theory with $q \in Q$ and $W$ a term in $D$, then

$$q\langle x_1 \,'\, \ldots \,'\, x_{r-1} \,'\, W\langle y_1 \,'\, \ldots \,'\, y_{s-1} \,'\, q\langle z_1 \,'\, \ldots \,'\, z_n\rangle \,'\, \ldots \,'\, y_m\rangle \,'\, \ldots \,'\, x_n\rangle$$

$$\Downarrow$$

$$q\langle x_1 \,'\, \ldots \,'\, x_{r-1} \,'\, W\langle y_1 \,'\, \ldots \,'\, y_{s-1} \,'\, z_r \,'\, \ldots \,'\, y_m\rangle \,'\, \ldots \,'\, x_n\rangle$$

The repeat reduction rewriting rule states that if a decision $q$ is encountered more than once in any one path of the decision tree, then it can be removed by assuming that any

repeated occurrence of $q$ on a path will be answered the same way it was answered the first time.

**Definition 2.3**

A decision tree, $t$, is *repeat reduced*, if there is no path in the tree which has a repeated decision.

Any tree, $t$, can be reduced to a repeat reduced tree $rp(t)$ by successively using the repeat reduction rewriting rule until the rewriting terminates.

*Idempotence Rewriting Rule*: If $D = (Q, E)$ is a decision theory with $q$ a decision in $D$, and $x$ a variable, then:

$$q\langle x\ '\ldots\ '\ x\rangle \Rightarrow x$$

**Definition 2.4**

A decision tree, $t$, is *idempotent reduced*, if there is no cooperation in $t$ which can be eliminated by the use of the idempotence rewriting rule.

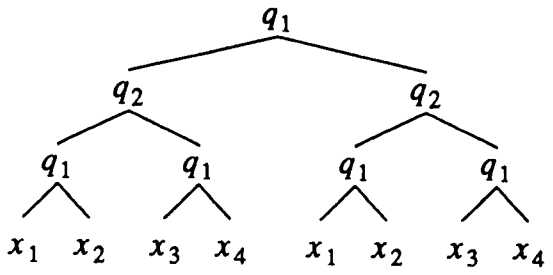
Using the idempotence rewriting rule, any tree, $t$, can be reduced to an idempotent reduced tree $id(t)$.

A simple way of improving a decision tree, is therefore, to remove repeated questions by means of the repeat reduction rewriting rule, and to use the idempotence rewriting rule to get rid of those decisions whose outcome is exactly the same regardless of how they are answered.

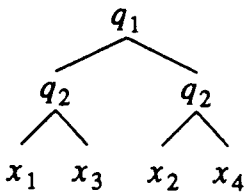
**Definition 2.5**

A term of a decision theory is said to be *simply reduced* if it is both idempotent reduced and repeat reduced.

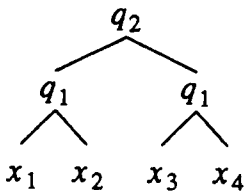
Repeat reduction and idempotent reduction are not confluent operations. For instance, consider the following decision tree:




Applying the repeat reduction rewriting rule results in




On the other hand, applying the idempotence rewriting rule to the original tree yields




Note that both forms are now simply reduced so that no more applications of either the repeat reduction or the idempotent rewriting rules are possible. Furthermore, in general, the application of the idempotence rule before the repeat reducing rule might require an additional application of the idempotence rule. Therefore, the process of obtaining a simply reduced term, called *simple reduction*, is defined as a repeat reduction followed by an idempotent reduction.

## 2.3 Decision Tree Equality

As it was observed in the definition of a decision theory, the identities in $E$ are all of the form $t_1 = t_2$. This gives rise to the idea of equality between terms in a decision theory. There are several notions of equality all of which will be carefully distinguished. Two such notions of equality are introduced below.

**Definition 2.6**

(i) Two terms, $t_1$ and $t_2$, in a decision theory, $D = (Q, E)$, are *structurally identical*, denoted $t_1 \equiv t_2$, if and only if, $t_2$ is the exact copy of $t_1$.

(ii) Two terms, $t_1$ and $t_2$, in a decision theory, $D = (Q, E)$, are *decision equivalent*, denoted $t_1 =_d t_2$, if and only if, they are equivalent using [D.1]-[D.3] and the identities in $E$.

Whether two trees are structurally identical can be easily decided by comparing the root decision for both trees and then recursively comparing their corresponding arguments. Whether two trees are decision equivalent, on the other hand, is a little harder to decide, since two trees are decision equivalent when each can be obtained from the other by using [D.1]-[D.3]. Two trees are decision equivalent when they represent the same decision process. This could be exhaustively checked by listing the outcome for each combination of answers to the decisions.

**Example 2.7**

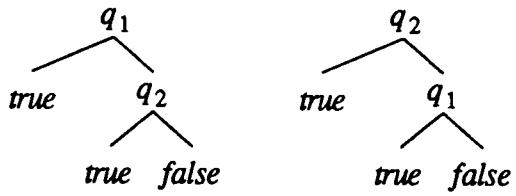
Consider the following two trees given in algebraic form:

$$t_1 = q_1 \langle true \text{ ' } q_2 \langle true \text{ ' } false \rangle \rangle$$

$$t_2 = q_2 \langle true \text{ ' } q_1 \langle true \text{ ' } false \rangle \rangle.$$

The goal of this example is to show that both trees are decision equivalent.

If $q_1$ and $q_2$ are two questions that can be answered either *yes* or *no*, then both trees can be represented graphically as follows:




where the first argument is taken when the decision is answered *yes* and the second argument is taken when it is answered *no*. Further, whenever a leaf labeled $x$ is reached, it means that the tree concludes $x$ at that leaf.

Observe that both trees arrive at the conclusion *true* whenever either $q_1$ or $q_2$ is answered *yes* and that they both conclude *false* when $q_1$ and $q_2$ are both answered *no*. Therefore, they both represent the same decision process. ■

If $t_1$ and $t_2$ are terms in of a decision theory, and a copy of $t_2$ is placed at each leaf of $t_1$ to obtain $t_1/t_2$, it is then clear that $t_1/t_2 = t_2$ by idempotence. This idea and the application of simple reduction gives a simple way of deciding whether or not two terms are decision equivalent. This is shown in the following proposition, the proof to which is found in [8].

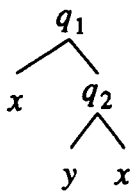
**Proposition 2.8**

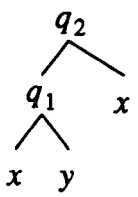
Two terms, $t_1$ and $t_2$, in a decision theory $D = (Q, E)$, are decision equivalent, if and only if, $id(rp(t_1/t_2)) \equiv id(rp(t_1))$.

The following example illustrates how this proposition can be used to decide whether or not two terms are decision equivalent.

**Example 2.9**

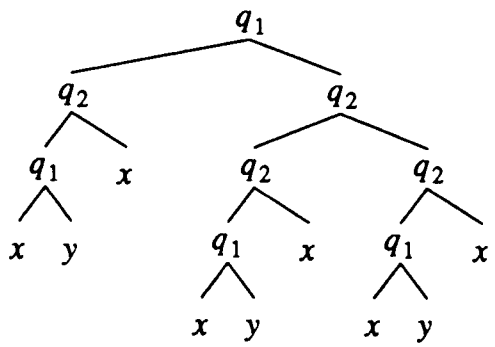
Given two decision trees,

$t_1 =$ 

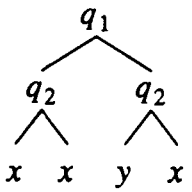

$t_2 =$ 


show that $t_1 =_d t_2$.

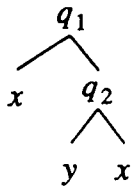
First, construct $t_3 = t_1/t_2$ by replacing the conclusions in $t_1$ with $t_2$.

$t_3 =$ 


Second, repeat reduce $t_3$ to obtain $rp(t_3)$.

$rp(t_3) =$ 


Finally, idempotent reduce to obtain $id(rp(t_3))$.

$id(rp(t_3)) =$ 


Since $t_1$ is simply reduced, and $id(rp(t_1/t_2)) \equiv t_1$, then $t_1$ is decision equivalent to $t_2$. ■

Another approach to determining the equality of terms in a decision theory compares the paths of the terms. A path in a decision tree is described by means of the list that contains all the decisions encountered in that path (and how they are answered), and the conclusion that is reached at the leaf at the end of the path. Such a pair is called a *spine*. Each decision and answer pair, denoted $q \diamond a$, is called a *branch instruction* and intuitively means that after the decision $q$, the argument $a$ is chosen.

**Definition 2.10**

The pair $(\{q_1 \diamond a_1, ..., q_n \diamond a_n\}, x)$ is a *spine* of a term, $t$, if there exists a path in $t$ in which, after responding to the decision $q_1$ with the answer $a_1$, $q_2$ with $a_2$, ..., and $q_n$ with $a_n$, the variable $x$ is reached at the leaf.

The set of all the spines in a term is called the *spine set*. A spine, $(L, x)$, can be interpreted as the list, $L$, of branch instructions that leads to the variable $x$. Two terms are then said to have *bag equivalent* spine sets if, as unordered lists of branch instructions, they both have equivalent spines leading to the same conclusions. Spine set bag equivalence is a very important property for decision trees, since the spine sets of two trees are bag equivalent when they arrive at exactly the same variables after the same decisions are made, even when those decisions are possibly taken in a different order.

Consider now the transposition identity (due to K. Chen and Z. Ras [6]), which is a consequence of [D.1]-[D.3]:

**[D.4]** (Transposition)

$$q_1\langle q_2\langle x_{11} \,'\, ... \,'\, x_{1m}\rangle \,'\, ... \,'\, q_2\langle x_{n1} \,'\, ... \,'\, x_{nm}\rangle\rangle$$
$$= q_2\langle q_1\langle x_{11} \,'\, ... \,'\, x_{n1}\rangle \,'\, ... \,'\, q_1\langle x_{1m} \,'\, ... \,'\, x_{nm}\rangle\rangle$$

The transposition identity provides an alternative way of stating the basic axioms of decision theories. As it was shown in [10], these identities are related in the following way:

(i) The identities [D.2] and [D.3] imply [D.4].

(ii) The identities [D.1] and [D.4] imply [D.2].

In fact, any two terms $W_1$ and $W_2$ also satisfy the transposition identity in the same way that they satisfy [D.1]-[D.3] (see [10]).

**Definition 2.11**

Two terms, $t_1$ and $t_2$, in a decision theory $D = (Q, E)$, are *transposition equivalent*, denoted $t_1 =_t t_2$, if $t_2$ can be obtained from $t_1$ after a finite number of transposition rewritings.

The transposition law defines an equivalence relation among decision trees, whose equivalent classes consist of transposition equivalent trees. In particular, an important result regarding transposition equivalence is given by *Chen and Ras's Lemma* which says that two repeat reduced terms have equivalent path bag sets, if and only if, they are transposition equivalent. This result implies that two transposition equivalent trees are equally efficient since the same number of decisions is involved in both decision processes. In general, a better tree is only obtained when some step in the decision process can be omitted. Therefore, in order to find all the optimal forms of a decision tree, it is only necessary to obtain a traversal of all the optimal transposition classes.

## 2.4 Optimality Criteria

An optimization criterion for a decision theory is a cost function that associates a value to each decision expression in the theory to indicate either the cost of evaluation, storage, or

testing of a term [31]. Decision expression optimization is thus the process of finding a term with minimal cost among the class of terms that are decision equivalent to that expression. An optimization procedure must try to reduce the cost of a term without changing the meaning of the decision process embodied in the original expression.

The purpose of this section is to describe the class of optimization criteria considered here, the *reasonable cost criteria.* A reasonable cost criterion must agree with a *reasonable preorder* [8]. A preorder is a relation that is both reflexive and transitive but not necessarily anti-symmetric. Thus, a preorder can be generated from any relation by forming the transitive-reflexive closure. Since the reasonable preorder is not anti-symmetric, two elements are said to be equivalent in the preorder, if they are both less than or equal to each other.

A *reasonable preorder relation,* $\leq_r$, in a decision theory is generated by the following:

**[RP.1]** Monotonic:

$$q\langle x_1 \text{ '} \ldots \text{ '} y \text{ '} \ldots \text{ '} x_n\rangle$$
$$\leq_r$$
$$q\langle x_1 \text{ '} \ldots \text{ '} z \text{ '} \ldots \text{ '} x_n\rangle$$

whenever, $y \leq_r z$

**[RP.2]** Idempotent well ordered:

$$x \leq_r q\langle x \text{ '} \ldots \text{ '} x\rangle$$

**[RP.3]** Transposition invariant:

$$q_1\langle q_2\langle x_{11} \text{ '} \ldots \text{ '} x_{1n}\rangle \text{ '} \ldots \text{ '} q_2\langle x_{m1} \text{ '} \ldots \text{ '} x_{mn}\rangle\rangle$$
$$\leq_r$$
$$q_2\langle q_1\langle x_{11} \text{ '} \ldots \text{ '} x_{m1}\rangle \text{ '} \ldots \text{ '} q_1\langle x_{1n} \text{ '} \ldots \text{ '} x_{mn}\rangle\rangle$$

**[RP.4]** Repeat reducing:

$$q\langle \ldots , x_{i-1} , W\langle \ldots , q\langle \ldots , y_i , \ldots \rangle , \ldots \rangle , \ldots \rangle$$

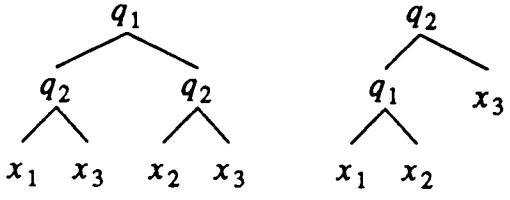
$$\geq_r$$

$$q\langle \ldots , x_{i-1} , W\langle \ldots , y_i , \ldots \rangle , \ldots \rangle$$

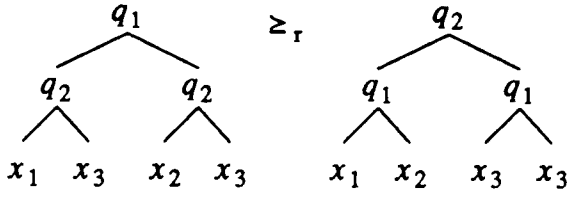
Under the preorder relation, $\leq_r$, two terms, $t_1$ and $t_2$, are *reasonably equivalent*, denoted $t_1 =_r t_2$, if and only if, $t_1 \geq_r t_2$ and $t_1 \leq_r t_2$. Furthermore, it is easy to see that if $t_1$ and $t_2$ are repeat reduced terms, then $t_1 =_r t_2$, if and only if, $t_1$ and $t_2$ are transposition equivalent [8].

**Example 2.12**

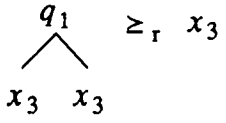
Consider the following two decision trees



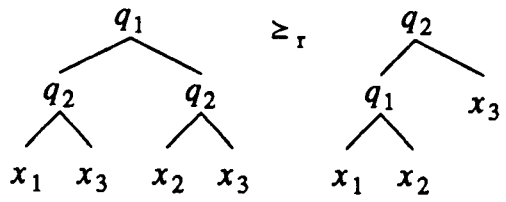

Since the preorder is transposition invariant, it follows that




Also, the preorder is idempotent well ordered, i.e.,




Therefore, since the preorder is monotonic, it follows that




■

Observe that the trees of the previous example are *semantically* equivalent, i.e. they have the same meaning as they are decision equivalent according to the criteria used in the previous section on the equality of terms. *Syntactically*, however, they are different forms since, as the example shows, one is always syntactically better than the other in terms of the reasonable preorder.

**Definition 2.13**

A *reasonable cost criterion*, $c$, on a decision theory $D = (Q, E)$, with terms $T(D)$, is a map:

$$c\colon T(D) \to R^+; \qquad t \mapsto c(t)$$

such that the cost function, $c$, is order preserving as a map from the terms of the decision theory with the reasonable preorder, to the real numbers with the usual ordering. Thus, if $t_1 \preceq_r t_2$, then $c(t_1) \leq c(t_2)$.

A reasonable criterion is thus a cost function which preserves the reasonable preorder. It is easy to show that both repeat reduction and idempotent reduction will not increase the cost of a term with respect to a reasonable criterion as $t \succeq_r rp(t)$ and $t \succeq_r id(t)$. In fact, as shown in [8], given a reasonable cost criterion, the cost of an expression which has been simply reduced, is no more than the cost of the original expression. Furthermore, for every decision expression, there is an equivalent expression of minimal cost. Obviously,

given any decision expression, it is useful to find an equivalent form of minimal cost. These optimal forms are discussed next.

## 2.5 Irreducible Terms

The problem of optimizing a decision tree with respect to a reasonable criterion is that of finding a *minimal element* in the reasonable preorder. In a preorder, a minimal element in it is one which is equivalent to every element below it. Such a term in the reasonable preorder is said to be an *irreducible*.

**Definition 2.14**

(i) A term, $t$, in a decision theory $D = (Q, E)$, is said to be *irreducible*, if and only if, for every term $t_0$, decision equivalent to $t$, then $t_0 \leq_r t$ implies $t_0 =_r t$.

(ii) Given the terms, $t, t_1, t_2$, in a decision theory $D = (Q, E)$, such that $t =_d t_1 =_d t_2$, then $t_1$ and $t_2$ are said to be (distinct) *irreducible forms* for $t$, if and only if, $t_1$ and $t_2$ are both irreducible trees, and $t_1$ and $t_2$ are not transposition equivalent.

Before a procedure for obtaining an irreducible form for a decision tree can be discussed, a closer look at the problem of optimizing a decision tree is in order.

**Definition 2.15**

(i) A decision, $q$, is said to be *semi-essential* in a term $t$, if $q$ occurs on every path of the term.

(ii) A decision, $q$, is said to be *essential* in a term $t$, if $q$ is semi-essential in every t' which is decision equivalent to $t$.

Intuitively, a decision $q$ is essential in $t$, if it cannot be omitted no matter how the overall decision process represented by $t$ is expressed. This leads to the following result, which was originally presented in [8].

**Lemma 2.16**

If $q$ is essential in $t$, then every $t'$ decision equivalent to $t$ is transposition equivalent to a term with $q$ at the root.

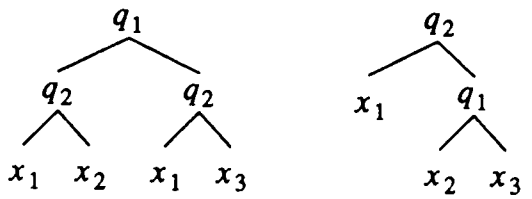
**Proof.**

The proof follows from the fact that $q$ is semi-essential in every $t'$. ■

Whenever an essential decision is encountered in a decision tree, it might be wise to try to resolve that question as soon as possible since it is obvious that its outcome is vital to the decision process represented by the tree. Therefore, the recognition of the occurrence of such decisions is important to the optimization procedure. After spotting an essential decision, $q$, in the term $t$, one should probably make it be the root of the term and then try to minimize the subtrees independently. The same technique can then be applied recursively to each argument of $q$.

**Example 2.17**

Consider the following decision equivalent terms:




$q_2$ is semi-essential on both terms and therefore it is an essential decision. On the other hand, $q_1$ is semi-essential in the first term but not in the second.

Observe that the second form of the tree cannot be reduced any further since the root of every subterm is essential in it. ■

**Definition 2.18**

(i) A decision, $q$, is said to have a *redundant occurrence* in a term, $t$, if there exist a term $t'$, decision equivalent to $t_q$, the subterm standing at the occurrence of $q$ in $t$, in which $q$ does not occur.

(ii) A decision, $q$, is said to be *semi-redundant* in a term $t$, if there is a path in $t$ on which $q$ does not occur.

Basically, a decision has a redundant occurrence in a decision tree if, regardless of how $q$ is answered, the outcome of the decision process is the same. Although it is usually desirable to remove any redundant decision from the tree, in some cases, the occurrence of such a decision can improve the performance of the tree. For instance, suppose that a decision $q$ can be answered *yes* or *no*. If the subtree after the branch that gets answered with *yes* is decision equivalent to the subtree after the branch that gets answered with *no*, then it is obvious that the way $q$ is answered is irrelevant as far as the logic of the decision process is concerned, and it could always be avoided. However, it is possible for an irreducible term to have a redundant occurrence of a decision (see [12]). In this role, a redundant decision is used to control the decision process rather than to affect the outcome. Basically, the decision chooses between the various alternative optimal forms available for its decision equivalent arguments. Of course, if the cost of this type of control is more than the gain obtained in the customized arguments, then the use of a redundant decision would not be justified.

Control decisions occur frequently in practice. For instance, knowing that a patient is hospitalized greatly changes the probability of him/her having certain infections. This in turn changes the approach taken in diagnosing the infection and discovering the disease.

However, the presence of a particular infectious disease is logically independent of whether or not the patient is hospitalized.

One way of dealing with redundant decisions is to assume that the designer intended for that decision to act as a control question and that the optimization should not get rid of it.

**Definition 2.19**

(i) A term, $t$, is said to be *semi-essentially simple* if the only semi-essential decision in $t$ is its root decision.

(ii) A simply reduced term is said to be *semi-simple*, if each of its sub-terms is semi-essentially simple.

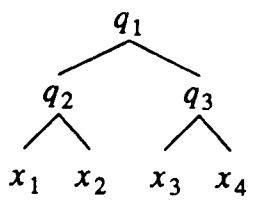
(iii) A term, $t$, in a decision theory $D = (Q, E)$, is said to be *decision delayed* by $q \in Q$, if $t$ is simply reduced and at each path to a leaf, $q$ is either the last decision on the path, or $q$ does not occur at all on it.

(iv) A term, $t$, in a decision theory $D = (Q, E)$, is said to be *decision factored* by $q \in Q$, if the sub-term at every occurrence of $q$ is semi-essentially simple.

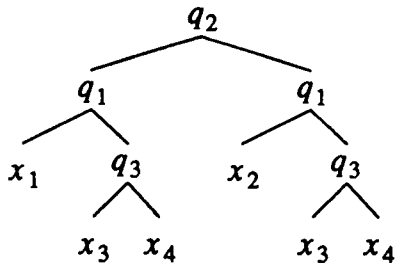
The procedure for decision delaying a simply reduced term, $t$, by a decision $q$, is the following: at every occurrence of $q$ in $t$, use the distribution identity, [D.2], to move any decision occurring after $q$ in the term, to the root of the sub-term in which $q$ appears. This process is repeated until $q$ is the last decision taken on every path on which $q$ occurs.

**Example 2.20**

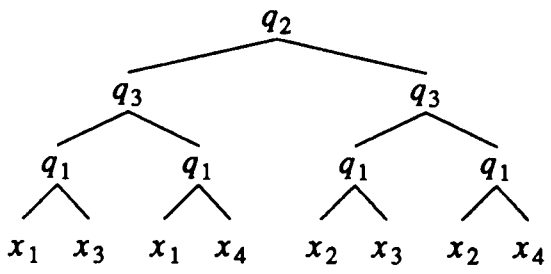
Consider the following decision tree:



To decision delay the tree by $q_1$, one can distribute $q_2$ in front of $q_1$ to produce,




Finally, distributing $q_3$ in front of $q_1$ at each subtree results in




which is decision delayed by $q_1$. ■

Using the definition of decision delayed trees, the following proposition can be used to determine whether a term has either an essential or a redundant occurrence of a decision.

**Proposition 2.21**

If a term $t$, decision equivalent to $t'$, is decision delayed by $q$, then

(i) $q$ is essential in $t'$, if and only if, it is semi-essential in $t$;

(ii) $q$ is said to have a redundant occurrence in $t'$, if and only if, $q$ does not occur on any of the paths in $t$.

**Proof.**

The proof to both of these statements can be found in [8]. ■

Similarly, the procedure to decision factor a term $t$ by a decision $q$ can be described as follows: at every occurrence of $q$ in $t$, use the transposition law to move $q$ past all the semi-

essential decisions in the term. This process is repeated until the subtrees after $q$ are semi-simple or leaves. Should all the leaves after an occurrence of $q$ be the same, then the idempotence rewriting rule can be used to eliminate $q$ from the tree and thus reduce the term. This provides a way for determining when a term is optimal under a reasonable cost criteria as shall be described in the following section.

## 2.6 The Basic Reduction Algorithm

A close look at the properties [RP.1]-[RP.4], reveals that if a term is found to be reducible, then one can improve it by applying idempotence to one of its transposition equivalent forms and testing whether the resulting term is irreducible.

As shown in [8], for repeat reduced terms, if a decision can be eliminated by idempotence at the leaves in one decision factored term, then it can be eliminated in every transposition equivalent decision factored term.

**Proposition 2.22**

> A simply reduced tree, $t$, in a decision theory $D = (Q, E)$, is irreducible, if and only if, for every decision $q$ in $t$, $t$ decision factored by $q$ can not be idempotent reduced.

The detailed description of an algorithm for reducing a discrete decision tree to an irreducible was presented in [12]. Although this algorithm is limited to discrete decision theories without identities, it introduces the basic prescription for eliminating unnecessary decisions from a decision tree. The reduction algorithm assumes that the original form of the decision tree is almost optimal for a given application and it will only improve the tree to make it irreducible.

**Algorithm 2.23 [Cockett's Reduction Algorithm]**

REDUCE ($t$ ):

**if** $t = q\langle t_1 \text{ ' } \ldots \text{ ' } t_n\rangle$

**then begin**

$r_1 \leftarrow$ REDUCE ($t_1$); ... ; $r_n \leftarrow$ REDUCE ($t_n$);

$r' \leftarrow$ DECISION FACTOR ($q\langle r_1 \text{ ' } \ldots \text{ ' } r_n\rangle$);

**return** IDEMPOTENT REDUCE ($r'$);

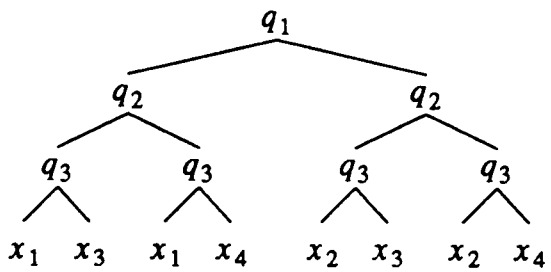
**end**

**elseif** $t = x$ **then return** $t$ ;

The following example demonstrates how the reduction algorithm obtains an irreducible form of a decision tree.

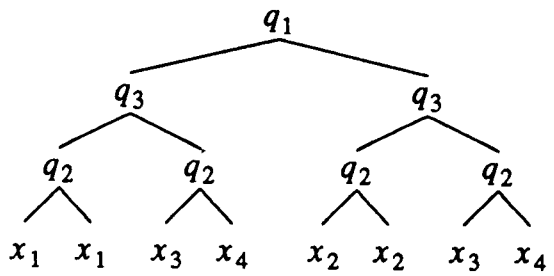
**Example 2.24**

Consider the problem of optimizing the following decision tree involving the decisions $q_1$, $q_2$ and $q_3$:

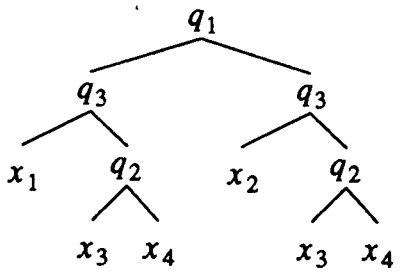



The reduction algorithm can be used to obtain an irreducible form decision equivalent to this term. First, both subtrees after $q_1$ are reduced. Since $q_3$ is essential in the term, all the subtrees after $q_2$ are irreducible.

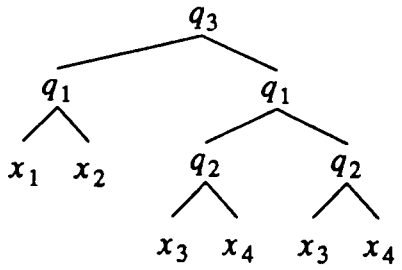
Then, $q_2$ is decision factored past $q_3$ to obtain:



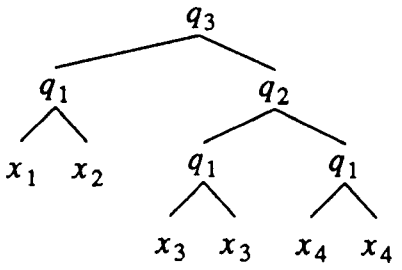

Idempotence then yields the term,



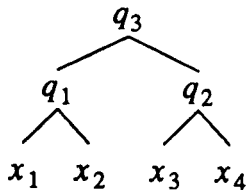

in which now the subtrees after $q_1$ are reduced. Then, $q_1$ is decision factored past $q_3$ to generate the term:




Then, $q_1$ is decision factored past $q_2$ in the subtree on the right, resulting in:




From which, idempotence yields the tree




which is obviously irreducible since the term is simple, that the root at each subterm is an essential decision. ■

As it was shown in [12], the complexity of the basic reduction algorithm is $O(N^3 \times T)$ where $N$ is the number of decisions and $T$ is the number of nodes in the tree. Furthermore, the algorithm requires only the storage to hold any decision factored version of the decision tree. The technique does not suffer from the prohibitive expenses associated with earlier attempts (e.g. dynamic programming and branch-and-bound methods). In addition, the syntactic nature of the optimization makes it unnecessary to have a precise cost criterion before applying the algorithm. In most applications, such a criterion is simply not available.

If the original decision tree is designed heuristically, or is generated from some logical specification (e.g. from production rules by some kind of chaining mechanism), then there is no guarantee that the form of the tree will be optimal [13]. Thus, it is very desirable to develop an algorithm that can find all the irreducible forms of a decision tree. The optimal decision tree for any reasonable cost criterion is an irreducible, thus, one approach to finding the optimal solution is to get all the irreducible forms of the tree first and then determine which irreducible form is best suited for the particular application.

Given a decision tree, there exist a large collection of trees that are semantically equivalent to it. A very much smaller number of these are irreducible. Therefore, it is highly desirable to have an optimization algorithm which provides only these for further consideration.

# CHAPTER 3

## FINDING ALL THE IRREDUCIBLES

The design of decision trees by heuristic methods is a very labor intensive task since trees become unmanageable very quickly. Suppose, for instance, that an initial decision tree representation for the problem is obtained, and later it is realized that it is more sensible to make a different decision at some node. This precipitates the redesign of the subtree at the node that needs modification. The problem is that when redesigning a portion of a tree, it is a non-trivial task to ensure that the meaning of the subtrees affected is not subtly changed. Maintenance of decision trees is, in general, a very difficult task.

Decision trees can also be generated from some logical specification of the problem (e.g. production rules) by simulating some inference engine (i.e. using some chaining mechanism) [14]. Although the maintenance problem is much more manageable in this case, there is no guarantee that the form of the resulting decision tree will be efficient [13]. Thus, the desire to automate the generation of optimal decision trees arises.

As discussed already, the use of traditional optimization techniques usually requires a very detailed knowledge of the specific cost criterion which characterizes the optimality of a particular solution to the problem. Unfortunately, for most decision-making processes (e.g. pattern recognition), that information is simply not available. A classical example is the problem of designing a decision tree classifier for reading license plate numbers from digitized and segmented images of automobiles. In the U.S.A. in particular, license plate numbers are generated at random, even though several states allow for personalized license plates. In fact, different states use different fonts, and even these change from year to year.

Therefore, in general, the frequency of occurrence of a particular letter in a specific font is not readily available.

As a result of this research effort, a set of algorithms for generating all the irreducible forms of a decision tree were developed. The purpose is to allow a system designer to examine the significant alternative forms which can be used to implement the solution of a problem. The system designer may elect to study all the alternatives before making a selection or, more probably, inspect several alternatives until an acceptable form is produced by the algorithm.

For a given decision tree, there exists a large collection of trees all of which are semantically equivalent. Fortunately, only a small number of these are irreducible. An algorithm which provides only these for further consideration by the system designer is introduced in this chapter.

One of the problems encountered when trying to generate the irreducible forms of a given tree is that one may also obtain the transposition variants of each irreducible at the same time. Since transposition equivalent terms are computationally equivalent, it makes sense to develop a planning strategy so that the search does not lead to irreducible forms which are transposition variants of already obtained forms. The procedure, which is described in this chapter, gives a general method for obtaining only one representative for each transposition equivalent class when generating all the irreducible forms of a decision tree. This technique is one of the major contributions provided by this investigation.

The algorithm can be divided into two parts: a reduction stage and a separation planning stage. The reduction stage is responsible for determining whether a decision tree is irreducible. This portion of the algorithm is based on the basic reduction algorithm discussed in the previous chapter.

The separation planning stage, on the other hand, guarantees that no two trivially equivalent irreducible forms are generated, and therefore reduces the number of possible candidates to be considered. It has the following features:

(i) Given a set of decision equivalent trees, the algorithm uses *separator plans* to partition the set into transposition equivalence classes.

(ii) The use of separator plans guarantees that only one representative tree from each transposition equivalence class is generated.

(iii) The separator plans are created dynamically, and the order in which they are generated is determined by the structure of the initial tree.

The reduction algorithm was already discussed in the previous chapter. The separation planning technique is introduced next.

The chapter is organized in two sections. The first section introduces a planning strategy which makes use of the idea of separators for generating all the optimal forms of a decision tree; it also develops some of the terminology used in later chapters. The second section introduces the algorithm for obtaining all the irreducible forms of a tree, along with some examples.

## 3.1 Separation Planning

Transposition equivalent trees are considered to be identical in terms of computational efficiency. This means that once one representative form a transposition equivalence class is reported by the optimization algorithm, there is no need to report any other tree in that class.

The planning technique introduced in this section makes use of some properties of transposition equivalent trees to guarantee that no two of these trees are reported. If two

trees are transposition equivalent, the decisions involved must have the same semi-essential and semi-redundant properties. The planning strategy makes use of this to ensure that once all the trees with a certain semi-essential decision have been reported, then the remaining decision trees will have that decision semi-redundant. The separation planning algorithm generates semi-essential and semi-redundant plans in an incremental fashion, driven by the structure of the input tree. Each time a plan is extended, it partitions the set of decision equivalent trees into smaller transposition invariant classes until all the trees in each class are transposition equivalent. Finally, a plan is quickly abandoned whenever the class it generates does not contain any irreducibles.

### 3.1.1 Separators

The basic component of a separator plan is called a separator. Separators provide a form of "shape" description for decision trees in terms of their semi-essential and semi-redundant decisions (recall last chapter's discussion on semi-essential and semi-redundant decisions).

**Definition 3.1**

Given a decision theory, $D = (Q, E)$, then a *separator* is defined recursively as follows:

(i) Any pair $(s_e, s_r)$, where $s_e \subseteq Q$, $s_r \subseteq Q$, and $s_e \cap s_r = \varnothing$, called a *primitive separator*, is a separator.

(ii) If $S_1, ..., S_n$ are separators and $q$ is a decision in $Q$ with arity $n$, such that $q$ does not occur in any of the separators $S_1, ..., S_n$, then

$$q\ \langle S_1 \text{'} ... \text{'} S_n \rangle$$

is also a separator.

A term $t$, in $D$, is said to *satisfy* the primitive separator $s = (s_e, s_r)$, if every decision in $s_e$ is semi-essential in $t$ and every decision in $s_r$ is semi-redundant in $t$. Similarly, a term, $t$, is said to satisfy the separator $S \equiv t_S\langle s_1'...'s_n\rangle$, where $s_1, ..., s_n$ are primitive separators, if $t$ is transposition equivalent to a decision tree $t_S\langle t_1'...'t_n\rangle$, where $t_i$ satisfies the primitive separator $s_i$, for $i = 1, ...,n$.

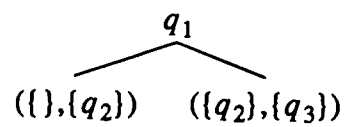
The following example illustrates these concepts.

**Example 3.2**

Given the decision theory, $D = (\{q_1, q_2, q_3\}, \{\})$, then the pair

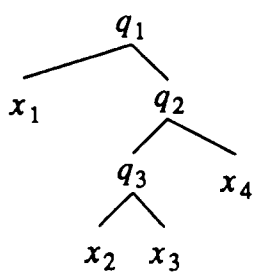
$$s = (\{q_1\}, \{q_2, q_3\})$$

is an example of a primitive separator, while

$S =$ 


is an example of a separator.

Now consider the tree, $t$, in $D$, where

$t =$ 


Since $q_1$ is semi-essential in $t$, while $q_2$ and $q_3$ are semi-redundant, then $t$ satisfies the primitive separator $s$. Furthermore, $t$ also satisfies the separator $S$, since $q_2$ is semi-redundant in the first argument of $q_1$, while $q_2$ is semi-essential and $q_3$ is semi-redundant in the second. ■

A crucial observation at this point is that transposition equivalent trees always satisfy the same separators. In other words, a separator can not be used to differentiate between two transposition equivalent trees.

**Lemma 3.3**

Given two repeat reduced trees, $t_1$ and $t_2$, such that $t_1 =_t t_2$, and a separator, $S$, then $t_1$ satisfies $S$, if and only if, $t_2$ also satisfies $S$.

**Proof.**

Let $S \equiv t_S\langle s_1 ' \ldots ' s_n\rangle$. If $t_1$ satisfies $S$, then there exists a term $t^*$, transposition equivalent to $t_1$, such that $t^* \equiv t_S\langle t^*_1 ' \ldots ' t^*_n\rangle$ where each $t^*_i$ satisfies the primitive separator $s_i$. Now, since $t_2$ is transposition equivalent to $t_1$, then it is also transposition equivalent to $t^*$, which means that $t_2$ also satisfies $S$. ■

If $T$ is any set of repeat reduced decision expressions, then, given a separator, $S$, the set $T_S$, $T_S \subseteq T$, is the set of $t \in T$ which satisfies $S$. An order relation, denoted by $>_s$, can be established on the set of separators. This order relation is generated by the following inequalities:

**[R.1]** (Primitive separator refinement)

$$(s_{e_1}, s_{r_1}) >_s (s_{e_2}, s_{r_2}), \text{ if } s_{e_1} \supseteq s_{e_2}, s_{r_1} \supseteq s_{r_2}$$

**[R.2]** (Primitive separator expansion)

$$q\langle (s_{e_1}, s_{r_1}) ' \ldots ' (s_{e_n}, s_{r_n})\rangle >_s (\{q\} \cup (\cap^n_{i=1} s_{e_i}), \cup^n_{j=1} s_{r_j})$$

**[R.3]** (Monotonic)

$$\text{If } S_1 >_s S_2, \text{ then } q\langle \ldots ' S_1 ' \ldots\rangle >_s q\langle \ldots ' S_2 ' \ldots\rangle$$

The expression $S_1 >_s S_2$ is read: "separator $S_1$ is more refined than separator $S_2$." Two important results about the effect of the order relation $>_s$ on the set of separators should be noted here. First, it constitutes a partial order relation, that is, it is anti-symmetric up to set equivalence. Second, if the decision trees which satisfy two separators are disjoint, then the sets of trees satisfying any subsequent refinement of these separators will also be disjoint.

**Lemma 3.4**

(i) The relation $\geq_s$ constitutes a partial order on the set of separators.

(ii) If $S_1 \geq_s S_2$, then $T_{S_1} \subseteq T_{S_2}$.

**Proof.**

(i) The transitive reflexive closure of [R.1]-[R.3] is a transitive reflexive relation.

To show that $\geq_s$ is an anti-symmetric relation, one must show that if $S_1 \geq_s S_2$ and $S_2 \geq_s S_1$, then $S_1 = S_2$. If both $S_1$ and $S_2$ are primitive separators, that is, $S_1 \equiv (s_{e_1}, s_{r_1})$ and $S_2 \equiv (s_{e_2}, s_{r_2})$, then it follows that $S_1 \geq_s S_2$ implies both $s_{e_1} \supseteq s_{e_2}$ and $s_{r_1} \supseteq s_{r_2}$. Conversely, $S_2 \geq_s S_1$ implies that $s_{e_1} \subseteq s_{e_2}$ and $s_{r_1} \subseteq s_{r_2}$. Therefore, $s_{e_1} = s_{e_2}$ and $s_{r_1} = s_{r_2}$, that is, $S_1 = S_2$.

Now, let $S_1$ and $S_2$ be two separators such that $S_1 \equiv t_{S_1}\langle s_{11}'...'s_{1n}\rangle$ and $S_2 \equiv t_{S_2}\langle s_{21}'...'s_{2m}\rangle$. For both $S_1 \geq_s S_2$ and $S_2 \geq_s S_1$ to hold, $t_{S_1} \equiv t_{S_2}$ must be true, which means that $n=m$, and $s_{1j} \geq_s s_{2j}$ and $s_{2j} \geq_s s_{1j}$ for $1 \leq j \leq n$. Since it has already been shown that $s_{1j} \geq_s s_{2j}$ and $s_{2j} \geq_s s_{1j}$ imply that $s_{1j} = s_{2j}$, then it can be concluded that $S_1 = S_2$.

(ii) To show that if $S_1 \geq_s S_2$ then $T_{S_1} \subseteq T_{S_2}$, it suffices to show that this holds for the generating inequalities [R.1]-[R.3].

If $t$ satisfies $S_1 \equiv (s_{e_1}, s_{r_1})$ and $s_{e_1} \supseteq s_{e_2}$ and $s_{r_1} \supseteq s_{r_2}$, then $t$ also satisfies $S_2 \equiv (s_{e_2}, s_{r_2})$, thus [R.1] is satisfied.

If $t$ satisfies $S_1 \equiv q\langle (s_{e_1}, s_{r_1})' \ldots ' (s_{e_n}, s_{r_n}) \rangle$, then $t =_t q\langle t_1' \ldots ' t_n \rangle$, where $t_i$ satisfies $(s_{e_i}, s_{r_i})$ for $1 \leq i \leq n$, so that $t$ also satisfies the primitive separator

$$S_2 \equiv (\{q\} \cup (\cap_{i=1}^{n} s_{e_i}), \cup_{j=1}^{n} s_{r_j}),$$

thus [R.2] is satisfied.

Suppose $S_1 \geq_s S_2$ and $T_{S_1} \subseteq T_{S_2}$. If $t$ satisfies $q\langle \ldots ' S_1' \ldots \rangle$, then $t =_t q\langle \ldots ' t_i' \ldots \rangle$, where $t_i$ satisfies $S_1$. Since $t_i$ satisfies $S_1$ it must satisfy $S_2$, therefore $t$ satisfies $q\langle \ldots ' S_2' \ldots \rangle$, so that [R.3] is satisfied. ■

**Definition 3.5**

Two separators, $S_1$ and $S_2$, are *disjoint*, if and only if, any repeat reduced decision tree, $t$, satisfies at most one. In other words, for every class $T$, $T_{S_1} \cap T_{S_2} = \varnothing$.

**Corollary 3.6**

Let $S_1$ and $S_2$ be two disjoint separators. If $S^*_1 \geq_s S_1$ and $S^*_2 \geq_s S_2$, then the separators $S^*_1$ and $S^*_2$ are also disjoint.

**Proof.**

By Lemma 3.4, $T_{S^*_1} \subseteq T_{S_1}$ and $T_{S^*_2} \subseteq T_{S_2}$, since $S^*_1 \geq_s S_1$ and $S^*_2 \geq_s S_2$, respectively. Hence, since $T_{S_1} \cap T_{S_2} = \varnothing$, then it follows that $T_{S^*_1} \cap T_{S^*_2} = \varnothing$. ■

### 3.1.2 Separator Plans

Given the notion of a separator, it is then desirable to be able to derive from it a set of more refined separators. A set of rewriting rules for generating more refined separators, called separator plans, is introduced next.

Separator plans are regarded as rewriting rules for deriving, from a given separator, a set of more refined ones. The separators in the set so generated by one of these rules must satisfy the following conditions: each must be a refinement on the original separator; they must all be mutually disjoint; and, they should cover the same set of trees satisfying the original separator.

**Definition 3.7**

(i) In a decision theory, $D = (Q, E)$, a *separation class*, $T$, is a set of repeat reduced trees in $D$, such that if $t_1 \in T$ and $t_1 =_t t_2$, then $t_2 \in T$, i.e., $T$ includes all the transposition variants.

(ii) If $S$ is a separator, and $P_S = \{S_1, ..., S_n\}$ is a set of separators referred to as a *plan for* $S$, then the rewriting rule $S \rightrightarrows P_S$, is a *separator plan*, if and only if:

**[P.1]** (Refinement Property)

$S_i \succ_s S$, for $1 \leq i \leq n$.

**[P.2]** (Disjointedness Property)

If $t$ satisfies $S$, then $t$ satisfies at most one of the separators $S_1, ..., S_n$ ($S_1, ..., S_n$ are disjoint).

**[P.3]** (Covering Property)

$\cup_{i=1}^{n} T_{S_i} = T_S$, i.e., for any separation class, $T$, if there exists a $t \in T$, that satisfies $S$, then there exists an $S_i \in P_S$ such that $t$ also satisfies $S_i$.

(iii) The separator plan $S_1 \rightrightarrows P_{S_1}$ is *more refined than* $S_2 \rightrightarrows P_{S_2}$, if for all the separators $S^*_1 \in P_{S_1}$, there exists a separator $S^*_2 \in P_{S_2}$, such that $S^*_1 \geq_s S^*_2$.

The following example illustrates how separator plans are constructed.

**Example 3.8**

The rewriting rule $s \rightrightarrows P_s$, where $s = (\{q_1\}, \{q_2, q_3\})$, and $P_s = \{S_1, S_2, S_3, S_4\}$ is and example of a separator plan, where

$S_1 =$ $q_1$ over $(\{\}, \{q_2, q_3\})$ and $(\{\}, \{\})$

$S_2 =$ $q_1$ over $(\{q_3\}, \{q_2\})$ and $(\{\}, \{q_3\})$

$S_3 =$ $q_1$ over $(\{q_2\}, \{q_3\})$ and $(\{\}, \{q_2\})$

$S_4 =$ $q_1$ over $(\{q_2, q_3\}, \{\})$ and $(\{\}, \{q_2, q_3\})$

To show this, one only has to verify that $s \rightrightarrows P_s$ satisfies [P.1]-[P.3].

Using [R.2], it is easy to show that $S_i >_s s$ for $1 \leq i \leq 4$. Hence, [P.1] is satisfied.

If $t$ satisfies $S_1$ then is must have both $q_2$ and $q_3$ semi-redundant in the first argument of $q_1$, therefore it can not satisfy any of the other separators in $P_s$, since they all require that at least one of $q_2$ or $q_3$ be semi-essential in that

argument. Similar arguments can be used to show that all the separators in $P_s$ are disjoint. Hence, [P.2] is satisfied.

If $t$ satisfies the primitive separator $(\{q_1\},\{q_2,q_3\})$, then $t$ is transposition equivalent to a tree

$$t' = \begin{array}{c} q_1 \\ \diagup \quad \diagdown \\ t_1 \quad t_2 \end{array}$$

where $q_2$ and $q_3$ each are semi-redundant in at least one subtree $t_1$, $t_2$. If both $q_2$ and $q_3$ are semi-redundant in $t_1$, then $t$ satisfies $S_1$. If $q_2$ is semi-redundant and $q_3$ is semi-essential in $t_1$, then $t$ satisfies $S_2$. If $q_3$ is semi-redundant and $q_2$ is semi-essential in $t_1$, then $t$ satisfies $S_3$. If neither $q_2$ or $q_3$ is semi-redundant in $t_1$, then $t$ satisfies $S_4$. Hence, [P.3] is satisfied. ■

The separator plans can be used as *replacement rules* to obtain more refined separator plans from old plans:

**Lemma 3.9**

If $S \mapsto P_S$, where $P_S = \{S_1, ..., S_n\}$, and $S_i \mapsto P_{S_i}$, where $S_i \in P_S$ and $P_{S_i} = \{S_{i1}, ..., S_{im}\}$, are separator plans, then $S \mapsto P'_S$, where $P'_S = \{S_1, ..., S_{i-1}, S_{i1}, ..., S_{im}, S_{i+1}, ..., S_n\}$ is a more refined separator plan.

**Proof.**

Firstly, $S \mapsto P'_S$ is a separator plan. To prove this, one only has to show that $S \mapsto P'_S$ satisfies [P.1]-[P.3].

Given that $S \mapsto P_S$ and $S_i \mapsto P_{S_i}$, are separator plans, where $S_i \in P_S$, then it follows that:

- For any $S_j \in P_S$, then $S_j >_s S$, and for any $S_{ij} \in P_{S_i}$, then $S_{ij} >_s S_i >_s S$. Thus, [P.1] is satisfied.
- If $S_j$, $S_k \in P_S$, then $S_j$ and $S_k$ are disjoint. Similarly, if $S_{ij}$, $S_{ik} \in P_{S_i}$, then $S_{ij}$ and $S_{ik}$ are disjoint. If $S_k \in P_S$ and $P_{ij} \in P_{S_i}$, then, as $S_k$ is disjoint from $S_i$, and $S_k \geq_s S_k$, $S_{ij} \geq_s S_i$, then by Corollary 3.6 it follows that $S_k$ is disjoint from $S_{ij}$. Thus, [P.2] is satisfied.
- Since $\cup_{j=1}^{n} T_{S_j} = T_S$, and $\cup_{j=1}^{m} T_{S_{ij}} = T_{S_i}$, then it follows that $\cup_{\forall S^*_j \in P'_S} T_{S^*_j} = T_S$. Thus, [P.3] is satisfied.

Finally, to show that $S \rightrightarrows P'_S$ is more refined than $S \rightrightarrows P_S$, note that any separator, $S_k$, in $P'_S$ is also either in $P_S$ or in $P_{S_i}$. If $S_k$ is in $P_S$, then $S_k \geq_s S_k \in P_S$, while if $S_k$ is in $P_{S_i}$, then $S_k \geq_s S_i \in P_S$. ■

The previous result demonstrates how new plans can be generated by replacement. Thus, given a collection of plans, this gives a method for generating new plans. There is, however, another rather important technique for generating new plans from simpler plans. Observe that a separator may be refined by refining one of the primitive plans at its leaves. Thus, if there is a separator plan for a primitive separator at a leaf, then a plan for the overall separator can be generated by replacing the leaf with the separators in the plan for the primitive separator.

The form taken by a plan for a primitive separator depends on the decisions that are encountered on the path to the leaf where the primitive separator occurs.

**Definition 3.10**

Let $s = (s_e, s_r)$ be a primitive separator that occurs on some leaf of a separator, $S$, in a decision theory $D = (Q, E)$, such that the decisions

$q_1, ..., q_r$ are encountered on the path to $s$. Then, a *leaf plan for s* given $\{q_1, ..., q_r\}$ is a separator plan of the form

$$s \ /\{q_1, ..., q_r\} \Rightarrow P_s$$

where $P_s = \{S_1, ..., S_n\}$ is a set of separators in the decision theory $D' = (Q - \{q_1, ..., q_r\}, E)$, such that no decision in $\{q_1, ..., q_r\}$ occurs in any of the separators in $P_s$.

**Example 3.11**

Let $D = (\{q_0, q_1, q_2, q_3\}, \{\})$ be a decision theory.

If $s = (\{q_1\}, \{q_2, q_3\})$ is a primitive separator occurring at a leaf of a separator $S$ in $D$, such that $q_0$ is the only decision encountered in the path to $s$, then an example of a leaf plan for $(\{q_1\}, \{q_2, q_3\})$, given $\{q_0\}$ would be the following rewriting:

$$(\{q_1\}, \{q_2, q_3\}) /\{q_0\} \Rightarrow \{S_1, S_2, S_3, S_4\}$$

where $S_1, S_2, S_3, S_4$ are those of Example 3.8. ■

**Lemma 3.12**

Given a separator, $S$, in a decision theory, $D = (Q, E)$, such that $S \equiv t_S\langle s_1'...'s_n\rangle$, where $s_i$, $1 \le i \le n$, is a primitive separator at the leaf of $S$ and $Q_i$ is the set of decisions on the path to that leaf, then, if $s_i /Q_i \Rightarrow \{S_1, ..., S_m\}$ is a leaf plan, and $P_S = \{S^i_1, ..., S^i_m\}$ is a set of separators in $D$ such that

$$S^i_j \equiv t_S \langle s_1 ' ... ' s_{i-1} ' S_j ' s_{i+1} ' ... ' s_n\rangle, \quad 1 \le j \le m,$$

then, $S \Rightarrow P_S$ is a separator plan.

**Proof.**

To prove that $S \mapsto P_S$ is a separator plan, it is necessary to show that the elements of $P_S$ are indeed separators that satisfy [P.1]-[P.3].

Let $T$ be a separation class in $D$. Any $t \in T_S$ is of the form $t =_t t_S\langle t_1' \ldots ' t_n\rangle$, where $t_i$ satisfies $s_i$ in $D^i$, for $1 \le i \le n$, and where $D^i = (Q - Q_i, E)$ is the decision theory associated with the ith leaf of $S$.

Since $s_i / Q_i \mapsto \{S_1, \ldots, S_m\}$ is a separation plan in $D^i$, then $S_j >_s s_i$ for $1 \le j \le m$, therefore, according to [R.3], it follows that

$$t_S\langle \ldots ' s_{i-1} ' S_j ' s_{i+1} ' \ldots\rangle >_s t_S\langle \ldots ' s_{i-1} ' s_i ' s_{i+1} ' \ldots\rangle,$$

i.e., $S^i_j >_s S$, for $1 \le j \le m$. Thus, [P.1] is satisfied.

Let $t$ be a decision tree in $D$ satisfying $S^i_j \in P_S$, $1 \le j \le m$. This implies that $t =_t t_S\langle t_1' \ldots ' t_i' \ldots ' t_n\rangle$, where $t_i$ satisfies $S_j$. Consider now any other separator in $S^i_k \in P_S$, $1 \le k \le m$, $k \ne j$, where

$$S^i_k \equiv t_S\langle \ldots ' s_{i-1} ' S_k ' s_{i+1} ' \ldots\rangle.$$

Since $s_i / Q_i \mapsto \{S_1, \ldots, S_m\}$ is a separator plan in $D^i$, then any tree, $t_i$ in $D^i$, that satisfies $S_j$, does not satisfy $S_k$, thus $t$ cannot satisfy the separator $S^i_k$. Therefore, [P.2] is satisfied.

Consider any tree $t \in T_S$. Since $t$ satisfies $S$, then $t =_t t_S\langle \ldots ' t_i' \ldots\rangle$, where $t_i$ satisfies $s_i$ in $D^i$, for some $1 \le i \le n$. As $s_i / Q_i \mapsto \{S_1, \ldots, S_m\}$ is a leaf plan, then there exists a separator $S_j$, $1 \le j \le m$, such that $t_i$ satisfies $S_j$. Therefore, $t$ satisfies $S^i_j \in P_S$. Thus, [P.3] is satisfied.

Finally, to show that the separators in $P_S$ are repeat reduced, observe that as $S$ is repeat reduced, so is $t_S$, and no decision on the path to $s_i$ occurs on $s_i$

itself for $1 \leq i \leq n$. Now, any separator $S^*$, resulting from the application of a leaf plan to $S$, will have the form $S^* \equiv t_S\langle S^*_1 '\ldots' S^*_n\rangle$, where $S^*_i \geq_s s_i$, for $1 \leq i \leq n$. Since the separator plans generated from a leaf plan for $s_i$ do not involve any decision on the path to $s_i$ in $S$, the resulting separator $S^*$ is also repeat reduced. ■

The leaf plans can be utilized as replacement rules in the following way. Let $S \Mapsto P_S$ be a separator plan, where $P_S=\{S_1, \ldots, S_n\}$. Now, let $S_i \in P_S$, be a separator of the form $S_i \equiv t_S\langle\ldots' s_j '\ldots\rangle$, where the primitive separator $s_j$ is reached after the set of decisions in $Q_j$ are encountered. If there exists a leaf plan of the form $s_j / Q_j \Mapsto \{S^j_1, \ldots, S^j_m\}$, then the separator plan $S \Mapsto P_S$ can be refined into the separator plan $S \Mapsto P'_S$, where $P'_S=\{S_1,\ldots,S_{i-1},S_{i1},\ldots,S_{im},S_{i+1},\ldots,S_n\}$ and $S_{ik} \equiv t_S\langle\ldots ' s_{j-1} ' S^j_k ' s_{j+1} ' \ldots\rangle \geq_s S_i$ for $k=1,\ldots,m$. The result is a more refined plan $P'_S$ for $S$.

The following two results establish a pair of leaf plans that can be used to generate separator plans from separators.

**Lemma 3.13**

Let $D = (Q, E)$ be a decision theory and let the primitive separator $s = (s_e, s_r)$ occur on some leaf of a separator $S$ in $D$, such that, if $Q_s$ is the set of decisions on the path to $s$, then no decision in $Q_s$ occurs in either $s_e$ or $s_r$. Then, the replacement

**[S.1]** (Semi-redundant leaf refinement)

$$(s_e, s_r) / Q_s \Mapsto \{(\{q\} \cup s_e, s_r), (s_e, \{q\} \cup s_r)\}$$
$$\text{where } q \in Q - (Q_s \cup s_e \cup s_r),$$

constitutes a leaf plan for $s$.

**Proof.**

To prove this lemma, one needs to show that the separation plan $s \rightleftharpoons P_s$ where $P_s = \{(\{q\} \cup s_e, s_r), (s_e, \{q\} \cup s_r)\}$, satisfies [P.1]-[P.3] in the decision theory D' = $(Q - Q_s, E)$.

Consider the separators $s_1 = (\{q\} \cup s_e, s_r)$ and $s_2 = (s_e, \{q\} \cup s_r)$ in $P_s$. According to [R.1], since $\{q\} \cup s_e \supseteq s_e$ and $s_r \supseteq s_r$, then $s_1 \succ_s s$. Similarly, since $s_e \supseteq s_e$ and $\{q\} \cup s_r \supseteq s_r$, then $s_2 \succ_s s$. Thus, $P_s$ satisfies [P.1].

Let $t$ be a decision tree in D'. If $t$ satisfies $s_1$, then $q$ is semi-essential in $t$. Hence, $t$ cannot satisfy $s_2$, which requires $q$ to be semi-redundant in $t$. Conversely, if $t$ satisfies $s_2$, then $q$ is semi-redundant in $t$. Hence, $t$ cannot satisfy $s_1$, which requires $q$ to be semi-essential in $t$. Therefore, $P_s$ satisfies [P.2].

Let $T$ be a separation class in $D'$. If $t \in T$ such that $t$ satisfies $s=(s_e, s_r)$, i.e., $t \in T_s$, then either $t$ is a variable, or $t$ is a decision tree involving at least one decision. If $t$ is a variable, then $s_e$ must be empty as there are no decisions in $t$, and $t$ must then satisfy the separator $s_2$. If $t$ is a decision tree involving at least one decision, then let $s_e^{max}$ be the set containing all the semi-essential decisions in $t$. If $q \in s_e^{max}$, then $t$ satisfies the separator $s_1$, otherwise, $t$ satisfies the separator $s_2$. Thus $P_s$ satisfies [P.3]. ■

**Definition 3.14**

(i) Given a set $d$ of decisions, then a *disjoint cover for d of order n* is an $n$-tuple of the form $(d_1,\ldots,d_n)$, where $\cup_{i=1}^{n} d_i = d$ and $d_i \cap d_j = \varnothing$, for $i \neq j$.

(ii) Given a decision theory $D = (Q, E)$, with $q \in Q$, then a plan $P$ is *uniform in q*, if $q$ is the root of every separator $S$ in $P$.

(iii) Given the primitive separator $s = (s_e, s_r)$, where $s_e \neq \{\}$, then the leaf plan $s\ /Q_s \Rightarrow P_s$ is a *disjoint redundancy plan* if given that $q \in s_e$, and $P_s$ is uniform in $q$, then every separator $S_i \in P_s$ is of the form

$$S_i = q \langle (s_{e_{i1}}, s_{r_{i1}}) \text{ ' } \ldots \text{ ' } (s_{e_{in}}, s_{r_{in}}) \rangle$$

where $(s_{r_{i1}}, \ldots, s_{r_{in}})$ is a *disjoint cover* for $s_r$ of order $n$ and $\{q\} \cup (\bigcap_{j=1}^{n} s_{ij}) = s_e$.

**Example 3.15**

Example 3.8 is also an example of a disjoint redundancy plan. ■

The fact that the semi-redundant requirements at the leaves of the separators in a disjoint redundancy plan are disjoint covers of the original semi-redundant requirement means that their semi-essential requirements must also be minimal. Specifying more semi-essentiality than needed at a leaf would destroy the covering property of the plan. On the other hand, specifying less semi-essentiality than needed at a leaf would destroy the disjointedness of the plan. In fact, given a primitive separator $s = (s_e, s_r)$, where $s_e \neq \{\}$, then there is only one disjoint redundancy plan for $s$ uniform in $q \in s_e$.

The following algorithm describes a method for generating a disjoint cover. Given a primitive separator $(s_e, s_r)$ and a decision $q \in s_e$ of arity $N$, the algorithm can be used to obtain a disjoint cover for the pair $(s_e, s_r)$ of order $N$, $\{(s_{e_1}, s_{r_1}), \ldots, (s_{e_N}, s_{r_N})\}$, from which the disjoint redundancy plan $q \langle (s_{e_1}, s_{r_1}) \text{ ' } \ldots \text{ ' } (s_{e_N}, s_{r_N}) \rangle$ can be generated.

**Algorithm 3.16 [Generating Disjoint Covers]**

DISJOINT COVER $((s_e, s_r), N)$:

**if** $N > 1$

**then begin**

$s_{r_N} \in$ SUBSETS $(s_r)$;

$C_N \in$ DISJOINT COVER $((s_e, s_r - s_{r_N}), N\text{-}1)$;

**return** $\{(s_e \cup (s_r - s_{r_N}), s_{r_N})\} \cup C_N$;

**end**

**else return** $\{(s_e, s_r)\}$;

Note that the non-deterministic procedure SUBSETS always return a different choice of a subset from the input list. Similarly the procedure DISJOINT COVER is non-deterministic as well, returning a different solution every time the algorithm backtracks to that step. The algorithm works by breaking the problem into similar but smaller sub-problems until the terminating condition is met (N = 1).

**Example 3.17**

This example illustrates how the Disjoint Cover algorithm can be used to generate a disjoint redundancy plan for the primitive separator $(\{q_1\}, \{q_2, q_3\})$, where $q_1$ has an arity of 2.

**Step 1:** Get a disjoint cover for $s_r$ based on the initial subset $\{q_2, q_3\}$.

$$s_{r_{11}} = \{q_2, q_3\} \subseteq s_r$$

$$s_{e_{11}} = \{\} = (s_e - \{q_1\}) \cup (s_r - s_{r_{11}})$$

$$s_{r_{12}} = \{\} = s_r - s_{r_{11}}$$

$$s_{e_{12}} = \{\} = (s_e - \{q_1\}) \cup (s_r - [s_{r_{11}} \cup s_{r_{12}}])$$

**Step 2:** Add the following separator to the set $P_s$.

$$S_1 = \begin{array}{c} q_1 \\ \diagup \quad \diagdown \\ (\{\}, \{q_2, q_3\}) \quad (\{\}, \{\}) \end{array}$$

**Step 3:** Get a disjoint cover for $s_r$ based on the initial subset $\{q_2\}$.

$$s_{r_{21}} = \{q_2\} \subseteq s_r$$

$$s_{e_{21}} = \{q_3\} = (s_e - \{q_1\}) \cup (s_r - s_{r_{21}})$$

$$s_{r_{22}} = \{q_3\} = s_r - s_{r_{21}}$$

$$s_{e_{22}} = \{\} = (s_e - \{q_1\}) \cup (s_r - [s_{r_{21}} \cup s_{r_{22}}])$$

**Step 4:** Add the following separator to the set $P_s$.

$$S_2 = \begin{array}{c} q_1 \\ \diagup \quad \diagdown \\ (\{q_3\}, \{q_2\}) \quad (\{\}, \{q_3\}) \end{array}$$

**Step 5:** Get a disjoint cover for $s_r$ based on the initial subset $\{q_3\}$.

$$s_{r_{31}} = \{q_3\} \subseteq s_r$$

$$s_{e_{31}} = \{q_2\} = (s_e - \{q_1\}) \cup (s_r - s_{r_{31}})$$

$$s_{r_{32}} = \{q_2\} = s_r - s_{r_{31}}$$

$$s_{e_{32}} = \{\} = (s_e - \{q_1\}) \cup (s_r - [s_{r_{31}} \cup s_{r_{32}}])$$

**Step 6:** Add the following separator to the set $P_s$.

$$S_3 = \begin{array}{c} q_1 \\ \diagup \quad \diagdown \\ (\{q_2\}, \{q_3\}) \quad (\{\}, \{q_2\}) \end{array}$$

**Step 7:** Get a disjoint cover for $s_r$ based on the initial subset $\{\}$.

$$s_{r_{41}} = \{\} \subseteq s_r$$
$$s_{e_{41}} = \{q_2, q_3\} = (s_e - \{q_1\}) \cup (s_r - s_{r_{41}})$$
$$s_{r_{42}} = \{q_2, q_3\} = s_r - s_{r_{41}}$$
$$s_{e_{42}} = \{\} = (s_e - \{q_1\}) \cup (s_r - [s_{r_{41}} \cup s_{r_{42}}])$$

**Step 8:** Add the following separator to the set $P_s$.

$$S_4 = \quad q_1$$
$$(\{q_2, q_3\}, \{\}) \qquad (\{\}, \{q_2, q_3\})$$

**Step 9:** Stop, as there are no more subsets of $s_r$. ■

Given a primitive separator $(s_e, s_r)$, and a decision $q \in s_e$, then the Disjoint Cover algorithm can be used to obtain a disjoint redundancy plan $P_s = \{S_1, ..., S_M\}$, which is uniform in $q$. Thus, if $S_i \in P_s$, and $q$ has arity $N$, then $S_i$ is of the form

$$S_i \equiv q \langle (s_{e_{i1}}, s_{r_{i1}}) ' ... ' (s_{e_{iN}}, s_{r_{iN}}) \rangle$$

where

$$s_{r_{i1}} \subseteq s_r,$$
$$s_{r_{ij}} \subseteq s_r - \bigcup_{k=1}^{j-1} s_{r_{ik}} \quad \text{for } j = 2, ..., N-1,$$
$$s_{iN} = s_r - \bigcup_{k=1}^{N-1} s_{r_{ik}} \bigcup_{k=1}^{N-1} s_{r_{ik}}, \quad \text{and}$$
$$s_{e_{ij}} = (s_e - \{q\}) \cup (s_r - \bigcup_{k=1}^{j} s_{r_{ik}})$$

Furthermore, since all the disjoint covers of $s_r$ are considered, if $s_r$ has $m$ elements in it, then $P_s$ consists of $N^m$ separators, i.e., $M = N^m$.

**Lemma 3.18**

Let $D = (Q, E)$ be a decision theory and let the primitive separator $s = (s_e, s_r)$, $s_e \neq \varnothing$, occur on some leaf of a separator $S$ in $D$, such that, if $Q_s$ is the set of decisions on the path to $s$, then no decision in $Q_s$ occurs in either $s_e$ or $s_r$. If $q \in s_e$, then the replacement, $s \Mapsto P_s$, given by,

[S.2] (Semi-essential leaf refinement)

$$\begin{aligned}(s_e, s_r)\ /Q_s \Mapsto \{ & q \langle (s_{e_{11}}, s_{r_{11}}) \text{ ' } \ldots \text{ ' } (s_{e_{1n}}, s_{r_{1n}}) \rangle, \\ & q \langle (s_{e_{21}}, s_{r_{21}}) \text{ ' } \ldots \text{ ' } (s_{e_{2n}}, s_{r_{2n}}) \rangle, \\ & \ldots, \\ & q \langle (s_{e_{m1}}, s_{r_{m1}}) \text{ ' } \ldots \text{ ' } (s_{e_{mn}}, s_{r_{mn}}) \rangle \ \},\end{aligned}$$

where $P_s$ is the disjoint redundancy plan for $(s_e, s_r)$ as generated by the Disjoint Cover algorithm, constitutes a leaf plan for $s$.

**Proof.**

To prove this lemma, it is necessary to show that the separation plan $s \Mapsto P_s$ above satisfies [P.1]-[P.3] in the decision theory D' = $(Q - Q_s, E)$.

Let $S_i = q \langle (s_{e_{i1}}, s_{r_{i1}}) \text{ ' } \ldots \text{ ' } (s_{e_{in}}, s_{r_{in}}) \rangle$ be one of the separators in $P_s$, where $1 \leq i \leq m$. Then, according to [R.2],

$$S_i >_s (\{q\} \cup (\cap_{j=1}^{n} s_{e_{ij}}),\ \cup_{j=1}^{n} s_{r_{ij}}).$$

Furthermore, since $(s_{r_{i1}}, \ldots, s_{r_{in}})$ is a disjoint cover for $s_r$ of order n, then $\cup_{j=1}^{n} s_{r_{ij}} \supseteq s_r$, and

$$\begin{aligned}\{q\} \cup (\cap_{j=1}^{n} s_{e_{ij}}) &= \{q\} \cup (\cap_{j=1}^{n} [(s_e - \{q\}) \cup (s_r - \cup_{k=1}^{j} s_{r_{ik}})]) \\ &= \{q\} \cup (s_e - \{q\}) \cup [\cap_{j=1}^{n} (s_r - \cup_{k=1}^{j} s_{r_{ik}})]\end{aligned}$$

$$= s_e \cup [\cap_{j=1}^{n} (s_r - \cup_{k=1}^{j} s_{r_{ik}})]$$

$$= s_e \cup \varnothing$$

$$\supseteq s_e$$

then, according to [R.1], $(\{q\} \cup (\cap_{j=1}^{n} s_{e_{ij}}), \cup_{j=1}^{n} s_{r_{ij}}) \succ_s (s_e, s_r)$. Therefore, $S_i \succ_s (s_e, s_r)$, so $P_s$ satisfies [P.1].

Let $t$ satisfy the separator $S_i = q \langle (s_{e_{i1}}, s_{r_{i1}}) ' \ldots ' (s_{e_{in}}, s_{r_{in}}) \rangle$, therefore, $t =_t q \langle t_1 ' \ldots ' t_n \rangle$ such that $t_k$ satisfies $(s_{e_{ik}}, s_{r_{ik}})$, for $1 \leq k \leq n$. Suppose that $t$ also satisfies the= separator $S_j = q\langle (s_{e_{j1}}, s_{r_{j1}}) ' \ldots ' (s_{e_{jn}}, s_{r_{jn}}) \rangle$, $j \neq i$. Since $(s_{r_{i1}}, \ldots, s_{r_{in}})$ and $(s_{r_{j1}}, \ldots, s_{r_{jn}})$ are two different disjoint covers for $s_r$, then, there exits at least one $l$, such that $s_{r_{il}} \neq s_{r_{jl}}$ and $s_{r_{ik}} = s_{r_{jk}}$ for $k < l$. Since $s_{r_{il}} \neq s_{r_{jl}}$, then, either $s_{r_{il}} - s_{r_{jl}} \neq \varnothing$ or $s_{r_{jl}} - s_{r_{il}} \neq \varnothing$. Without loss of generality, it can be assumed that $q' \in s_{r_{il}} - s_{r_{jl}} \neq \varnothing$. Since $t_l$ satisfies $(s_{e_{il}}, s_{r_{il}})$, then $q'$ must be semi-essential in $t_l$. However, since $q' \in s_{r_{il}}$, $q' \notin s_{r_{jl}}$ and $s_{e_{jl}} = s_e \cup [s_r - \cup_{k=1}^{l} s_{r_{jk}}]$ then $q' \in s_{e_{jl}}$. But $q'$ is semi-redundant in $t_l$, therefore, $t_l$ cannot satisfy the separator $(s_{e_{jl}}, s_{r_{jl}})$. Hence, $P_s$ satisfies [P.2].

Let $T$ be a separation class in $D'$. Now, let $t \in T$ such that $t$ satisfies $s = (s_e, s_r)$, i.e., $t \in T_s$. Since $s_e$ is non-empty, then $t$ is not a variable. Furthermore, since $q \in s_e$, then $t =_t q\langle t_1 ' \ldots ' t_n \rangle$. Let $s_{e_i}^{max}$ and $s_{r_i}^{max}$ be the sets containing all the semi-essential and semi-redundant decisions in $t_i$ respectively. Obviously, $t$ must also satisfy the separator $S^{max} = (\{q\} \cup (\cap_{i=1}^{n} s_{e_i}^{max}, \cup_{j=1}^{n} s_{r_j}^{max})$. Now, since $\{q\} \cup (\cap_{i=1}^{n} s_{e_i}^{max}) \supseteq s_e$ and $\cup_{j=1}^{n} s_{r_j}^{max} \supseteq s_r$, then $S^{max} \succ_s s$. Also, observe that there exists a separator, $S_i$, such that $S_i = q\langle (s_{e_{i1}}, s_{r_{i1}}) ' \ldots ' (s_{e_{in}}, s_{r_{in}}) \rangle$, with $(s_{r_{i1}}, \ldots, s_{r_{in}})$

a disjoint cover for $s_r$ of order n such that $s_{r_{ij}} \subseteq s_{r_j}^{max}$ and $s_{e_{ij}} \subseteq s_{e_j}^{max}$, for $j = 1, ..., n$. Therefore, $t$ satisfies the separator $S_i$, and thus $P_s$ satisfies [P.3]. ■

**Definition 3.19**

(i) A separator plan, $S \rightrightarrows P_S$, is a *leaf refinement*, if $P_S$ is the plan resulting from using Lemma 3.12 at some leaf of $S$ to obtain a set of more refined separators.

(ii) A separator plan, $S \rightrightarrows P_S$, is *irreducible* if and only if, given $S \rightrightarrows P'_S$,

$$P_S >_s P'_S$$

implies that $P'_S = P_S$.

Whenever a new symbol is introduced in the planning, as in the semi-redundant refining rule [S.1], the new symbol is used to guarantee the disjointedness of the plan by forcing it to be semi-essential in one separator and semi-redundant in the other.

On the other hand, when no new symbols are introduced, care must be taken to guarantee that the new plan will be disjoint. The following result shows that any leaf plan of depth one that does not introduce a new symbol must be uniform in some decision.

**Lemma 3.20**

Given a primitive separator $s = (s_e, s_r)$, and a leaf plan of depth one, $s / Q_s \rightrightarrows P_s$, if no new symbols are introduced in the plan, then $P_s$ has to be uniform in some decision $q$.

**Proof.**

Given two separators $S_1, S_2 \in P_s$, such that

$$S_1 \equiv q_1 \langle (s_{e_{11}}, s_{r_{11}}) \text{ ' } \ldots \text{ ' } (s_{e_{1n}}, s_{r_{1n}}) \rangle, \quad \text{and}$$
$$S_2 \equiv q_2 \langle (s_{e_{21}}, s_{r_{21}}) \text{ ' } \ldots \text{ ' } (s_{e_{2m}}, s_{r_{2m}}) \rangle,$$

where $q_1 \neq q_2$ and $q_1, q_2 \in s_e$, then to prove this lemma it suffices to show that $S_1$ and $S_2$ are not necessarily disjoint.

Let $t$ be a decision tree satisfying $S_1$, such that

$$t \equiv q_1 \langle q_2 \langle t_{11} \text{ ' } \ldots \text{ ' } t_{1m} \rangle \text{ ' } \ldots \text{ ' } q_2 \langle t_{n1} \text{ ' } \ldots \text{ ' } t_{nm} \rangle \rangle.$$

Since $t$ satisfies $S_1$, then any of its subtrees $q_2 \langle t_{i1} \text{ ' } \ldots \text{ ' } t_{im} \rangle$, satisfies the separator $(s_{e_{1i}}, s_{r_{1i}})$.

Consider now the following tree,

$$t' \equiv q_2 \langle q_1 \langle t_{11} \text{ ' } \ldots \text{ ' } t_{n1} \rangle \text{ ' } \ldots \text{ ' } q_1 \langle t_{1m} \text{ ' } \ldots \text{ ' } t_{nm} \rangle \rangle.$$

Obviously, $t =_t t'$, so $t'$ also satisfies $S_1$. Furthermore, if every subtree $q_1 \langle t_{1i} \text{ ' } \ldots \text{ ' } t_{ni} \rangle$, satisfies the separator $(s_{e_{2i}}, s_{r_{2i}})$, then $t'$ also satisfies $S_2$.

Hence, $S_1$ and $S_2$ are not disjoint. ■

Next, it is shown that the application of either [S.1] or [S.2] at a leaf of a separator results in an irreducible plan. This is a very desirable result, since it means that by using only two refining leaf plans a minimal plan in the separation ordering is always obtained.

**Lemma 3.21**

If the separator plan, $S \Mapsto P_S$, is a leaf refinement resulting from the application of the semi-redundant leaf refinement [S.1] at some leaf of $S$, then $S \Mapsto P_S$ is irreducible.

**Proof.**

Let $S \rightrightarrows P_S$ be a separator plan, where $S \equiv t_S \langle \ldots 's_i' \ldots \rangle$. Suppose that there exists a plan $P'_S$ for $S$ such that $S \rightrightarrows P_S \ >_s \ S \rightrightarrows P'_S$. Therefore, if $S_j \in P_S$, then there exists a $S^*_j \in P'_S$, such that $S_j \geq_s S^*_j >_s S$.

If $S_j \equiv t_S \langle \ldots 'S_{ij}' \ldots \rangle$ and $S^*_j \equiv t_S \langle \ldots 'S^*_{ij}' \ldots \rangle$, then

$$t_S \langle \ldots 'S_{ij}' \ldots \rangle \geq_s t_S \langle \ldots 'S^*_{ij}' \ldots \rangle >_s t_S \langle \ldots 's_i' \ldots \rangle$$

Hence, $S_{ij} \geq_s S^*_{ij} >_s s_i$ in the decision theory $D^i = (Q - Q_i, E)$, where $Q_i$ is the set of decisions that occur on the path to $s_i$. Let $S \rightrightarrows P_S$ be a semi-redundant leaf refinement of $S$ at some leaf $s_i$. Hence, $s_i$ must be of the form $(s_e, s_r)$, and $S_{ij} \in \{(\{q\} \cup s_e, s_r), (s_e, \{q\} \cup s_r)\}$ where $q \in Q - (Q_i \cup s_e \cup s_r)$.

If $S_{ij} = (\{q\} \cup s_e, s_r)$, then

$$(\{q\} \cup s_e, s_r) \geq_s S^*_{ij} >_s (s_e, s_r)$$

and hence, $S^*_{ij} = (\{q\} \cup s_e, s_r) = S_{ij}$.

Similarly, if $S_{ij} = (s_e, \{q\} \cup s_r)$, then

$$(s_e, \{q\} \cup s_r) \geq_s S^*_{ij} >_s (s_e, s_r)$$

and hence, $S^*_{ij} = (s_e, \{q\} \cup s_r) = S_{ij}$.

Therefore, $S \rightrightarrows P_S$ is irreducible. ■

**Lemma 3.22**

If the separator plan, $S \rightrightarrows P_S$, is a leaf refinement resulting from the application of the semi-essential leaf refinement [S.2] at some leaf of $S$, then $S \rightrightarrows P_S$ is irreducible.

**Proof.**

Let $S \rightrightarrows P_S$ be a separator plan, where $S \equiv t_S \langle \ldots 's_i' \ldots \rangle$. Suppose that there exists a plan $P'_S$ for $S$ such that $S \rightrightarrows P_S \ >_s \ S \rightrightarrows P'_S$. Therefore, if $S_j \in P_S$, then there exists a $S^*_j \in P'_S$, such that $S_j \geq_s S^*_j >_s S$.

If $S_j \equiv t_S \langle \ldots 'S_{ij}' \ldots \rangle$ and $S^*_j \equiv t_S \langle \ldots 'S^*_{ij}' \ldots \rangle$, then

$$t_S \langle \ldots 'S_{ij}' \ldots \rangle \geq_s t_S \langle \ldots 'S^*_{ij}' \ldots \rangle >_s t_S \langle \ldots 's_i' \ldots \rangle$$

Hence, $S_{ij} \geq_s S^*_{ij} >_s s_i$ in the decision theory $D^i = (Q - Q_i, E)$, where $Q_i$ is the set of decisions that occur on the path to $s_i$. Let $S \rightrightarrows P_S$ be a semi-essential leaf refinement of $S$ at some leaf $s_i$. Hence, $s_i$ must be of the form $(s_e, s_r)$, where $s_e \neq \varnothing$, and $S_{ij}$ is of the form

$$S_{ij} \equiv q \langle (s_{e_1}, s_{r_1}) ' \ldots ' (s_{e_n}, s_{r_n}) \rangle$$

where $q \in s_e$, $\{q\} \cup (\bigcap_{k=1}^{n} s_{e_k}) = s_e$ and $(s_{r_1}, \ldots, s_{r_n})$ is a disjoint cover for $s_r$ of order n.

$S^*_{ij}$ can not be a primitive separator since $S_{ij} \geq_s S^*_{ij} >_s (s_e, s_r)$. Furthermore, it cannot be of depth greater than one since $S_{ij} \geq_s S^*_{ij}$ and $S_{ij}$ has depth one. Therefore, $S^*_{ij}$ must have a depth of one.

Since $S^*_{ij}$ is separator of depth one and since no new symbols are introduced in the plan, then, by Lemma 3.20, $P'_S$ must be uniform in $q'$, where $q' \in s_e$. Since $P_S$ is uniform in $q \in s_e$ and $S_{ij} \geq_s S^*_{ij}$, then $q' = q$. Therefore, $S^*_{ij}$ must be of the form

$$S^*_{ij} \equiv q \langle (s'_{e_1}, s'_{r_1}) ' \ldots ' (s'_{e_n}, s'_{r_n}) \rangle$$

where $(s_{e_k}, s_{r_k}) \geq_s (s'_{e_k}, s'_{r_k})$, i.e.,

$$s'_{e_k} \subseteq s_{e_k},\ s'_{r_k} \subseteq s_{r_k}$$

Now, since $(s_{r_1},...,s_{r_n})$ is a disjoint cover for $s_r$, then $s'_{r_k}$ must be the same as $s_{r_k}$ since the union of the $s'_{r_k}$, $1 \leq k \leq n$ must be equal to $s_r$.

Similarly, $s'_{e_k}$ must be equal to $s_{e_k}$ since dropping any semi-essential requirement would destroy the disjointedness of the separators $S'_{ij}$ as established by [S.2] on the $S_{ij}$ 's.

Therefore, $S'_j = S_j$ and hence, $S \rightrightarrows P_S$ is an irreducible separator plan. ■

**Definition 3.23**

(i) A separator, $S$, is said to be *consistent* with respect to a separation class, $T$, if and only if, there exists a $t \in T$, such that $t$ satisfies $S$. If there is no such $t$ in $T$, then $S$ is said to be *inconsistent.*

(ii) A *consistent application* of the leaf refinement rules [S.1]-[S.2] is one that is applied to a consistent separator.

Observe that [P.3] suggests that in order to determine whether a separator plan $S \rightrightarrows P_S$ is consistent, it suffices to show that there exists a $t \in T_S$, since that also means that there exists a separator $S' \in P_S$ such that $t \in T_{S'}$.

**Proposition 3.24**

If $T$ is a separation class, then a consistent separator $S$ is maximal in the separation ordering, if and only if, there does not exist an application of [S.1]-[S.2] which yields a consistent separator plan for $S$.

**Proof.**

($\Leftarrow$) If either of the leaf refinement rules, [S.1] or [S.2], can be consistently applied, then clearly, $S$ is not maximal.

($\Rightarrow$) If $S$ is not maximal, then there exists a separator $S'$, $S' >_s S$, such that $S'$ is consistent. Suppose that $S'$ is obtained from $S$ by one of

the generating relations [R.1] or [R.2] applied at a leaf. Firstly, observe that $S$ is not maximal if any of the primitives at the leaves demand a semi-essential as [S.2] can be applied. Thus, it can be assumed that every leaf primitive separator is of the form $s_i = (\{\}, s_{r_i})$. [S.1] can be consistently applied if $s_i$ is refined using [R.1] or [R.2] where a demand for a semi-essential is introduced. The only remaining case to consider is when only more redundancy is required. Either there is a $t$ which will satisfy an [S.1] refinement or there is no such $t$. In the latter case, the remainder at that leaf of the plan for every $t \in T$ must be a variable. In which case [S.1] can be applied. ■

**Lemma 3.25**

If $S_{max}$ is a maximal separator, then the separation class, $T_{S_{max}}$, is a transposition equivalence class.

**Proof.**

Since a maximal separator cannot be refined any further, then it must have all of its paths closed by the application of [S.1], therefore, without loss of generality, it can be assumed that the shape of $S_{max}$ is the following:

$$S_{max} \equiv t_S \langle (\{\}, s_{r_1}^{max}), \ldots, (\{\}, s_{r_n}^{max}) \rangle$$

where $s_{r_i}^{max}$ contains all the decisions not in the path to the ith leaf of $S_{max}$.

Since a separation class contains all the transposition variants in it, then there exists a $t \in T_{S_{max}}$, such that $t \equiv t_S \langle x_1' \ldots' x_n \rangle$, where the $x_i$ are variables.

Consider another tree $t' \in T_{S_{max}}$. Since $t'$ satisfies $S_{max}$, then $t'$ must be transposition equivalent to $t_S \langle t_1' ... ' t_n \rangle$, where $t_i$ satisfies $(\{\}, s_{r_i}^{max})$. For $t_i$ to satisfy the primitive plan $(\{\}, s_{r_i}^{max})$, $t_i$ must be a variable. Therefore, it follows that, $t' =_t t$, so $T_{S_{max}}$ is a transposition equivalence class. ■

The separator plans for a separator $S$ can be ordered by the amount of refinement of the separators in each plan.

**Definition 3.26**

(i) A *complete separator plan* is a separator plan, $S \rightrightarrows P_S$, where $P_S = \{S_1, ..., S_n\}$, such that each separator $S_i \in P_S$ is either maximal or inconsistent.

(ii) A *minimal complete separator plan* is one that is not the refinement of any complete separator plan.

Given a complete separator plan, $P_S$, one can generate a tree for each of the transposition equivalent classes in $T$ by finding, for all the maximal separators, $S_{max} \in P_S$, the corresponding tree in $T$ that has the same structure as $S_{max}$ but with variables at the leaves.

Given a separation class, $T$, then a minimal complete separator plan for $T$, contains a disjoint description of all the transposition equivalent classes in $T$. In general, when generating all the irreducible forms of a tree it is desirable to avoid the generation of transposition equivalent irreducibles since the are equally efficient. Hence the importance of being able to generate a minimal complete separator plan. The following theorem describes a method for generating a minimal complete separator plan.

**Theorem 3.27**

Given a separation class, $T$, the null separator, $(\{\}, \{\})$, and the leaf refinement rules [S.1]-[S.2], then every consistent application of the leaf

refinement rules [S.1]-[S.2], starting from ({}, {}), terminates in a minimal complete separator plan.

**Proof.**

The null separator, ({}, {}), is always consistent since any decision tree $t \in T$ satisfies it. Any consistent application of [S.1]-[S.2] results in a more refined separator plan which is also consistent. Proposition 3.24 shows that given a consistent separator, $S$, either $S$ is maximal or there is a consistent separator plan for $S$.

If $S$ is an inconsistent separator, then $T_S = \varnothing$ since no term in $T$ satisfies $S$. Therefore, an inconsistent plan can not be refined into a consistent separator plan. Hence, a consistent application of [S.1]-[S.2] can only be performed on separators that are neither maximal nor inconsistent.

As Proposition 3.24 guarantees that there always exists a consistent separator plan for a non-maximal consistent separator, then the application of [S.1]-[S.2] terminates whenever all the separators in the separator plan are either maximal or inconsistent, that is, [S.1]-[S.2] always terminate in a complete separator plan. Furthermore, such plan is also minimal, since the application of [S.1] and [S.2] always result in irreducible separator plans and once all the separators are either maximal or inconsistent no more consistent applications of [S.1]-[S.2] can be obtained. ■

An algorithm for obtaining all the irreducible forms of a tree without identities based on the principle of separation planning is introduced in the following section.

### 3.1.3 The Separation Planning Algorithm

An algorithm which implements the separation planning technique is presented next. Given an initial separator plan containing only the null separator and a separation class, the algorithm develops a complete separation plan. At every stage, the algorithm recursively develops a more refined separator plan by refining any one of its consistent separators by means of a consistent application of either [S.1] or [S.2] at the leaves. The algorithm terminates when this process can not be repeated any further.

**Algorithm 3.28 [Separation Planning]**

SEPARATION PLANNING ($P$, $T$):

**if** COMPLETE PLAN ($P$, $T$)

**then return** $P$;

**else begin**

$S \in$ NON-MAXIMAL SEPARATORS ($P$, $T$);

$P_S \in$ CONSISTENT REFINEMENT ($S$);

$P' \in$ SEPARATION PLANNING( $(P - S) \cup P_S$, $T$);

**return** $P'$;

**end**

The procedure COMPLETE PLAN evaluates if the initial plan is a complete separator plan. If it is, then the procedure SEPARATION PLANNING returns the plan $P$. Otherwise, a non maximal separator $S$ is returned from the non-deterministic procedure NON-MAXIMAL SEPARATORS. A consistent refinement $P_S$ is then obtained from $S$ by means of the procedure CONSISTENT REFINEMENT. Finally, the plan resulting from removing the separator $S$ and replacing it with the refinement $P_S$ is refined further by calling the procedure SEPARATION PLANNING recursively.

The following example illustrates how Algorithm 3.28 is used to generate a complete separator plan.

**Example 3.29**

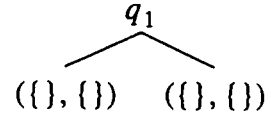
Let $D = (\{q_1, q_2\}, \{\})$ be a decision theory and $T$ be all the repeat reduced trees in $D$. To generate a minimal complete separator plan, start with the null separator and use [S.1] to produce the following plan:

$$(\{\}, \{\}) \mapsto \{(\{q_1\}, \{\}), (\{\}, \{q_1\})\}$$

Using [S.2] on the first separator in the list, results in the plan

$$(\{\}, \{\}) \mapsto \{S_1, S_2\}$$

where

$S_1 =$ 

, $S_2 = (\{\}, \{q_1\})$

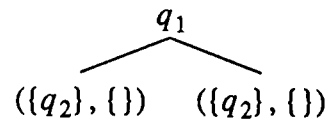
The following leaf plan

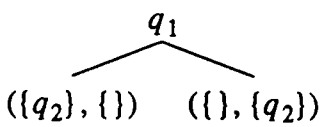
$$(\{\}, \{\}) / \{q_1\} \mapsto \{(\{q_2\}, \{\}), (\{\}, \{q_2\})\}$$

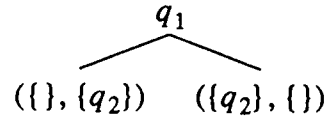
can then be used to refine both leaves of $S_1$ to produce the plan

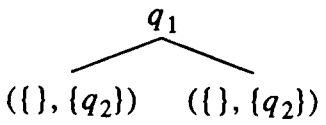
$$(\{\}, \{\}) \mapsto \{S_{11}, S_{12}, S_{13}, S_{14}, S_2\}$$

where

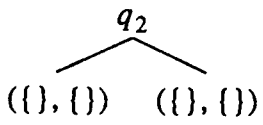
$S_{11} =$ 


$S_{12} =$ 


$S_{13} =$ 


$S_{14} =$ 

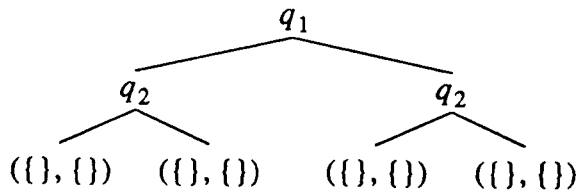

Now, applying the leaf plan

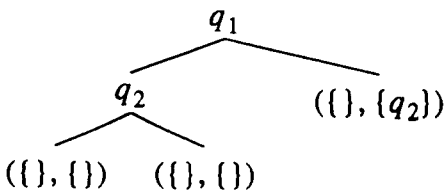
$(\{q_2\}, \{\})\ /\{q_1\} \Longmapsto$ 

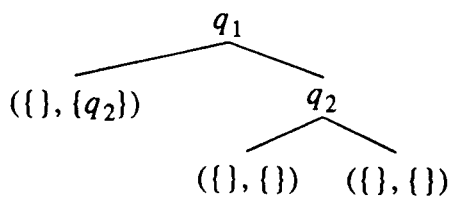

on the separators $S_{11}$, $S_{12}$, and $S_{13}$, results in

$$(\{\}, \{\}) \Longmapsto \{S_{111}, S_{121}, S_{131}, S_{14}, S_2\}$$

where

$S_{111} =$ 


$S_{121} =$ 


$S_{131} =$ 

$S_2$ can be refined by means of the following leaf plan

$$(\{\}, \{q_1\})\,/\{\} \Rightarrow \{(\{q_2\}, \{q_1\}), (\{\}, \{q_1, q_2\})\}$$

resulting in the separator plan

$$(\{\}, \{\}) \Rightarrow \{S_{111}, S_{121}, S_{131}, S_{14}, S_{21}, S_{22}\}$$

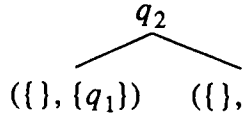
where

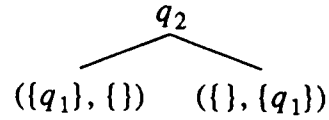
$$S_{21} = (\{q_2\}, \{q_1\}) \text{ and } S_{22} = (\{\}, \{q_1, q_2\})$$

Consider now the leaf plan

$$(\{q_2\}, \{q_1\})\,/\{\} \Rightarrow \{S_{211}, S_{212}\}$$

where

$S_{211}$ = 


$S_{212}$ = 


Replacing $S_{21}$ with $S_{211}$ and $S_{212}$ yields

$$(\{\}, \{\}) \Rightarrow \{S_{111}, S_{121}, S_{131}, S_{14}, S_{211}, S_{212}, S_{22}\}$$

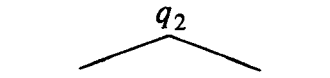
$S_{211}$ can then be refined with the following leaf plan

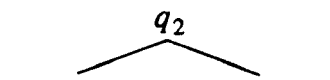
$$(\{\}, \{\})\,/\{q_2\} \Rightarrow \{(\{q_1\}, \{\}), (\{\}, \{q_1\})\}$$

producing

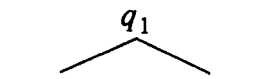
$$(\{\}, \{\}) \Rightarrow \{S_{111}, S_{121}, S_{131}, S_{14}, S_{2111}, S_{2112}, S_{212}, S_{22}\}$$

where

$S_{2111} =$ 


$S_{2112} =$ 

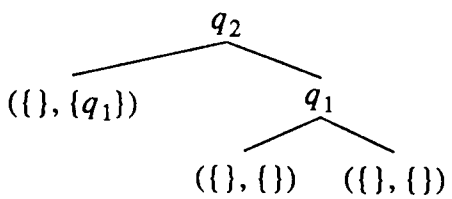

finally, the following leaf plan

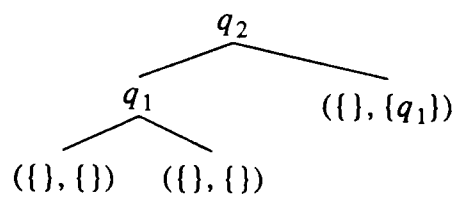
$(\{q_1\}, \{\}) \,/\, \{q_2\} \Longmapsto$ 


can be used on $S_{2111}$ and $S_{212}$ (the only non-maximal separators) to obtain the minimal complete separator plan:

$$(\{\}, \{\}) \mapsto \{S_{111}, S_{121}, S_{131}, S_{14}, S_{21111}, S_{2112}, S_{2121}, S_{22}\}$$

where

$S_{21111} =$ 


$S_{2121} =$ 


■

Algorithm 3.28 is very expensive in terms of the amount of storage that it requires. An improved separation algorithm is presented next. The improvement consists in keeping only one separator at any given time and relying on a recursive procedure to generate all the

separators in the complete separator plan. Only those separators that are maximal in the initial separation class are reported.

Given the null separator, and a separation class, the following algorithm can recursively generate all the maximal separators of a complete separation plan.

**Algorithm 3.30 [Separation Planning]**

MAXIMAL SEPARATOR ($S$, $T$):

**if** MAXIMAL ($S$)

**then return** $S$;

**else begin**

$S' \in$ LEAF REFINEMENT ($S$)

**return** MAXIMAL SEPARATOR ($S'$, $T$);

**end**

Given an initial separator, Algorithm 3.30 generates a maximal separator by recursively refining the leaves of the initial separator (through the calls to the procedure LEAF REFINEMENT) until neither [S.1] nor [S.2] can be applied consistently. Once a maximal separator is generated (the procedure MAXIMAL returns a true value), it is reported. In order to generate a new maximal separator, the algorithm backtracks to the last choice point and selects a new leaf refinement. By starting with the null separator, the algorithm is capable of generating all the maximal separators by exhausting all the leaf refinement choices.

The correctness of Algorithms 3.28 and 3.30 is determined by Theorem 3.27. Both algorithms start with the null separator and make use of the leaf refining rules [S.1] and [S.2] alone to obtain more refined separator plans. Since the algorithms terminate when [S.1] and [S.2] can no longer be applied consistently, then, by Theorem 3.27, the result is a minimal complete separator plan.

The use of separator plans ensure that the decisions that have already been placed at the root of the term are forced to be semi-redundant in future terms by decision delaying them until they disappear down at least one branch. The separator plans are also used to generate alternate solutions to the separation problem by forcing different forms of the tree. To ensure that no duplicate trees are generated, each separator specifies exactly the kind of occurrence a particular decision is to have in any one of the paths of the tree.

## 3.2 Generating All the Irreducible Forms of a Decision Tree

An algorithm that uses separation planning to generate all the irreducible forms of a decision tree is described next. The algorithm is a modification of Algorithm 3.30 in which only the irreducible forms of the separators consistent with the input tree are reported. Therefore, the separators generated are developed until they are maximal and then they are checked for irreducibility. To check for irreducibility at a term whose subtrees are all known to be irreducible, it is necessary to decision factor the term by its root decision, $q$, and verify that the idempotent law cannot be used to remove $q$ from the tree. If $q$ can be removed, then the term is not irreducible.

The input to the algorithm is a simply reduced tree and the null separator plan. The output consists of all the irreducible forms of the original tree. The notation of the algorithm assumes that the separator $S$ has the following structure

$$S \equiv t_S \langle s_1 \, ' \ldots ' \, s_i \, ' \ldots ' \, s_n \rangle,$$

while the tree $t$ has the following form:

$$t \equiv t_S \langle t_1 \, ' \ldots ' \, t_i \, ' \ldots ' \, t_n \rangle, \text{ where } t_k \text{ satisfies } s_k \text{ for } 1 \leq k \leq n.$$

**Algorithm 3.31 [Finding all the Irreducible Trees]**

GET ALL IRREDUCIBLES ($S$, $t$):

**if** MAXIMAL ($S$)

**then begin**

**if** IRREDUCIBLE ($t$)

**then return** $t$;

**else** BACKTRACK TO LAST CHOICE POINT;

**end**

**else begin**

$S_j \in$ LEAF REFINEMENT ($s_i$, $Q_i$);

$t_j \leftarrow$ APPLY SEPARATOR PLAN ($S_j$, $t_i$);

$t' \leftarrow$ GET ALL IRREDUCIBLES ($t_S\langle \ldots , s_{i-1} , S_j , s_{i+1} , \ldots\rangle$,

$t_S\langle \ldots , t_{i-1} , t_j , t_{i+1} , \ldots\rangle$);

**return** $t'$;

**end**

The algorithm works as follows. It first test if the separator $S$ is maximal (MAXIMAL returns a true value). If $S$ is maximal and $t$ is irreducible, then the tree $t$ is reported. If $t$ is not irreducible, then the algorithm backtracks to the last choice point. Once a decision tree is reported, the user can request an alternative optimal solution by forcing the algorithm to backtrack to the last choice point. If the separator is not maximal, then a leaf refinement is generated non-deterministically through the procedure LEAF REFINEMENT (this creates a choice point). The leaf refinement is subsequently applied to $t$ and the procedure GET ALL IRREDUCIBLES is called recursively to obtain an irreducible tree corresponding to a maximal separator.

**Theorem 3.32**

Given a simply reduced tree, $t$, Algorithm 3.31 finds all the irreducible forms for $t$.

**Proof.**

The algorithm starts with the null separator, which, by definition, is consistent with $t$ and makes use of the leaf refining rules [S.1] and [S.2] alone to obtain more refined separators. Since the algorithm terminates when [S.1] and [S.2] can no longer be applied consistently, then, by Theorem 3.27, the result is a minimal complete separator plan.

For every separator, $S$, obtained, a corresponding consistent tree is generated whose tree portion is identical to the tree portion of $S$.

Every time a maximal separator is generated, its corresponding consistent tree is checked for irreducibility. If the tree is irreducible, then it is reported.

The algorithm terminates when all the choice points have been exhausted, i.e., when all the maximal separators in a minimal complete separator plan have been visited.

Since $t$ is simply reduced, the complete separator plan generated corresponds to a separation class that includes all the decision equivalent forms of $t$. Since all the irreducible forms of $t$ are included in this class, then all non-transposition equivalent irreducibles are visited by the algorithm.

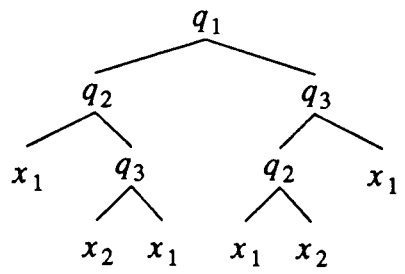
Therefore, Algorithm 3.31 obtains all the irreducible forms for $t$. ■

The following examples illustrate how Algorithm 3.31 can be used to generate all the optimal forms of a decision tree.

**Example 3.33**

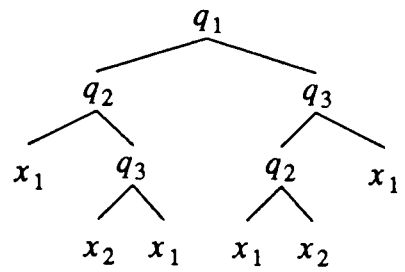
Consider the tree, $t$, in the decision theory $D = (\{q_1, q_2, q_3\}, \{\})$, such that $t$ has the following form:

$t =$

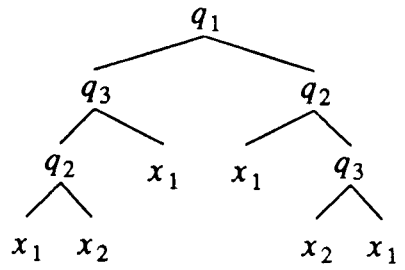



Algorithm 3.31 can be used to generate the set $T_{\text{irred}} = \{t_1, t_2, t_3, t_4\}$, containing all the irreducible forms of $t$, where:

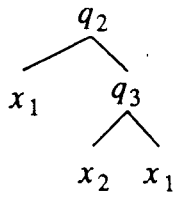
$t_1 =$



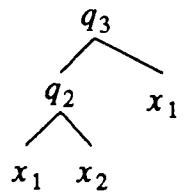

$t_2 =$




$t_3 =$



$t_4 =$

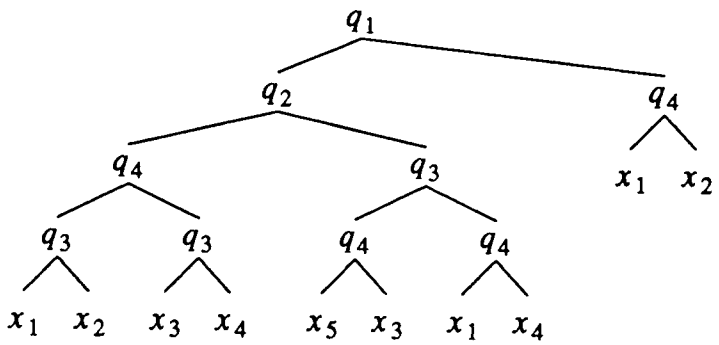



■

**Example 3.34**

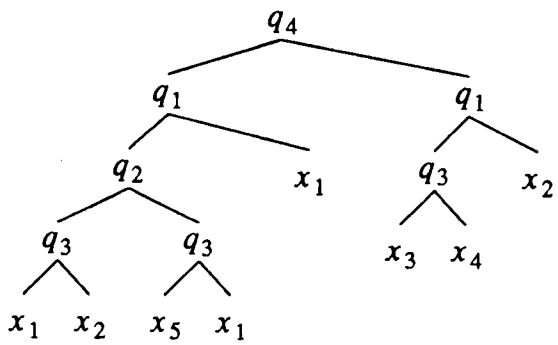
Consider a tree, $t$, in the decision theory $D = (\{q_1, q_2, q_3, q_4\}, \{\})$, such that $t$ has the following form:

$t =$

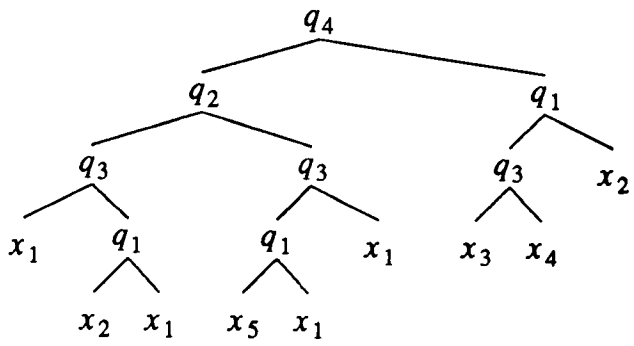



Algorithm 3.31 can be used to generate the set $T_{\text{irred}} = \{t_1, t_2, t_3, t_4, t_5, t_6\}$, containing all the irreducible forms of $t$, where:

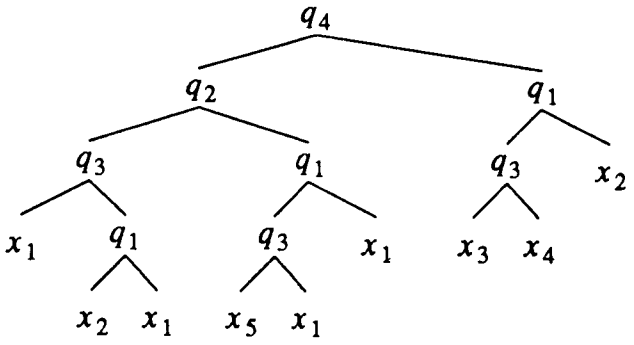
$t_1 =$



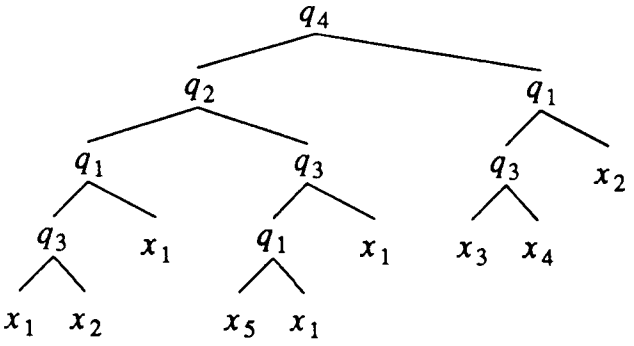
$t_2 =$


$q_4$
$q_2$ $q_1$
$q_3$ $q_3$ $q_3$ $x_2$
$x_1$ $q_1$ $q_1$ $x_1$ $x_3$ $x_4$
$x_2$ $x_1$ $x_5$ $x_1$

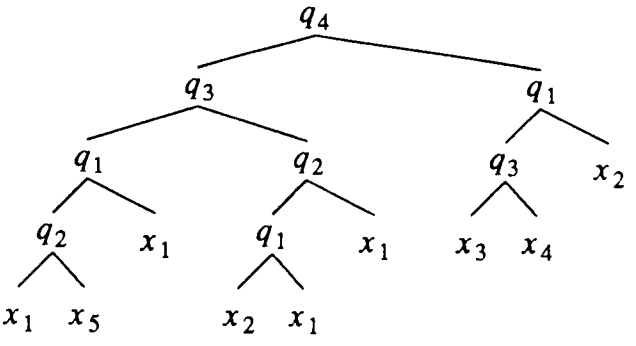
$t_3 =$


$q_4$
$q_2$ $q_1$
$q_3$ $q_1$ $q_3$ $x_2$
$x_1$ $q_1$ $q_3$ $x_1$ $x_3$ $x_4$
$x_2$ $x_1$ $x_5$ $x_1$

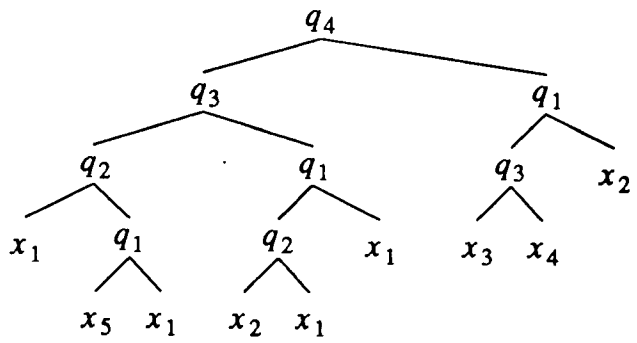
$t_4 =$


$q_4$
$q_2$ $q_1$
$q_1$ $q_3$ $q_3$ $x_2$
$q_3$ $x_1$ $q_1$ $x_1$ $x_3$ $x_4$
$x_1$ $x_2$ $x_5$ $x_1$

$t_5 =$


$q_4$
$q_3$ $q_1$
$q_1$ $q_2$ $q_3$ $x_2$
$q_2$ $x_1$ $q_1$ $x_1$ $x_3$ $x_4$
$x_1$ $x_5$ $x_2$ $x_1$

$t_6 =$




■

Algorithm 3.31 can be implemented more efficiently by observing that there are several ways by which the refinement of unfruitful separators can be avoided. For instance, if a decision that is known to be essential in a term is added to the semi-redundant requirement of the separator, it is obvious that no further refinement of the separator will yield one that is consistent. Similarly, if a decision that does not occur in the term is added to the semi-essential requirement of the separator, then any further refinement of the separator will only result in the introduction of redundant decisions.

# CHAPTER 4

## DECISION THEORIES WITH IDENTITIES

Thus far the discussion of decision theories has avoided the problems that arise when identities, over and above [D.1]-[D.3], are introduced. Intuitively, this corresponds to the case when the decisions in the theory are totally independent. In practice, however, this is rarely the case.

The presence of identities in a decision theory implies the existence of dependencies between some of the decisions in the theory. In particular, the outcome of some set of decisions may have implications concerning the possible outcomes of another set of decisions. In most applications, decisions are related in this manner and thus, it is important to address this issue.

The relationships between the decisions in a decision theory can be represented either as equalities between the terms in the theory (proposed by Cockett in [11]), as propositional rules in internal disjunctive form (used extensively by Michalski, e.g. [31]), or as attribute value dependencies (proposed by Chen in [5]). The latter representation is a very useful one since identities in a decision theory can be shown to correspond to relationships between the attributes in an information system. In particular, Chen's notion of knowledge representation trees, in which knowledge deduction rules corresponding to attribute value dependencies are incorporated into the decision tree structure, gives a tight and useful representation for identities which shall be used in the next chapter.

Cockett [9] was responsible for developing the notion of characterizing identities and for demonstrating that every identity between two terms in a decision theory corresponds to

a characterizing identity. In addition, he was first to propose a Boolean algebra for the manipulation of characterizing identities.

However, several important issues were left unsolved by Cockett's original research. Some of these issues were solved in the course of this research effort. In particular, one of these problems was that of determining the retrieval of an expression when arbitrary identities are present. This *retrieval problem* turned out to be a crucial one for this investigation. In order to extend the algorithms presented in Chapter 3 to decision theories with identities, this problem had to be solved. The purpose of this chapter is to develop the machinery that will be required when a solution to this problem is presented in the next chapter.

The chapter is organized as follows. The first section summarizes some results originally due to Cockett which are relevant to this investigation and presents some improvements and extensions developed which were developed by the author. The second section introduces the algebra of characterizing terms and derives some additional results of relevance to the following chapter. The third section deals with the issue of deciding the equality of terms in decision theories with arbitrary decision identities.

## 4.1 Theoretical Preliminaries

The identities [D.1]-[D.3], that is, idempotence, transposition (distribution), and repetition, were shown in [9] to form a complete set of axioms. This means that, given any decision theory, they can derive any identity which holds true for all the terms in the theory. This result was shown to be correct even when arbitrary identities are present in the theory.

In the latter case, however, some of the manipulations introduced thus far take on a different form as the presence of arbitrary identities establishes the equality of terms which would not otherwise be equal. In addition, problems like deciding the equality of decision trees and evaluating retrievals from a term are affected by the constraints due to the way in which decisions are related to one another.

The purpose of this section is to show that there exist several ways of representing the dependencies of decisions in a decision theory. It will be shown that identities represented as equalities between decision trees, as introduced in [11], are equivalent to a very special kind of identities called characterizing identities. These characterizing identities in turn are shown to yield entailment identities which correspond to Chen's notion of knowledge representation trees.

As it will be proved shortly, every identity in a decision theory is equivalent to a characterizing identity. In other words, there always exists an equivalent representation of a decision theory which only uses characterizing identities. The main advantage of characterizing identities is that they can be represented by terms in a decision theory. In the discussion that follows, it is shown that these terms are well suited for syntactic manipulations and thus have considerable practical value.

**Definition 4.1**

An identity, $t_1 = t_2$, is a *characterizing identity* in a decision theory, $D = (Q, E)$, if $t_1$ and $t_2$ have the same internal tree structure, that is, are structurally identical, but differ by the variables at their leaves. Furthermore, there must only be three different variables: one which only occurs in $t_1$, one which only occurs in $t_2$ and one which whenever it occurs, occurs at the same leaf on both $t_1$ and $t_2$.

The variables which occur on one side of the identity but not on the other are said to be located on *unreachable* branches. This means that in every model of the theory, these branches classify the empty set (see [9]) and, as a result, they can be substituted arbitrarily. In contrast, the variables which occur on both sides of the identity are said to be located on *reachable* branches.

In practice, every characterizing identity can be represented by means of a *characterizing term.* A characterizing term is a tree in which the branches leading to unreachable leaves are labeled with the variable $F$ (for false or unreachable) and the ones leading to reachable leaves are labeled with the variable $T$ (for true or reachable).

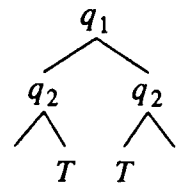
Thus, the characterizing identity

$$q_1\langle q_2\langle L \ ' \ T\rangle ' q_2\langle T \ ' \ L\rangle\rangle = q_1\langle q_2\langle R \ ' \ T\rangle ' q_2\langle T \ ' \ R\rangle\rangle$$

has as its characterizing term the following decision tree

$$q_1\langle q_2\langle F \ ' \ T\rangle ' q_2\langle T \ ' \ F\rangle\rangle.$$

A characterizing term is a way of representing an equality between two terms as a single term. Graphically, these terms will be represented as decision trees of the form




in which unreachable branches are totally omitted.

In order to show that any identity of the form $t_1 = t_2$ is equivalent to a characterizing identity it is necessary to overcome two major obstacles. The first obstacle is to arrange an

equivalent identity with the same tree structure on the left and on the right. The second obstacle is to arrange both sides of the equation so the variables have the same form.

First, consider:

$$t_1 = t_2/t_1 = id(rp(t_2/t_1)).$$

The form of the result has $id(rp(t_2))$ with certain trees attached to the leaves, say $t_{11},\ldots,t_{1n}$. This can be denoted by:

$$id(rp(t_2/t_1)) = id(rp(t_2))/\langle t_{11},\ldots,t_{1n}\rangle.$$

Notice that the trees $t_{11},\ldots,t_{1n}$ are all remnants of $id(rp(t_1))$. Now, it may be assumed that $t_2 = id(rp(t_2))$ and, if $x_{ir}$ is the variable at the *rth* leaf of $id(rp(t_2))$, then

$$t_2 = id(rp(t_2)) = id(rp(t_2))/\langle t_{11}/x_{i1},\ldots,t_{1n}/x_{in}\rangle.$$

Where $t_{1j}/x_{ir}$ is the tree structurally identical to $t_{1j}$, $j = 1,\ldots,n$, but with all of its variables replaced by $x_{ir}$. The original identity is thus equivalent to

$$id(rp(t_2))/\langle t_{11},\ldots,t_{1n}\rangle = id(rp(t_2))/\langle t_{11}/x_{i1},\ldots,t_{1n}/x_{in}\rangle,$$

where now the trees are structurally identical.

The next step involves a process called variable separation. Variable separation consists of renaming the variables of the terms so that on each leaf of either term the variables are distinct. This is accomplished by substituting for each variable, $x_i$, a term which is structurally identical but whose variables are distinct on each leaf. This may be achieved by placing $x_{ir}$ on the rth leaf from the left in the tree which is to replace $x_i$. Once the substitutions have been made, repeat reductions of the resultant trees (on both sides of the identity) will result in an identity in which the variables are separated. In fact, the

variable on the rth leaf of each term is now of the form $x_{ir}$ where $i$ may vary according to which side of the identity is examined and refers to the original variable at that leaf. The $r$ refers to the leaf number.

Notice that the original identity may be obtained from the variable separated identity by means of the substitution

$$x_{ir} \leftarrow\!\shortmid x_i ,$$

for every $i$ and $r$. Thus the variable separated identity is equivalent to the original identity.

Finally, once a structurally identical, variable separated identity has been obtained, its variables can be substituted as follows to obtain a characterizing identity:

(i) substitute $L$ if the variable occurs only on the LHS,

(ii) substitute $R$ if the variable occurs only on the RHS,

(iii) substitute $T$ if it occurs on both sides.

The separated identity can be recaptured again by variable separation thus the characterizing identity is equivalent to the previous identity. This gives:

**Proposition 4.2**

Every identity is equivalent to a characterizing identity.

**Proof.**

By the argument above, given an arbitrary identity of the form $t_1 = t_2$, one can obtain an equivalent identity in which both sides are in structurally identical form. Finally, using variable separation one can generate the corresponding characterizing identity. ■

**Example 4.3**

Consider the identity

$$q_1\langle x_1 \,' x_2\rangle = q_2\langle x_2 \,' x_1\rangle$$

Putting it in structurally identical form gives:

$$q_2\langle q_1\langle x_1\ 'x_2\rangle' q_1\langle x_1\ 'x_2\rangle\rangle = q_2\langle q_1\langle x_2\ 'x_2\rangle' q_1\langle x_1\ 'x_1\rangle\rangle$$

Variable separation results in:

$$q_2\langle q_1\langle x_{11}\ 'x_{22}\rangle' q_1\langle x_{13}\ 'x_{24}\rangle\rangle = q_2\langle q_1\langle x_{21}\ 'x_{22}\rangle' q_1\langle x_{13}\ 'x_{14}\rangle\rangle$$

Identifying in the reachable and unreachable leaves gives the characterizing identity:

$$q_2\langle q_1\langle L\ 'T\rangle' q_1\langle T\ 'L\rangle\rangle = q_2\langle q_1\langle R\ 'T\rangle' q_1\langle T\ 'R\rangle\rangle \quad \blacksquare$$

Alternatively, examining both sides of the identity

$$id(rp(t_2))/\langle t_{11},\ldots,t_{1n}\rangle = id(rp(t_2))/\langle t_{11}/x_{i1},\ldots,t_{1n}/x_{in}\rangle$$

and identifying which variables occur in the same location on both sides, then it is clear that the identity can be represented by a single characterizing term of the form:

$$id(rp(t_2))/\langle s_{11},\ldots,s_{1n}\rangle,$$

where $s_{1j}$ is a tree structurally identical to $t_{1j}$, for each $j = 1,\ldots,n$, whose *kth* leaf is labeled with the variable $T$, if the corresponding kth leaf of $t_{1j}$ is equal to $x_{ir}$ (the *rth* leaf of $id(rp(t_2))$), or labeled with the variable $F$ otherwise.

Therefore, given an identity of the form $t_1 = t_2$, an equivalent characterizing term is given by the expression:

$$id(rp(t_2[x_i \leftarrow t_1[x_j \leftarrow \delta(x_i,x_j)]])),$$

where $\delta(a,b) = T$, if $a \equiv$ b and $\delta(a,b) = F$ otherwise.

Thus, in the example above, the identity can be expressed as a characterizing term of the form:

$$q_2\langle x_2 \,'\, x_1\rangle[x_1 \leftarrow q_1\langle x_1 \,'\, x_2\rangle[x_1 \leftarrow T, x_2 \leftarrow F],$$
$$x_2 \leftarrow q_1\langle x_1 \,'\, x_2\rangle[x_1 \leftarrow F, x_2 \leftarrow T]\ ]$$

$$= q_2\langle x_2 \,'\, x_1\rangle[x_1 \leftarrow q_1\langle T \,'\, F\rangle, x_2 \leftarrow q_1\langle F \,'\, T\rangle]$$

which, upon substitution, results in the characterizing term

$$q_2\langle q_1\langle F \,'\, T\rangle' q_1\langle T \,'\, F\rangle\rangle.$$

**Definition 4.4**

An identity, $t_1 = t_2$, is *valid* in $D = (Q, E)$ if it can be derived algebraically from the identities in $E$ and the decision identities [D.1]-[D.3].

Characterizing terms allow for a rather special sort of substitution.

**Lemma 4.5 [Reachable Substitution]**

If e is a valid characterizing term in $D = (Q, E)$ then, for any term $t$ in $D$, the identity, $t = e[T \leftarrow t]$, in which only the reachable variable $T$ is substituted, is valid.

**Proof.**

Let the characterizing term e be of the form $e\langle L \,'\, T\rangle = e\langle R \,'\, T\rangle$.

It is clear that $t = e\langle t \,'\, t\rangle$. But also,

$$e\langle L \,'\, t\rangle = e\langle t \,'\, t\rangle = e\langle R \,'\, t\rangle.$$

Hence the identity holds. ■

**Definition 4.6**

(i) Given two identities, $t_1=t_2$ and $s_1=s_2$, then $t_1=t_2$ *implies* $s_1=s_2$, denoted $(t_1=t_2) \Rightarrow (s_1=s_2)$, if $s_1=s_2$ is a valid identity in $D' = (Q,\{t_1=t_2\})$.

(ii) One identity, $t_1=t_2$, with corresponding characterizing term $e_1$, is said to be *decision equivalent* to another, $s_1=s_2$, with corresponding characterizing term $e_2$, if $e_1$ is decision equivalent to $e_2$.

In fact, given the characterizing terms $e_1$ and $e_2$, in $D = (Q, E)$, then $e_1$ implies $e_2$, denoted $e_1 \Rightarrow e_2$, if and only if, $D' = (Q, \{e_1\})$ has $e_2$ valid. It is clear then that the characterizing terms are preordered by implication. Furthermore, as shown in next section, when the elements of this preorder are taken as equivalence classes under decision equivalence, the result is a partial ordering.

## 4.2 The Algebra of Characterizing Terms

Two characterizing terms, $e_1$ and $e_2$, can be amalgamated into a single term, denoted $e_1 \wedge e_2$, by placing a copy of $e_2$ on each reachable leaf of $e_1$, to form:

$$e_1\langle F\ 'T\rangle[T \leftarrowtail e_2\langle F\ 'T\rangle]$$

It is clear that if $e_1$ and $e_2$ are valid terms then $e_1 \wedge e_2$ is valid. Furthermore, $e_1 \wedge e_2$ is a characterizing term.

Similarly, if $e_1$ is a characterizing term then, rather than substituting for $T$, one could substitute for $F$. In fact, given a second characterizing term, $e_2$, one can form:

$$e_1\langle F\ 'T\rangle[F \leftarrowtail e_2\langle F\ 'T\rangle]$$

which is denoted $e_1 \vee e_2$. Obviously, $e_1$ implies $e_1 \vee e_2$ as the above substitution maintains the identity even when $e_2$ is not a valid term.

Substituting $e_2\langle T\ 'T\rangle$ for $T$ in $e_1 \vee e_2$ results in:

$$e_1\langle F\ 'T\rangle[F \leftarrowtail e_2\langle F\ 'T\rangle, T \leftarrowtail e_2\langle T\ 'T\rangle]$$

from which the following characterizing term is obtained

$$e_1\langle e_2\langle F\ 'T\rangle 'e_2\langle T\ 'T\rangle\rangle.$$

Applying transposition yields:

$$e_2\langle e_1\langle F\ 'T\rangle 'e_1\langle T\ 'T\rangle\rangle,$$

which is the same, applying idempotence, as

$$e_2\langle F\ 'T\rangle[F \leftarrowtail e_1\langle F\ 'T\rangle].$$

Thus, $e_1$ implies $e_1 \vee e_2$, which is decision equivalent to $e_2 \vee e_1$. As the next result confirms, the operators "$\wedge$" and "$\vee$" behave like the join and meet lattice operations. The "top" and the "bottom" of the lattice correspond to the special characterizing terms $T$ (corresponding to the characterizing identity $T = T$) and $F$ (corresponding to the characterizing identity $L = R$) respectively. In addition to the join and meet operators, a complementation operator, $\neg$, can be defined by swapping the variables $T$ and $F$.

**Lemma 4.7**

If $e_1$, $e_2$ and $e_3$ are characterizing terms, then the following decision equivalences hold:

(i) Commutative laws:

$e_1 \wedge e_2 = e_2 \wedge e_1$ and $e_1 \vee e_2 = e_2 \vee e_1$.

(ii) Associative laws:

$(e_1 \wedge e_2) \wedge e_3 = e_1 \wedge (e_2 \wedge e_3)$ and $(e_1 \vee e_2) \vee e_3 = e_1 \vee (e_2 \vee e_3)$.

(iii) Distributive laws:

$e_1 \vee (e_2 \wedge e_3) = (e_1 \vee e_2) \wedge (e_1 \vee e_3)$ and

$e_1 \wedge (e_2 \vee e_3) = (e_1 \wedge e_2) \vee (e_1 \wedge e_3)$.

(iv) Identity laws:

$T \wedge e_1 = e_1$ and $F \vee e_1 = e_1$.

(v) Complement laws:

$\neg e_1 \wedge e_1 = F$ and $\neg e_1 \vee e_1 = T$.

**Proof.**

Notice that in each of these sets of laws one equation is the dual of the other. Due to this duality between the join and meet operators and the top ($T$) and the bottom ($F$) of the lattice, only one equation in each set of postulates needs to be proved in order to show that the lemma is correct.

(i) To show that the lattice is commutative, note that:

$$e_1 \wedge e_2 = e_1\langle F\ 'T\rangle[T \leftarrowtail e_2\langle F\ 'T\rangle].$$

Replacing $F$ with $e_2\langle F\ 'F\rangle$ yields:

$$e_1\langle F\ 'T\rangle[T \leftarrowtail e_2\langle F\ 'T\rangle, F \leftarrowtail e_2\langle F\ 'F\rangle] =$$
$$e_1\langle e_2\langle F\ 'F\rangle 'e_2\langle F\ 'T\rangle\rangle.$$

Transposing the terms $e_1$ and $e_2$, results in

$$e_2\langle e_1\langle F\ 'F\rangle 'e_1\langle F\ 'T\rangle\rangle,$$

which, by idempotence, reduces to

$$e_2\langle F\ 'T\rangle[T \leftarrow e_1\langle F\ 'T\rangle] = e_2 \wedge e_1.$$

(ii) To show that the associative law holds in this lattice, notice that solving the first equation yields:

$$(e_1 \wedge e_2) \wedge e_3 = e_1 \wedge (e_2 \wedge e_3)$$

$$(e_1\langle F'T\rangle[T \leftarrow e_2\langle F'T\rangle]) \wedge e_3 = e_1 \wedge (e_2\langle F'T\rangle[T \leftarrow e_3\langle F'T\rangle])$$

$$e_1\langle F\ 'e_2\langle F'T\rangle\rangle \wedge e_3 = e_1 \wedge e_2\langle F\ 'e_3\langle F'T\rangle\rangle$$

$$e_1\langle F\ 'e_2\langle F'T\rangle\rangle[T \leftarrow e_3\langle F'T\rangle] = e_1\langle F'T\rangle[T \leftarrow e_2\langle F\ 'e_3\langle F'T\rangle\rangle]$$

$$e_1\langle F\ 'e_2\langle F\ 'e_3\langle F'T\rangle\rangle\rangle = e_1\langle F\ 'e_2\langle F\ 'e_3\langle F'T\rangle\rangle\rangle$$

(iii) To show that the lattice is distributive, observe that solving the left-hand side of the first equation gives:

$$\begin{aligned} e_1 \vee (e_2 \wedge e_3) &= e_1\langle F\ 'T\rangle \vee (e_2\langle F\ 'T\rangle[T \leftarrow e_3\langle F\ 'T\rangle]) \\ &= e_1\langle F\ 'T\rangle[F \leftarrow e_2\langle F\ 'e_3\langle F\ 'T\rangle\rangle] \\ &= e_1\langle e_2\langle F\ 'e_3\langle F\ 'T\rangle\rangle\ 'T\rangle. \end{aligned}$$

Similarly, solving the right-hand side of the equation yields:

$$\begin{aligned} &(e_1 \vee e_2) \wedge (e_1 \vee e_3) = \\ &\quad = (e_1\langle F'T\rangle[F \leftarrow e_2\langle F'T\rangle]) \wedge (e_1\langle F'T\rangle[F \leftarrow e_3\langle F'T\rangle]) \\ &\quad = e_1\langle e_2\langle F'T\rangle'T\rangle \wedge e_1\langle e_3\langle F'T\rangle'T\rangle \\ &\quad = e_1\langle e_2\langle F'T\rangle'T\rangle[T \leftarrow e_1\langle e_3\langle F'T\rangle'T\rangle] \\ &\quad = e_1\langle e_2\langle F\ 'e_1\langle e_3\langle F'T\rangle'T\rangle\rangle'e_1\langle e_3\langle F'T\rangle'T\rangle\rangle. \end{aligned}$$

Which, by repeat reduction, reduces to:

$$e_1\langle e_2\langle F\ 'e_3\langle F\ 'T\rangle\rangle\ 'T\rangle.$$

Therefore, $e_1 \vee (e_2 \wedge e_3) = (e_1 \vee e_2) \wedge (e_1 \vee e_3)$.

(iv) To proof the identity law, just note that:

$$T \wedge e_1 = T\,[T \leftarrow e_1] = e_1.$$

(v) To show that the lattice is complemented, observe that :

$$\begin{aligned} \neg e_1\langle F\ 'T\rangle \wedge e_1\langle F\ 'T\rangle &= e_1\langle T\ 'F\rangle \wedge e_1\langle F\ 'T\rangle \\ &= e_1\langle T\ 'F\rangle[T \leftarrow e_1\langle F\ 'T\rangle] \\ &= e_1\langle e_1\langle F\ 'T\rangle\ 'F\rangle, \end{aligned}$$

which, can be reduced by idempotency to $F$.

Therefore, $\neg e_1 \wedge e_1 = F$. ■

The identities above give the characterization of a distributed and complemented lattice. Thus, the characterizing terms of a decision theory constitute a Boolean algebra. (For a more detailed discussion on Boolean algebras see [20, 26].)

**Corollary 4.8 [Characterizing Terms as a Boolean Algebra]**

The characterizing terms of a decision theory, $D = (Q, E)$, form the Boolean algebra $\langle \mathbf{B}; \neg, \vee, \wedge \rangle$, where $\mathbf{B}$ is the set of decision equivalence classes of the valid characterizing terms in $D$, and $\neg, \vee, \wedge$, are the complement, meet, and join operators, respectively.

**Proof.**

Given by the identities above (see [20, 26]). ■

In addition to the postulates of Boolean algebra that were proved above, the algebra of characterizing terms satisfies the following postulates.

**Corollary 4.9**

Given the Boolean algebra $\langle \mathbf{B};\neg,\vee,\wedge\rangle$ in the decision theory $D = (Q, E)$, were the elements of **B** are the equivalent classes (under decision equivalence) of the valid characterizing terms in $D$, then, for every $e_1, e_2 \in \mathbf{B}$, the following postulates hold:

(i) Involution law:

$\neg(\neg e_1) = e_1$.

(ii) Idempotent laws:

$e_1 \wedge e_1 = e_1$ and $e_1 \vee e_1 = e_1$.

(iii) Null laws:

$F \wedge e_1 = F$ and $T \vee e_1 = T$.

(iv) Absorption laws:

$e_1 \vee (e_1 \wedge e_2) = e_1$ and $e_1 \wedge (e_1 \vee e_2) = e_1$.

(v) De Morgan's laws:

$\neg e_1 \wedge \neg e_2 = \neg(e_1 \vee e_2)$ and $\neg e_1 \vee \neg e_2 = \neg(e_1 \wedge e_2)$.

**Proof.**

Follows from Corollary 4.8 as all of these laws are a consequence of $\langle \mathbf{B};\neg,\vee,\wedge\rangle$ being a Boolean algebra (see [20, 26]). ■

In addition to being a Boolean algebra, the lattice structure defined for the characterizing terms in a decision theory establishes a preorder between these terms. The following discussion shows that this preorder corresponds to that given by implication.

**Definition 4.10**

Given the characterizing terms $e_1$ and $e_2$, in a decision theory $D = (Q, E)$, then $e_1$ *precedes* $e_2$ in the Boolean preorder,denoted $e_1 \preceq_{\mathbf{B}} e_2$, if and only if, $e_1 \vee e_2 = e_2$.

**Lemma 4.11**

Given the characterizing terms $e_1$ and $e_2$, in a decision theory $D = (Q, E)$, then $e_1 \preceq_B e_2$ and $e_2 \preceq_B e_1$, if and only if, $e_1$ and $e_2$ are decision equivalent.

**Proof.**

By the Boolean preorder, $e_1 \preceq_B e_2$ if and only if $e_1 \vee e_2 = e_2$. Similarly, $e_2 \preceq_B e_1$ if and only if $e_2 \vee e_1 = e_1$.

But, since $e_1 \vee e_2 = e_2 \vee e_1$, it follows that $e_1$ and $e_2$ must be decision equivalent. ■

**Lemma 4.12**

Given the characterizing terms $e_1$ and $e_2$, in a decision theory $D = (Q, E)$, then $e_1 \preceq_B e_2$, if and only if, $e_1 \Rightarrow e_2$.

**Proof.**

This can be proved by showing that $e_1 \vee e_2 = e_2$ if and only if $e_1$ implies $e_2$.

($\Rightarrow$) If $e_1 \vee e_2 = e_2$, i.e. $e_1\langle T'F\rangle[F \leftarrowtail e_2\langle T'F\rangle] =_d e_2\langle T'F\rangle$, then it is obvious that there is a substitution of $e_1$ which gives a term equivalent to $e_2$.

($\Leftarrow$) Conversely, if $e_1 \Rightarrow e_2$, then there exists a substitution of $e_1$ that gives a term equivalent to $e_2$. Let $c_1$ and $c_2$ be the characterizing identities corresponding to the characterizing terms $e_1$ and $e_2$ respectively. If $c_1$ implies $c_2$ then there is a substitution for the variables $T, L, R$ such that $c_1[T \leftarrowtail t_T, L \leftarrowtail t_L, R \leftarrowtail t_R]$ is decision equivalent to $c_2$. As $c_2$ is a characterizing identity, it follows that in $t_T$ only the variable $T$ occurs, otherwise, $R$ or $L$ will occur on both sides of the identity. This makes $t_T$ decision equivalent to $T$. In $t_L$ only the variables $T$ and $L$ can occur and in $t_R$ only the variables $T$

and $R$ can occur for similar reasons. Furthermore, $t_L$ and $t_R$ must be structurally identical, their only difference being in using $L$ or $R$. Thus, $c_1\langle T\,'\,L \leftarrowtail t_L\,'\,R \leftarrowtail t_R\rangle$ is decision equivalent to $c_2$. But this corresponds to a substitution in $e_1$ of the form $e_1[F \leftarrowtail t_F]$.

Let $t_F$ be equal to $s\langle T\,'\,F\rangle$ and $e_1\langle T\,'\,s\langle T\,'\,F\rangle\rangle$ be such a term decision equivalent to $e_2\langle T\,'\,F\rangle$. Thus,

$$\begin{aligned} e_1 \vee e_2 &= e_1\langle T\,'\,F\rangle[F \leftarrowtail e_2\langle T\,'\,F\rangle] \\ &= e_1\langle T\,'\,F\rangle[F \leftarrowtail e_1\langle T\,'\,s\langle T\,'\,F\rangle\rangle] \\ &= e_1\langle T\,'\,e_1\langle T\,'\,s\langle T\,'\,F\rangle\rangle\rangle, \end{aligned}$$

which, by repeat reduction results in:

$$\begin{aligned} e_1 \vee e_2 &= e_1\langle T\,'\,s\langle T\,'\,F\rangle\rangle \\ &= e_2\langle T\,'\,F\rangle \end{aligned}$$

Thus, implying that $e_1 \vee e_2 = e_2$. ■

Therefore, in order to show that $e_1$ implies $e_2$, one only needs to show that $e_1 \wedge e_2$ is decision equivalent to $e_1$.

Even though it is not the case that every identity is valid, it is obvious that if $e_1$ implies $e_2$ and $e_1$ is valid in a decision theory, $D$, then $e_2$ is also valid. In fact, the valid identities form a filter in the sense given below.

**Lemma 4.13**

The set of valid characterizing identities, $C$, of $D = (Q, E)$ forms a filter in the sense that:

(i) if $e_1 \in C$ and $e_1 \Rightarrow e_2$, then $e_2 \in C$ ;

(ii) if $e_1, e_2 \in C$ then $e_1 \wedge e_2 \in C$.

**Proof.**

(i) It has already been observed that $C$ is closed to implication.

(ii) The validity of $e_1 \wedge e_2$ has already been shown above. ■

As it will be shown next, all the identities of a finite decision theory may be represented by a single characterizing term.

**Definition 4.14**

Let $D = (Q, E)$ be a decision theory. A characterizing term, $e$, is a *characteristic term for D* if $e$ is valid in $D$ and $e$ implies every identity of $E$.

**Lemma 4.15**

If $e_1$ and $e_2$ are characterizing terms, then $e_1 \wedge e_2$ is a characterizing term and $e_1$ and $e_2$ are valid if and only if $e_1 \wedge e_2$ is valid.

**Proof.**

It remains to show that $e_1 \wedge e_2$ implies both $e_1$ and $e_2$.

Observe that as a result of both the absorption and commutative laws, $(e_1 \wedge e_2) \vee e_i = e_i$ for $i = 1, 2$.

The same result can be obtained by manipulating the characterizing terms and noticing that $e_1 \wedge e_2$ implies $e_2$ as the unreachable variables of $e_1$ in $e_1 \wedge e_2$ may be separated from those corresponding to the copies of $e_2$. The former can then be uniformly substituted by $e_2$ to obtain $e_2$ by idempotence. Similarly, it implies $e_1$ since the unreachable variables on the copies of $e_2$ in $e_1 \wedge e_2$ can be substituted by the reachable variable $T$. Idempotence then gives $e_1$. ■

**Corollary 4.16**

Every decision theory, $D = (Q, E)$, in which $E$ is finite has a characteristic term.

**Proof.**

Let $E = \{t_{11}=t_{12},\ldots,t_{n1}=t_{n2}\}$. First, transform each identity into a characterizing identity, from which its corresponding characterizing term can be obtained, to produce $E' = \{e_1,\ldots,e_n\}$. Then, form

$$e_1 \wedge e_2 \wedge \ldots \wedge e_n.$$

This is clearly a characteristic term for $D$. ■

Thus, the algebra of characterizing identities forms a lattice which looks like the one shown in Figure 4.1.

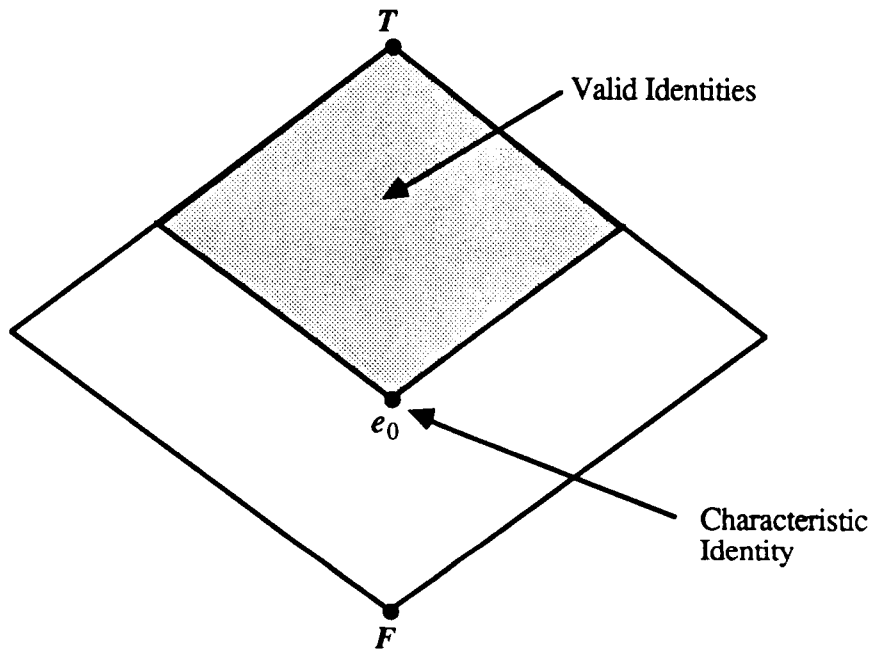



**Figure 4.1** Lattice of characterizing identities.

**Example 4.17**

Consider the identities

$$q\langle x_1 \,'\, x_2 \,'\, x_2\rangle = q_1\langle x_2 \,'\, x_1\rangle,$$
$$q\langle x_1 \,'\, x_2 \,'\, x_1\rangle = q_2\langle x_2 \,'\, x_1\rangle.$$

As characterizing terms, these are respectively:

$$q\langle q_1\langle F \,'\, T\rangle' q_1\langle T \,'\, F\rangle' q_1\langle T \,'\, F\rangle\rangle = e_1,$$
$$q\langle q_2\langle F \,'\, T\rangle' q_2\langle T \,'\, F\rangle' q_2\langle F \,'\, T\rangle\rangle = e_2.$$

An equivalent characterizing term for these two is:

$$id(rp(e_1 \wedge e_2)) = q\langle q_1\langle F \,'\, q_2\langle F \,'\, T\rangle\rangle'$$
$$q_1\langle q_2\langle T \,'\, F\rangle' F\rangle'$$
$$q_1\langle q_2\langle F \,'\, T\rangle' F\rangle\rangle. \quad \blacksquare$$

The value of compiling all the identities into one big characteristic term is questionable. Many small identities are often easier to handle. However, if one is going to find a good representation of the identities for subsequent manipulations then the ability to amalgamate in this manner is crucial.

## 4.3 Equality of Terms

As discussed in Chapter 2, equality in a decision theory without identities is determined by placing one term on the leaves of the other and then simply reducing over the first term. This procedure will not always work in a decision theory which has identities.

First, the idea of putting one tree on the leaves of the other is examined more closely. In order to show that the identity $t_1 = t_2$ holds in some decision theory, it is necessary to show that the following identity holds:

$$t_1/t_2 = t_1$$

**Definition 4.18**

An *entailment identity* is any identity $t_1 = t_2$ where $t_2$ is structurally identical to $t_1$ except that the leaves have been replaced by subtrees or variables.

As for characterizing identities, a convenient notation for this is introduced here since the tree on the right-hand side of the identity is structurally identical to the start of the tree on the left-hand side. Thus, the identity

$$q_1\langle q_2\langle x_1 \,'\, x_2\rangle \,'\, x_1\rangle =_d q_2\langle x_1 \,'\, q_1\langle x_2 \,'\, x_1\rangle\rangle$$

can be written in the form $t_1/t_2 = t_1$ and as an entailment identity as follows

$$\begin{aligned} q_1\langle q_2\langle x_1 \Rrightarrow\ & q_2\langle x_1 \,'\, q_1\langle x_2 \,'\, x_1\rangle\rangle \,'\\ & x_2 \Rrightarrow\ q_2\langle x_1 \,'\, q_1\langle x_2 \,'\, x_1\rangle\rangle\rangle \,'\\ & x_1 \Rrightarrow\ q_2\langle x_1 \,'\, q_1\langle x_2 \,'\, x_1\rangle\rangle\rangle \end{aligned}$$

Where the operator, $\Rrightarrow$, is called the *entailment operator* and, $x \Rrightarrow t$, means that, on the right-hand side of the identity, the tree ends with the variable, $x$, while,on the left-hand side of the identity the tree continues with the subtree $t$. The variable, $x$, is called the *entailing variable*. The tree t is called the *entailed tree*. In this case it is clear that repeat reductions of the identity will give:

$$q_1\langle q_2\langle x_1 \Rrightarrow x_1 \,'\, x_2 \Rrightarrow x_2\rangle \,'\, x_1 \Rrightarrow q_2\langle x_1 \,'\, x_1\rangle\rangle$$

Arriving at this identity, in which each entailed tree has all the variables at its leaves equal to the entailing variable, and applying idempotency, the equality of the terms is proved. In the absence of identities this is a necessary and sufficient condition for equality [8]. In the presence of identities, however, it is necessary not only to repeat reduce but also to check the unreachable branches in order to ascertain whether an entailment identity is valid.

**Definition 4.19**

The *ith* leaf of a tree, $t$, is *unreachable* in a decision theory $D = (Q, E)$, if replacing the variable, $x_i$, at that leaf of $t$ with $F$ and all other variables with $T$ results in a valid characterizing term in $D$.

It is readily seen that this algebraic definition of unreachable agrees with the semantic one, given in [9], which says a leaf is unreachable if for every model it classifies the empty set. The problem which is now addressed concerns how to determine whether a leaf is unreachable.

**Lemma 4.20**

A leaf of a term, $t$, is unreachable if and only if there is a valid characterizing term which when substituted idempotently at that leaf in $t$, can be repeat reduced to an unreachable leaf.

**Proof.**

Start with the term $t[x \leftarrow e\langle F\ 'x\rangle]$, in which any occurrence of a variable, $x$, in $t$ is substituted by $e\langle F\ 'x\rangle$. If under repeat reduction all the leaves below that leaf have an unreachable then idempotent reduction can deposit the variable $F$ on the leaf. Thus the leaf is unreachable.

Conversely, if the leaf is unreachable, then by uniformly substituting all the other variables by $T$, a characterizing term which has the property of the lemma is obtained. ■

The lemma does not provide a direct way of finding unreachable leaves, however, the following result gives the key to an algorithm.

**Lemma 4.21**

If $e_1$ and $e_2$ are valid characterizing terms in $D = (Q, E)$ and $e_1 \Rightarrow e_2$ then, if $t[x_i \leftarrow\!\!\rightarrow e_2]$ makes the ith leaf of $t$ unreachable, then $t[x_i \leftarrow\!\!\rightarrow e_1]$ also makes ith leaf of $t$ unreachable.

**Proof.**

If $e_1 \Rightarrow e_2$ then there is a substitution of $e_1$ that can be used to obtain a term decision equivalent to $e_2$. Since $e_1 \vee e_2 = e_1[F \leftarrow\!\!\rightarrow e_2] = e_2$ is a valid identity, it is clear that $T$ must be substituted by a term which when reduced has no occurrence of $F$. $F$ must be substituted by a term in $F, T$. This means that after the substitution, if the leaf is unreachable then it must have been before. ■

**Corollary 4.22**

The ith leaf of a term, $t$, is unreachable in $D = (Q, E)$ if and only if there is a finite subset $\{e_1,\ldots,e_n\}$ of $E$ such that $t[x_i \leftarrow\!\!\rightarrow e_1 \wedge \ldots \wedge e_n]$ makes the leaf unreachable.

**Proof.**

Every valid characterizing term is implied by such a join of terms. ■

This provides a mechanism for determining whether a leaf is reachable. This can now be put to use in determining the equality of two terms $t_1$ and $t_2$. As above, the entailment identity can be formed by $t_1 = t_1/t_2$. The idea of the algorithm is to eliminate bad leaves

from this entailment identity. It fails when a bad leaf cannot be eliminated. A bad leaf is a leaf of an entailed tree in an entailment identity, $t_1/t_2$, whose variable is not the same as its entailing variable. If there are no bad leaves, idempotence gives the identity and $t_1$ is decision equivalent to $t_2$. If there are bad leaves but they are all unreachable, they may be freely substituted by any variable. Thus, in particular, the entailing variable may be substituted. This substitution allows us to idempotent reduce and to prove the identity again. It is clear that if this procedure works that $t_1$ will be equivalent to $t_2$. For the converse, if there is a bad leaf and $t_1=t_2$ is a valid identity, then it suffices to show that the leaf is unreachable.

Notice that if $e$ is a characterizing term equivalent to $t_1=t_2$ then the bad leaves in the identity $t_1 = t_1/t_2$ correspond to the unreachable branches of $e$. The problem of deciding if two trees are decision equivalent in a decision theory is therefore equivalent to that of determining if the corresponding characterizing term is valid under [D.1]-[D.3] and the identities particular to the theory.

This gives the following result:

**Proposition 4.23**

In a decision theory, $D = (Q, E)$, with characteristic term $e'$, the identity $t_1=t_2$, corresponding to the characterizing term $e$, is valid if and only if $e \vee e' = e$.

**Proof.**

The characterizing term $e$ is valid in the decision theory $D = (Q, \{e'\})$ if $e$ can be implied by $e'$. It has already been shown that $e' \Rightarrow e$ if and only if $e \vee e' = e' \vee e = e$. Thus the desired result is obtained. ■

This result provides a method for deciding when two trees are decision equivalent in the presence of a set of identities. The first step for determining if the trees $t_1$ and $t_2$, are

decision equivalent, is to construct a characterizing term, $e$, equivalent to the identity $t_1 = t_2$. The next step is to obtain the characteristic term, $e' = e_1 \wedge \ldots \wedge e_n$, by amalgamating the set of characterizing terms, $e_1,\ldots,e_n$, respectively equivalent to the identities $t_{11}=t_{12},\ldots,t_{n1}=t_{n2}$ in the theory. Finally, the identity $e \vee e'$ can be obtained by replacing the unreachable leaves of $e$ with $e'$, and if the result of simply reducing this term results in the same tree, $e$, then both $t_1$ and $t_2$ are decision equivalent under the decision theory.

**Example 4.24**

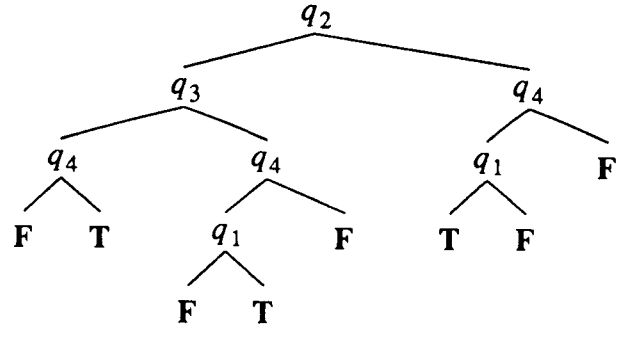
(i) Consider the decision theory $D = (\{q_1, q_2, q_3, q_4\}, E)$, where $E = \{q_1\langle x_1\ ' x_2\rangle = q_2\langle x_2\ ' x_1\rangle, q_3\langle x_1\ ' x_2\rangle = q_4\langle x_2\ ' x_1\rangle\}$. Now consider whether the following decision trees

$$t_1 = q_2\langle q_3\langle x_2\ ' x_3\rangle\ ' q_4\langle x_1\ ' x_2\rangle\rangle \text{ and}$$
$$t_2 = q_4\langle q_1\langle x_1\ ' x_3\rangle\ ' x_2\rangle$$

are decision equivalent.

To decide this, the characterizing term corresponding to the identity $t_1 = t_2$ is obtained:

$e$ =




Next, it is necessary to put the identities of the theory into characterizing form. In this form, they are respectively:

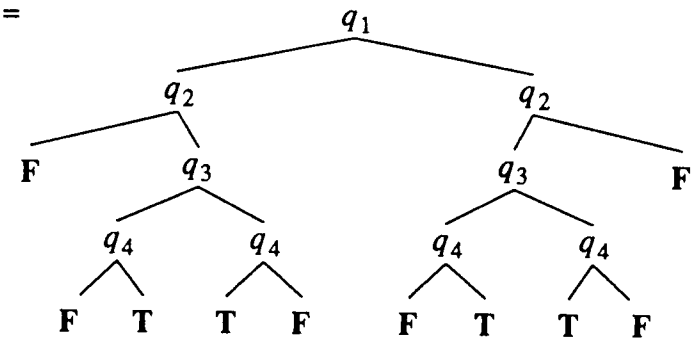
$$e_1 = q_1\langle q_2\langle F\ 'T\rangle\ 'q_2\langle T\ 'F\rangle\rangle \quad \text{and}$$

$$e_2 = q_3\langle q_4\langle F\ 'T\rangle\ 'q_4\langle T\ 'F\rangle\rangle$$

Then the characteristic identity is formed

$$e' = e_1 \wedge e_2$$

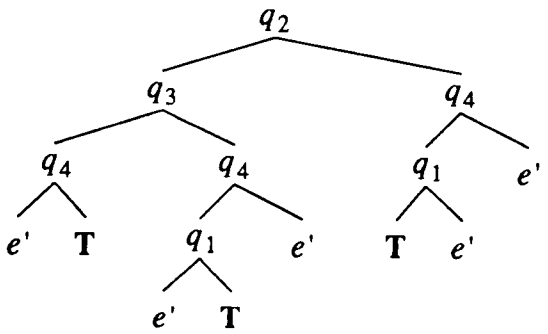
$e'$ =




Finally, by substituting the variable $F$ in $e$ by $e'$, it follows that

$$e \vee e' = e[F \leftarrow e']$$

$e \vee e' =$



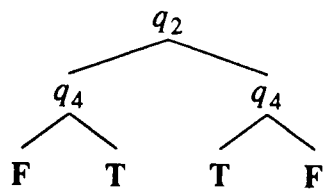

which, by simple reduction, can be shown to reduce to the term $e$.

Therefore, these two trees are indeed equal in this theory.

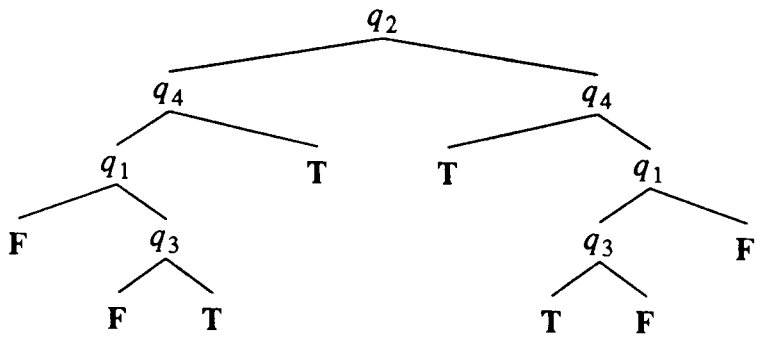
(ii) Testing the trees

$$q_2\langle x_1\ 'x_2\rangle \text{ and } q_4\langle x_2\ 'x_1\rangle$$

in the same theory, results in the characterizing term




Replacing the unreachable leaves with $e'$ above and simply reducing gives




As this tree can not be reduced to the characterizing term, it is concluded that the decision equivalence of the trees in question does not hold in this theory. ■

The procedure just described can also be used for deciding decision equivalence between terms in a decision theory which has no identities. In such case, the characteristic term is the top of the lattice, $T$. Since $e_1 \vee T = T$ for any characterizing term $e_1$, the above procedure will always yield the correct result as the equality $e \vee T = e$ can only be obtained when the characterizing term $e$ does not have any unreachable leaves. In other words, the equality will not hold if there are any bad leaves in the corresponding entailment identity.

In the notation for valid entailment identities it is useful to allow the omission of unreachable branches. Thus, the notation for characterizing terms can be freely mixed with that for an entailment identity. For example this means that instead of writing the entailment $x_1 \Rightarrow\!\!\!\!\gg x_2$, the branch is simply omitted. But this is not the only place this can occur as all the branches which lead only to bad leaves in entailed trees may be omitted.

# CHAPTER 5

# DECISION CLUSTERS

Decision trees are often viewed as a form of representation ideally suited for problems where efficiency is a critical factor. However, when viewed as an alternative representation for knowledge, decision trees have the characteristic of embedding both logic and control information in their structure. This has the advantage of suggesting an explicit implementation of the solution to the problem. Unfortunately, it also has the adverse effect of making it difficult to separate the logical from the procedural knowledge [14]. The arrival of expert systems technology has served to demonstrate the many advantages of keeping the logical specification of the knowledge separate from its use. Yet, the use of a standard inference engine to generate the procedural implementation of rule-based systems has been shown to be inadequate, since they alone cannot provide the means for implementing arbitrary control sequencings [13].

A different approach to the generation of a procedural form from some logical specification (production rules) is to use backward chaining to obtain an equivalent decision tree representation. Once in a decision tree form, the manipulations presented in the previous chapters can be used to generate the other procedural representations of the solution. This approach provides the ability to separate the knowledge from its use while also allowing the generation of an efficient implementation. In particular, using Chen's notion of knowledge representation trees, which allows rules giving dependencies between attributes to be incorporated into a decision tree, gives a useful representation for knowledge [5].

A common problem encountered when building rule-based knowledge systems is that, in general, there is no easy way of deciding when a set of rules forms a complete logical specification of the problem without taking into consideration the procedure needed to use the knowledge in the rules. A decision tree representation of the problem can be useful in determining if the logical information in the rules is complete. In the absence of dependencies between the decisions in the problem, one only needs to test that the decision tree representation has no unreachable leaves. However, when arbitrary dependencies are allowed, the test is more involved. The problem is that of deciding if a decision expression is retrieval complete for some set of decisions. A solution to the problem of obtaining the retrieval of an arbitrary term in a discrete decision theory with identities was developed by this research. This is presented in this chapter along with a method for deciding when a decision expression is retrieval complete.

In general, a decision expression can be viewed as an independent cluster of knowledge which combines both logical and procedural information about a problem in addition to satisfying a set of arbitrary decision identities. Thus, a decision cluster can then be considered as a functional specification for some decision problem. A decision expression is functional if it can be used to determine the value of some decision in terms of another set of decisions. However, given a logical specification in terms of a collection of rules, a common task is that of determining which of those rules are relevant to the decision-making process and which are not. It is clear that only those decisions which have any bearing on the outcome of a process should be included in a decision cluster. This is the motivation for developing the notion of minimally retrieval complete knowledge clusters which is presented in this chapter. Finally, a set of syntactic rules developed by the author are presented in this chapter with the aim of manipulating decision clusters while preserving the semantic integrity of the overall decision expression.

The chapter is organized in four sections. The first section introduces the concepts of functional identities and decision clusters. The second section introduces a set of syntactic rules that provide the framework from which the algorithms for manipulating decision trees that were presented in Chapter 3 can be extended to decision clusters. The third section presents the idea of the q-retrieval of a conjunction of branch instructions on a term and discusses its properties. Finally, the last section introduces retrieval completeness and the use of decision clusters as self-contained modules of knowledge.

## 5.1 Functional Identities and Decision Clusters

Decision clusters are decision expressions with special retrieval properties. Therefore, it is appropriate to begin this section with a general discussion about the retrieval of terms in a discrete decision theory. The discussion leads into the treatment of identities and their retrieval properties. In particular, identities of a special form, called functional, are given special consideration, and are later shown to correspond to decision clusters.

One of the fundamental results presented in [10] was that in the absence of any other decision identities but [D.1]-[D.3], two terms in a decision theory are decision equivalent if and only if they have the same retrieval properties. In the absence of identities, the retrieval properties of decision trees are defined as follows [8]:

**Definition 5.1**

(i) If $B$ is the conjunction of branch instructions, $q_1 \diamond a_1 \wedge \ldots \wedge q_n \diamond a_n$, then a spine $S = (P, z)$ is said to *agree* with $B$ if whenever $q \diamond x$ occurs in $B$ and $q \diamond y$ occurs in $P$, then $x = y$.

(ii) If $t$ is a term and $B$ is a conjunction of branch instructions, as above, then the *simple retrieval* of $B$ in $t$, denoted $ret(B, t)$, is defined as follows,

$$ret(B, t) = \{x \mid S = (P, x) \text{ is a spine in } t, \text{ and } S \text{ agrees with } B\}.$$

Unfortunately, it is not clear how this definition applies to terms in a decision theory with identities. In particular, two equivalent terms in the theory may yield rather different retrievals as some of the leaves may actually be unreachable. One way to solve this problem is to develop a more comprehensive definition for the retrieval.

**Definition 5.2**

The *retrieval* of a conjunction of branch instructions, $B$, in a term, $t$, in a decision theory, $D = (Q, E)$, is the intersection of all the simple retrievals of $B$ on terms decision equivalent to $t$ in $D$.

By the retrieval lemma presented in [10], this definition is equivalent to the original notion of retrieval for decision theories in which $E$ is empty. However, this definition is unsatisfactory as the retrieval has now become rather difficult to calculate. The basic idea of the following discussion is that if the unreachable leaves are all marked in a term, then by ignoring those leaves, more reasonable retrieval results may be obtained. However, as shown below, this approach alone does not solve the retrieval problem.

**Definition 5.3**

Let $D = (Q, E)$ be a decision theory and $q$ be a decision in $Q$ with arity $n$ and codomain $\{a_1, ..., a_n\}$. Then,

(i) Given a term, $t$, such that $q$ does not occur in $t$, then a *function for* $q$ is an identity of the form

$$t = q\ \langle x_1\ '\ ...\ '\ x_n \rangle,$$

where the variables, $x_1$, ..., $x_n$, are distinct.

(ii) An identity, $e$, is *functional on* $q$, if there exists a function for $q$ that can be derived from $e$.

(iii) If $t = q\langle x_1 \text{ '} \ldots \text{ '} x_n\rangle$ is a function for $q$ and $B$ is a conjunction of branch instructions, then the *functional retrieval* of $B$ in a function for $q$ is defined as $\{a_i \mid x_i \in ret(B, t) \cap \{x_1, \ldots, x_n\}\}$.

For instance, the identity

$$q_1\langle q_2\langle x_1 \text{ '} x_3\rangle \text{'} x_2\rangle = q_0\langle x_2 \text{ '} x_1\rangle.$$

is a function for $q_0$. The identity may be read as follows: to decide $q_0$ decide $q_1$, if $q_1$ is taken in its first argument, then decide $q_2$, if $q_2$ is taken in its first argument, then as $x_1$ is reached on the LHS of the identity, one should conclude that the decision $q_0$ must be taken in its second argument. Similarly, if $q_1$ is taken in its second argument, then one should conclude that the decision $q_0$ must be taken in its first argument. Thus, this kind of identity provides a functional relationship between $q_0$ and the set of decisions $\{q_1, q_2\}$. Note, however that if the decision $q_1$ is decided in its first argument and $q_2$ is decided in its second argument, then the value for $q_0$ cannot be decided.

Functional relationships given in this form are of limited practical value due to the relatively high cost involved in extracting a function from an arbitrary identity. The manipulation of these functional relationships can be facilitated by the introduction of decision clusters.

**Definition 5.4**

Let $D = (Q, E)$ be a decision theory.

(i) A decision $q$ is said to be *reachably semi-essential* in a term, $t$, if $q$ occurs on every reachable path in $t$.

(ii) A decision $q$ is said to be *definite* in a term, $t$, if at every occurrence of $q$ in $t$, at most one of its arguments leads to a reachable leaf.

(iii) A characterizing term $e$ is said to be a *decision cluster* on $q$ in $\{q_1, \ldots, q_n\}$, if $e$ is a term in $\{q, q_1, \ldots, q_n\}$ and $q$ is both reachably semi-essential and definite in $e$.

For instance, the functional identity,

$$q_1\langle q_2\langle x_1 \,'\, x_3\rangle' \, x_2\rangle = \; q_0\langle x_2 \,'\, x_1\rangle,$$

has a corresponding characterizing term of the form:

$$q_1\langle q_2\langle q_0\langle F \,'\, T\rangle' \, q_0\langle F \,'\, F\rangle\rangle' \, q_0\langle T \,'\, F\rangle\rangle,$$

which can be idempotently reduced to produce:

$$q_1\langle q_2\langle q_0\langle F \,'\, T\rangle' \, F\rangle' \, q_0\langle T \,'\, F\rangle\rangle.$$

In the latter form, the identity becomes a decision cluster on $q_0$ in $\{q_1, q_2\}$, since $q_0$ is both reachably semi-essential and definite. In fact, the following lemma shows that for each functional identity, there exists an equivalent decision cluster and vice versa.

**Lemma 5.5**

Given a decision $q$ in a decision theory $D = (Q, E)$, then every function for $q$ involving only the decisions, $q, q_1, \ldots, q_n$, is equivalent to a decision cluster on $q$ in $\{q_1, \ldots, q_n\}$.

**Proof.**

($\Rightarrow$) Given a function for $q$ of the form

$$t = q\,\langle x_1 \,'\, \ldots \,'\, x_n\rangle,$$

then an equivalent decision cluster can be found by forming the corresponding characterizing term. Recall that such a characterizing term has the form: $t / q\langle x_1'...'x_n\rangle$. Since the variables $x_1, ..., x_n$ are all distinct, it is obvious that the corresponding characterizing term has $q$ reachably semi-essential and definite.

($\Rightarrow$) Conversely, given a decision cluster on $q$ in $\{q_1, ..., q_n\}$, $C$, one can decision delay it by $q$ to produce an equivalent cluster $C'$. Since $q$ is reachably semi-essential and definite in $C$, then it must also be reachably semi-essential and definite in $C'$. A function for $q$ can then be formed as follows: The *RHS* of the function is a decision tree one level deep, with $q$ at the root and with a different variable $(x_1, x_2, ...)$ assigned to each argument of $q$. The *LHS* of the function has the same structure as $C'$, except that, the variable $x_0$ is assigned to every unreachable path in $C$, and every occurrence of a sub-term with $q$ at the root in $C'$ is replaced instead by that variable $x_i$, from the *RHS* of the function, which is associated with the argument of $q$ that is on a reachable path (recall that $q$ is definite in $C'$). ■

In fact, as the next result shows, the notion of a decision cluster provides a method for determining whether or not a given identity is functional.

**Proposition 5.6**

A characterizing term, $e$, is functional on $q$, if and only if, $e$ decision delayed by $q$ results in a term in which $q$ is reachably semi-essential and definite.

**Proof.**

($\Rightarrow$) If a characterizing term, $e$, is functional on $q$ then there exists a function for $q$ which can be derived from $e$. According to Lemma 5.5, there exists an equivalent decision cluster on $q$. By definition, the corresponding decision cluster has $q$ reachable semi-essential and definite.

($\Leftarrow$) Since $e$ is a characterizing term, the result of decision delaying $e$ by $q$ is another characterizing term. If $q$ is both reachably semi-essential and definite in the resulting term, then the term is also a cluster on $q$. Thus, $e$ is functional on $q$. ■

This provides a test for determining whether or not a given identity is functional on some decision $q$. First, obtain the corresponding characterizing term. Then decision delay $q$ in the term. If $q$ occurs on every reachable path of the resulting term and it is definite at every occurrence, then the original identity is functional on $q$.

An alternative way of writing the identity above is as follows:

$$q_0 :\!> \quad q_1\langle q_2\langle 2\ '\ _\rangle\ '\ 1\rangle,$$

where $q_0$ is called the *goal decision* and the variable at each leaf of the term indicates which argument of $q_0$ is to be concluded at the end of that path. The underscore indicates that $q_0$ can not be decided for that combination of decisions. Notice that the branch marked with the underscore corresponds to the unreachable leaf in the decision cluster which is on a path where $q_0$ does not occur. These unreachable leaves can be removed from the tree without any loss of information. The result is a practical representation for expressing the functional relationship between the goal decision and the decisions in a cluster.

In fact, any decision tree can be viewed as a decision cluster in which the goal decision corresponds to the function that the decision tree solves and each variable encountered at the end of a leaf corresponds to the value that the goal decision assumes as a function of the other decisions in the tree.

Obviously, there exists a very strong relationship between functions, functional identities and decision clusters on the same decision $q$. Therefore, to evaluate the functional retrieval of a conjunct on any one of these expressions one only needs to perform the retrieval on the corresponding function for $q$.

## 5.2 Manipulation of Decision Clusters

The goal of this section is to present a set of decision identities for manipulating decision clusters. Since these decision identities are derived from the identities [D.1]-[D.3], they preserve the retrieval properties of any decision expression. Not surprisingly, the main difference between both sets of identities lies in the treatment of unreachable branches. Two of the identities introduced in this section, migration (weak idempotence) and reachable transposition, are of particular importance. The purpose of these identities is to provide the framework from which a set of algorithms for manipulating decision clusters can be developed.

### 5.2.1 Migration or Reachable Idempotence

Recall the notation used to describe entailing identities. Given a variable, $x$, and a term, $t$, such that all the reachable paths in $t$ arrive at the same conclusion $x$, then a *deduction node* in a term, $t'$, is an identity at a leaf of $t'$, which has the form $x \Rightarrow t$.

The presence of deduction nodes in a decision tree partitions the term into two distinct portions.

**Definition 5.7**

Let W be a term in a decision theory, $D = (Q, E)$, and let $t$ be a decision tree of the form

$$t = \mathrm{W}\langle x_1 \Rrightarrow t_1 \,'\, \ldots \,'\, x_n \Rrightarrow t_n \rangle$$

then:

(i) The *antecedent subtree* of $t$, denoted by ant($t$), is the term above the deduction nodes which has the form:

$$\mathrm{ant}(t) = \mathrm{W}\langle x_1 \,'\, \ldots \,'\, x_n \rangle.$$

(ii) The *conclusion subtree* at the ith leaf of $t$, is the term, $t_i$, below the deduction node at that leaf, such that every reachable path in $t_i$ arrives at the conclusion $x_i$.

Figure 5.1 shows the different parts of a decision cluster. The variable $x$ in the deduction node $x \Rrightarrow t$ will be referred to as either a *conclusion* or an entailing variable depending on whether in the discussion the variable is viewed from the antecedent subtree or from its entailed conclusion subtree respectively.

Once a deduction node, $x \Rrightarrow t$, is reached in a decision cluster, it can be concluded that the value for the goal decision is $x$. In addition, given an antecedent path, $P$, leading to a deduction node, $x \Rrightarrow t$, then the reachable paths in the conclusion subtree $t$ define the only possible combinations of answers to the set of questions in $t$ that are consistent with the conjunction of branch instructions in $P$.

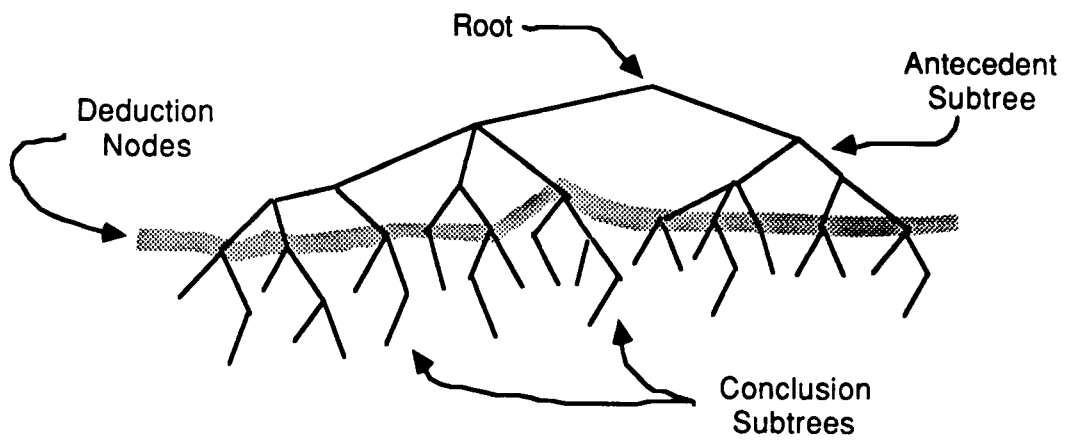



**Figure 5.1** A decision cluster and its components: deduction nodes, antecedent subtree and conclusion subtrees.

A decision identity for manipulating deduction nodes in decision clusters is now introduced. The identity, which will be called *migration*, is similar to idempotence but its effect is somewhat weaker.

**[DR.1]** (Migration or Reachable Idempotence)

$$q\langle x \Rightarrow t_1' \ldots ' x \Rightarrow t_{i-1}' _ ' x \Rightarrow t_{i+1}' \ldots ' x \Rightarrow t_n \rangle$$
$$= x \Rightarrow q\langle t_1' \ldots ' t_{i-1}' _ ' t_{i+1}' \ldots ' t_n \rangle .$$

The migration identity can be used to reduce the length of reachable paths in the antecedent subtree of a decision cluster. The purpose of migrating a conclusion variable closer to the root of the term is thus to try to resolve the value for the goal decision in the cluster as early as possible in the decision process. This procedure results in the following rewriting rule.

*Migration Law:* Let $D = (Q,E)$ be a decision theory with $W$ a term in $Q$ and $x$ a conclusion variable. If the term $W$ has arity $n$, and every term $t_k$, $1 \leq k \leq n$, has every reachable path leading to the variable $x$, then:

$$W\langle x \Rightarrow t_1' \ldots 'x \Rightarrow t_{i-1}'_'x \Rightarrow t_{i+1}' \ldots 'x \Rightarrow t_n\rangle$$
$$\Downarrow$$
$$x \Rightarrow W\langle t_1' \ldots 't_{i-1}'_'t_{i+1}' \ldots 't_n\rangle$$

In other words, the migration law states that if all the reachable paths of a term in a decision theory terminate in the same conclusion variable, then the antecedent portion of that term can be replaced by a deduction node with that conclusion as the entailing variable.

**Definition 5.8**

A decision tree, $t$, in a decision theory, $D = (Q, E)$, is *conclusion migrated*, if there is no path in $t$ which has a conclusion variable that can be migrated towards the root by means of the migration law.

**Lemma 5.9**

The migration law is a confluent terminating rewriting.

**Proof.**

To show that the rewriting terminates, notice that the path bags of the antecedent subtree are reduced every time the migration law is applied to migrate a conclusion variable towards the root of the term. Since a term in a discrete decision theory has a finite number of path bags, all of which have finite length, after a finite number of rewritings, the migration law can no longer be applied.

To show that the rewriting is confluent, it suffices to observe that the migration law does not change the path bags of the term, it only changes the location of the deduction nodes along the paths. Thus, when the migration terminates, it results in a unique antecedent subtree, followed by a unique set of conclusion subtrees. ■

### 5.2.2 Reachable Repetition, Distribution and Transposition

In this section, a set of decision identities suitable for the manipulation of unreachable leaves is derived from the decision identities [D.1]-[D.3]. The purpose of these identities is to allow the manipulation of decision clusters while preserving their retrieval properties.

**[DR.2]** (Reachable Distribution)

$$q_1\langle x_1 '\ldots ' x_{i-1} ' q_2\langle y_1 '\ldots ' y_{j-1} '_ ' y_{j+1} '\ldots ' y_m\rangle ' x_{i+1} '\ldots ' x_n\rangle$$
$$= q_2\langle q_1\langle x_1 '\ldots ' x_{i-1} ' y_1 ' x_{i+1} '\ldots ' x_n\rangle '\ldots '$$
$$q_1\langle x_1 '\ldots ' x_{i-1} ' y_{j-1} ' x_{i+1} '\ldots ' x_n\rangle ' q_1\langle x_1 '\ldots ' x_{i-1} '_ ' x_{i+1} '\ldots ' x_n\rangle '$$
$$q_1\langle x_1 '\ldots ' x_{i-1} ' y_{j+1} ' x_{i+1} '\ldots ' x_n\rangle '\ldots ' q_1\langle x_1 '\ldots ' x_{i-1} ' y_m ' x_{i+1} '\ldots ' x_n\rangle\rangle .$$

**[DR.3]** (Reachable Repetition)

$$q\langle x_1 '\ldots ' x_{i-1} ' q\langle y_1 '\ldots ' y_n\rangle ' x_{i+1} '\ldots ' x_{j-1} ' q\langle \ldots ' z_{j-1} '_ ' z_{j+1} '\ldots\rangle ' x_{j+1} '\ldots ' x_n\rangle$$
$$= q\langle x_1 '\ldots ' x_{i-1} ' y_i ' x_{i+1} '\ldots ' x_{j-1} '_ ' x_{j+1} '\ldots ' x_n\rangle .$$

**[DR.4]** (Reachable Transposition)

$$q_1\langle q_2\langle x_{11} '\ldots ' x_{1m}\rangle '\ldots ' q_2\langle x_{i-1,1} '\ldots ' x_{i-1,m}\rangle '_ '$$
$$q_2\langle x_{i+1,1} '\ldots ' x_{i+1,m}\rangle '\ldots ' q_2\langle x_{n1} '\ldots ' x_{nm}\rangle\rangle$$
$$= q_2\langle q_1\langle x_{11} '\ldots ' x_{i-1,1} '_ ' x_{i+1,1} '\ldots ' x_{n1}\rangle '\ldots '$$
$$q_1\langle x_{1m} '\ldots ' x_{i-1,m} '_ ' x_{i+1,m} '\ldots ' x_{nm}\rangle\rangle .$$

The effect of [DR.2] is the same as that of [D.2] if the unreachable branches in the term are expanded and labeled with a common symbol, say $F$. Similarly, the effect of [DR.3] is the same as that of [D.3]. Basically, the repeat reduction rewriting rule can be generated from either one as they both state that once the value of a decision has been determined along a (reachable) path any other occurrence of the same decision must yield the same result. Note that the application of the repeat reduction rule can cause the discovery of unreachable paths, which must then be eliminated from the term. Intuitively, all time-

invariant decision processes must always satisfy the repetition identities in order to guarantee the consistency of the system.

Finally, reachable transposition, [DR.4], can be shown to be a consequence of the commutative property of the join operator "$\wedge$" on terms in a decision theory. It is obvious that [DR.4] implies [D.4] when the terms have no unreachable paths. Observe, however, that [DR.4] cannot be derived from [D.4] independently. Nevertheless, if one replaces all the unreachable leaves in a term with the variable $F$ (assuming of course that the variable $F$ does not occur on any reachable path), then it can be shown that [D.1] together with [D.4] do imply [DR.4].

It is observed, however, that idempotence ([D.1]) is a rather strong decision identity that must always be present in any axiom scheme used. Thus, the following result:

**Lemma 5.10**

Given a decision theory, $D = (Q, E)$, the decision identities

[D.1], [DR.4] and [D.3]

form a complete set.

**Proof.**

The set of decision identities [D.1], [D.2] and [D.3] was shown to be complete by Cockett in [9].

Similarly, the fact that [D.1], [D.4] and [D.3] is equivalent to the set [D.1], [D.2] and [D.3] was proved in [9, 10].

Thus, to prove this lemma it suffices to show that

[D.1], [DR.4] and [D.3] (1)

is equivalent to

[D.1], [D.4] and [D.3]. (2)

This is shown by proving that each scheme can be derived from the other. To derive (2) from (1), just observe that [D.4] is implied in an obvious way from [DR.4]. To show that (1) can be implied by (2), notice that [D.1] can be applied to the unreachable paths in the LHS of [DR.4] to expand those paths and make the second decision semi-essential in the term. Applying the transposition identity to that result, followed by the collapse of any unreachable paths yields the RHS of [DR.4]. Therefore, the scheme [D.1], [DR.4] and [D.3] is equivalent to the scheme [D.1], [D.4] and [D.3]. ■

The application of the identity [DR.4] to terms in a decision theory with identities provides the means for defining the *reachable transposition law.*

*Reachable Transposition Law*: Given two terms, $W_1$ and $W_2$, in a decision theory $D = (Q, E)$, then

$$W_1\langle W_2\langle x_{11}, \ldots, x_{1m}\rangle, \ldots, W_2\langle x_{i-1,1}, \ldots, x_{i-1,m}\rangle, _,$$
$$W_2\langle x_{i+1,1}, \ldots, x_{i+1,m}\rangle, \ldots, W_2\langle x_{n1}, \ldots, x_{nm}\rangle\rangle$$
$$\Updownarrow$$
$$W_2\langle W_1\langle x_{11}, \ldots, x_{i-1,1}, _, x_{i+1,1}, \ldots, x_{n1}\rangle, \ldots,$$
$$W_1\langle x_{1m}, \ldots, x_{i-1,m}, _, x_{i+1,m}, \ldots, x_{nm}\rangle\rangle.$$

The terms $W_1$ and $W_2$ must appear on every reachable path in the term before the reachable transposition law can be applied. Notice that if the ith argument of $W_1$ is on an unreachable path, then the transposed term will have the ith argument of $W_1$ unreachable from every argument of $W_2$. Moreover, if the jth argument after $W_2$ is unreachable from the kth argument of $W_1$, then, when $W_1$ and $W_2$ are transposed, the kth argument of $W_1$ is unreachable from the jth argument of $W_2$.

**Definition 5.11**

Two terms, $t_1$ and $t_2$, in a decision theory $D = (Q, E)$ are *reachably transposition equivalent*, denoted $t_1 =_{rt} t_2$, if $t_2$ can be obtained from $t_1$ after a finite number of rewritings using the reachable transposition law.

**Lemma 5.12**

If $q$ is reachably semi-essential in a tree, $t$, then there exists a tree, $t'$, with $q$ at the root, such that $t'$ is reachably transposition equivalent to $t$.

**Proof.**

Let $W_1$ be the portion of $t$ that is in front of $q$ and let $W_2$ be equal to $q$. Since $q$ occurs on every reachable path in $t$, then the reachable transposition law can be applied to $t$ to transpose $q$ in front of $W_1$, resulting in the reachably transposition equivalent tree $t'$ with $q$ at the root. ■

Next, it is shown that *Chen and Ras's Lemma* (with a slight modification) also holds for arbitrary decision clusters. First, some concepts are introduced.

**Definition 5.13**

(i) If $P = \{q_1 \diamond a_1, \ldots, q_n \diamond a_n\}$ is the list of branch instructions corresponding to a reachable path in a term, $t$, associated with a conclusion variable, $x$, then the pair $(P, x)$ is a *reachable spine* in the term $t$.

(ii) A *reachable spine set* is the set of all the reachable spines in a term.

(iii) Two terms are *reachable bag equivalent* if, as unordered lists of branch instructions, their reachable spine sets are the same.

**Lemma 5.14 [Chen and Ras's Lemma for Identities]**

Two repeat reduced terms are reachably transposition equivalent, if and only if, their sets of reachable spines are bag equivalent.

**Proof.**

($\Leftarrow$) Assume that $t_1$ and $t_2$ have bag equivalent reachable spine sets. Let $q_1$ be the root of $t_1$ and $q_2$ be the root of $t_2$, if $q_1$ and $q_2$ are different, then, since $q_1$ occurs on every reachable path of $t_1$, it must occur on every reachable path of $t_2$ as well. Therefore, there exists a tree $t'_2$, reachably transposition equivalent to $t_2$ with $q_1$ at the root. Obviously, $t'_2$ has the same reachable spine set as $t_2$. This process can be repeated recursively on the subtrees of $t_1$ and $t'_2$, considering shorter paths at every step. Since the length of every reachable path is finite, after a finite number of reachable transposition rewritings, a term identical to $t_1$ can be obtained from $t_2$. Therefore, $t_1$ and $t_2$ are reachably transposition equivalent.

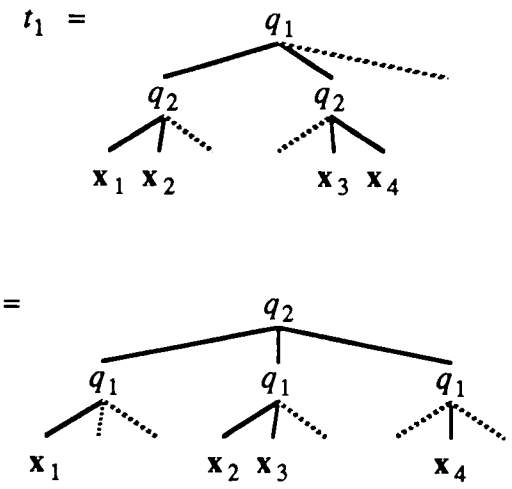
($\Rightarrow$) Assume now that $t_1$ and $t_2$ are reachably transposition equivalent. As the reachable transposition law does not alter the reachable spines, then $t_1$ and $t_2$ are reachable bag equivalent. ■

**Example 5.15**

The following two trees

$$t_1 = q_1\langle q_2\langle x_1\ 'x_2\ '_\rangle\ 'q_2\langle_\ 'x_3\ 'x_4\rangle\ '_\rangle \quad \text{and}$$
$$t_2 = q_2\langle q_1\langle x_1\ '_\ '_\rangle\ 'q_1\langle x_2\ 'x_3\ '_\rangle\ 'q_1\langle_\ 'x_4\ '_\rangle\rangle$$

are reachable transposition equivalent. This is easier to see if they are both expressed in graphical form and unreachable leaves are removed. Note that the following trees have equivalent reachable path bag sets.




■

Given the manipulations presented in this section, the algorithms for generating irreducibles can be extended to handle any decision cluster in a discrete decision theory. A modification to the basic reduction algorithm is presented in Figure 5.2.

Observe that the main difference lies in the treatment of deduction nodes. For each term in a decision cluster, the algorithm obtains an irreducible form by first reducing the subtrees. Then, it uses the reachable transposition law to decision factor the antecedent subtree by its root decision. The migration law is then used to remove any redundant occurrence of the root decision in the decision factored antecedent subtree. Finally, at each deduction node, the basic reduction algorithm is applied to find an irreducible form for the conclusion subtree.

Similarly, the algorithm for finding all the irreducible forms of a decision tree which was introduced in Chapter 3 can be extended to handle decision clusters. For this, the semi-essential leaf refinement, [S.2], is modified to generate a disjoint redundancy plan whose order depends on the number of reachable branches of the node to which it is applied. Separator plans are developed only up to the deduction nodes. Again, conclusion subtrees are reduced by the application of the basic reduction algorithm. Finally, only

those decision expressions whose antecedent subtrees are irreducible are reported by the algorithm. The extended algorithm can be used to generate all the irreducible forms of a decision cluster. Therefore, the algorithm can develop all the optimal solutions of any problem whose logical and procedural specification can be represented in terms of a functional identity.

**Algorithm 5.16 [Reduction Algorithm for Decision Clusters]**

REDUCE ($t$):

**if** $t = q\langle t_1 \text{ ' } \dots \text{ ' } t_n \rangle$

**then begin**

$r_1 \leftarrow$ REDUCE ($t_1$); ... ; $r_n \leftarrow$ REDUCE ($t_n$);

$r' \leftarrow$ ANTECEDENT DECISION FACTOR ($q\langle r_1 \text{ ' } \dots \text{ ' } r_n \rangle$);

**return** MIGRATE ($r'$);

**end**

**elseif** $t = x \Rrightarrow q\langle t_1 \text{ ' } \dots \text{ ' } t_n \rangle$

**then begin**

$r_1 \leftarrow$ REDUCE ($x \Rrightarrow t_1$); ... ; $r_n \leftarrow$ REDUCE ($x \Rrightarrow t_n$);

$r' \leftarrow$ DECISION FACTOR ($q\langle r_1 \text{ ' } \dots \text{ ' } r_n \rangle$);

**return** IDEMPOTENT REDUCE ($r'$);

**end**

**elseif** $t = x$ **then return** $t$ ;

**Figure 5.2** Algorithm for finding an irreducible form of a decision cluster

## 5.3 Retrieval Properties of Decision Clusters

The functional retrieval of a conjunction of branch instructions on a decision cluster obtains all of the possible values of the goal decision which agree with the conjunction. This means that unreachable branches should be ignored in the retrieval. Using functional retrievals, one would expect to obtain the same retrieval properties in decision clusters as those found in [10] for decision trees in discrete decision theories without identities. Recall, however, that the retrieval lemma presented in [10] is extremely strong: it says that two terms have the same retrieval if and only if they are decision equivalent. The suppression of unreachable leaves does not effect this result. The following example illustrates the type of problem posed.

**Example 5.17**

Let $D = (\{q_1, q_2, q_3\}, \{q_1\langle x\ 'y\rangle = q_2\langle y\ 'x\rangle, q_1\langle x\ 'y\rangle = q_3\langle y\ 'x\rangle\})$ be a decision theory. It follows that

$$q_1 :> q_2\langle 2\ '1\rangle \text{ and}$$
$$q_1 :> q_3\langle 2\ '1\rangle$$

are functional identities on $q_1$ as they are both decision clusters on $q_1$. However, the retrieval of $[q_3 \diamond 1]$ on the first identity results in $\{2, 1\}$ but on the second results in $\{1\}$. ■

Obviously, these results are highly unsatisfactory. Observe that, although the notion of a functional retrieval is very intuitive, it can only be defined on functional identities. Therefore, a more general definition of the retrieval as applied to identities is required. Recall that an identity can be viewed as a decision tree, i.e. as a characterizing term. Thus, the pair $(\{q_1 \diamond a_1, ..., q_n \diamond a_n\}, x)$ is defined as a *spine* of a characterizing term, $e$, if $x$ is

the variable reached at the end of a path in $e$ which agrees with the set of branch instructions $\{q_1 \diamond a_1, ..., q_n \diamond a_n\}$. A *reachable spine* is one that leads to the variable $T$. Similarly, an *unreachable spine* is one that leads to the variable $F$.

**Definition 5.18**

Given a decision theory, $D = (Q, E)$, then, the *q-retrieval* of a conjunction of branch instructions, $B$, on an arbitrary characterizing term, $e$, denoted $ret_q(B, e)$, is defined as follows:

$$ret_q(B, e) = \{a_i \mid q \diamond a_i \text{ agrees with some spine } S \in Sp_R(B, e)\}$$

where $Sp_R(B, e)$ denotes the set of reachable spines in $e$ which agree with $B$.

First, it is shown that the q-retrieval of a characterizing term is invariant under the decision identities [D.1], [DR.4] and [D.3]. To show that the q-retrieval of a characterizing identity is invariant to [DR.4], simply note that reachable transposition does not change any of the reachable spines in the term. Similarly, the q-retrieval is invariant to [D.3] since repetition does not alter the set of reachable spines except for possibly reducing the size of some reachable spines by removing repeated occurrences of the same decision. Finally, to show that the q-retrieval of a characterizing identity is also invariant to idempotence ([D.1]), consider the q-retrieval of the term shown in Figure 5.3(a) along the path $P_0$. Regardless of how $q_0$ is answered, the retrieval on $q$ is the same, as all the reachable paths below $q_0$ reach the variable $T$ on the same argument of $q$. Now, since the q-retrieval is invariant under reachable transposition, the identity [DR.4] can be used to produce the tree shown in Figure 5.3(b). Finally, applying idempotence at the leaves of the term yields the tree of Figure 5.3(c), where it is obvious that the retrieval along the path $P_0$ has not been changed.

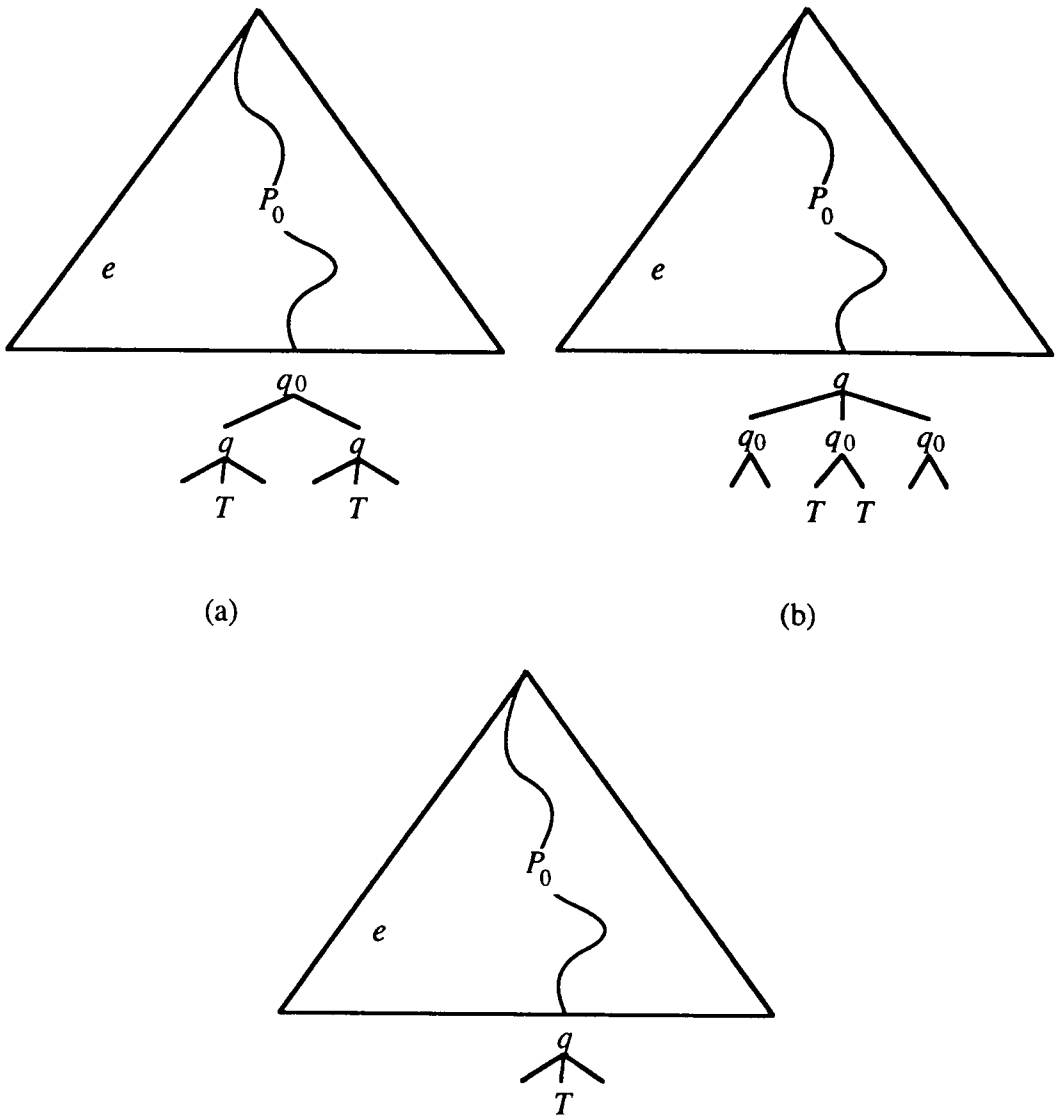



**Figure 5.3** Invariance of q-retrieval under idempotence: (a) Original tree; (b) Term after reachable transposition; (c) Resulting term after applying idempotence.

The following results show some of the properties of the q-retrieval over the algebra of characterizing terms.

**Lemma 5.19**

Given a decision theory $D = (Q, E)$, and a decision $q$ with arity $n$ and codomain $\{a_1, ..., a_n\}$, if $e_1$ and $e_2$ are characterizing terms in $q$ and $B$ is a conjunction of branch instructions, then

(i) $ret_q(B, T) = \{a_1, ..., a_n\}$

(ii) $ret_q(B, F) = \{\}$

(iii) $ret_q(B, e_1 \vee e_2) = ret_q(B, e_1) \cup ret_q(B, e_2)$

**Proof.**

Both (i) and (ii) follow from the definition of the q-retrieval.

To show (iii), observe that $ret_q(B, e_1 \vee e_2) = ret_q(B, e_2 \vee e_1)$. Also, $ret_q(B, e_1 \vee e_2) \supseteq ret_q(B, e_1)$ as the reachable spine set of $e_1$ is always contained in the reachable spine set of $e_1 \vee e_2$. Similarly, observe that $ret_q(B, e_2 \vee e_1) \supseteq ret_q(B, e_2)$. Therefore,

$$\begin{aligned} ret_q(B, e_1 \vee e_2) &\supseteq ret_q(B, e_1) \\ &\supseteq ret_q(B, e_2) \end{aligned}$$

which means that

$$ret_q(B, e_1 \vee e_2) \supseteq ret_q(B, e_1) \cup ret_q(B, e_2)$$

But $ret_q(B, e_1 \vee e_2)$ cannot be bigger that the union as $Sp_R(B, e_1 \vee e_2)$ consists of paths which have as subpaths either a path of $Sp_R(B, e_1)$ or a path of $Sp_R(B, e_2)$. Suppose that there exists an element $a_i$ of $ret_q(B, e_1 \vee e_2)$ which is not in the union of the q-retrievals of $e_1$ and $e_2$.

Then, $q \diamond a_i$ cannot occur on a reachable path of $e_1$, so it must occur on a reachable path of $e_2$. But this results in a contradiction.

Therefore, $ret_q(B, e_1 \vee e_2) = ret_q(B, e_1) \cup ret_q(B, e_2)$. ■

Hence, given the Boolean algebra of characterizing identities, $\langle \mathbf{B}; \neg, \vee, \wedge \rangle$, the q-retrieval preserves the top ($T$), bottom ($F$) and meet ($\vee$). However, as shown next, neither the join($\wedge$) nor the negation ($\neg$) is preserved by the q-retrieval property.

**Lemma 5.20**

There exists a decision theory $D = (Q, E)$, and a decision $q$ with arity $n$ and codomain $\{a_1, ..., a_n\}$, such that if $e_1$ and $e_2$ are characterizing terms in $q$ and $B$ is a conjunction of branch instructions, then:

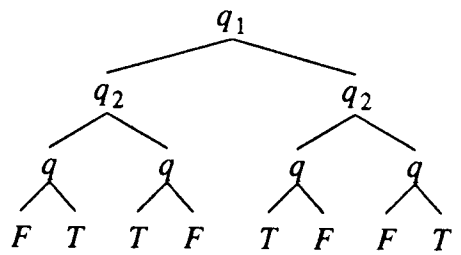
(i) $ret_q(B, e_1 \wedge e_2) \neq ret_q(B, e_1) \cap ret_q(B, e_2)$

(ii) $e_1 \neq e_2 \nRightarrow ret_q(B, \neg e_1) \neq ret_q(B, \neg e_2)$

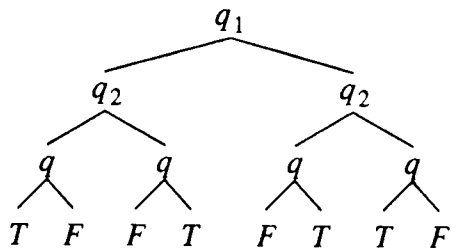
**Proof.**

(i) Consider the following two trees:

$e_1 =$




$e_2 =$




Obviously,

$$ret_q(q_1 \diamond 1, e_1) = ret_q(q_1 \diamond 1, e_2)$$
$$= \{1, 2\}.$$
$$= ret_q(q_1 \diamond 1, e_1) \cap ret_q(q_1 \diamond 1, e_2)$$

But note that $e_2 = \neg e_1$, thus,

$$ret_q(q_1 \diamond 1, e_1 \wedge e_2) = ret_q(q_1 \diamond 1, e_1 \wedge \neg e_1)$$
$$= ret_q(q_1 \diamond 1, F)$$
$$= \varnothing.$$

(ii) This result is actually a consequence of (i) above.

Consider again the previous example. Notice that although $e_1 \neq e_2$,

$$ret_q(q_1 \diamond 1, \neg e_1) = ret_q(q_1 \diamond 1, e_2)$$
$$= ret_q(q_1 \diamond 1, e_1)$$
$$= ret_q(q_1 \diamond 1, \neg e_2)$$

and thus the result. ■

**Lemma 5.21**

If $e_1$ and $e_2$ are functional identities then:

(i) $e_1 \wedge e_2$ is a functional identity.

(ii) $e_1 \vee e_2$ is *not* always a functional identity.

**Proof.**

(i) It is clear that if $e_1$ and $e_2$ are functional on some decision $q$, then $e_1 \wedge e_2$ is also functional on $q$ as, for each occurrence of $q$, repeat reduction guarantees that only the argument leading to the variable $T$ in $e_1$ can possibly lead to a reachable branch when $e_2$ is substituted. The decision $q$ is therefore definite at each occurrence in $e_1 \wedge e_2$.

Furthermore, $q$ is reachably semi-essential in $e_1 \wedge e_2$ as it was reachably semi-essential in $e_1$. Thus, $e_1 \wedge e_2$ is functional on $q$.

(ii) The fact that both $e_1$ and $e_2$ are functional identities on some decision $q$ does not mean that $e_1 \vee e_2$ is also functional on $q$. Consider the case where $e_1$ and $e_2$ yield different q-retrievals to the same conjunct of branch instructions. Since the q-retrieval on $e_1 \vee e_2$ is now the union of the individual q-retrievals of $e_1$ and $e_2$, then $q$ is not definite in $e_1 \vee e_2$. Thus, $e_1 \vee e_2$ is not a functional identity. ■

## 5.4 Retrieval Complete Knowledge Clusters

The following discussion introduces the notion of retrieval completeness with the aim of providing a method for deciding when a decision cluster can be used as an independent module of procedural and logical knowledge about some process.

**Definition 5.22**

Given a decision theory, $D = (Q, E)$, let $C$ be a decision cluster on $q \in Q$, and $R$ be a set of conjunctions of branch instructions in $Q - \{q\}$. Then, $C$ is said to be *retrieval complete* in $R$, if, given any valid identity, $e$, functional on $q$ and any conjunction of branch instructions, $B \in R$, then

$$ret_q(B, C) \subseteq ret_q(B, e).$$

**Lemma 5.23**

If the functional identities $e_1$ and $e_2$ are retrieval complete in $R$, then both $e_1 \vee e_2$ and $e_1 \wedge e_2$ are retrieval complete in $R$.

**Proof.**

Let $B \in R$ be a conjunct of branch instructions.

By Lemma 5.19, $ret_q(B, e_1 \vee e_2) = ret_q(B, e_1) \cup ret_q(B, e_2)$. Now, since both $e_1$ and $e_2$ are retrieval complete, then it follows that $ret_q(B, e_1) = ret_q(B, e_2)$. And therefore,

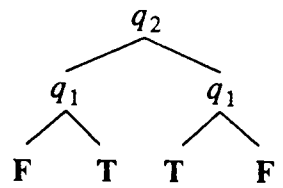
$$ret_q(B, e_1 \vee e_2) = ret_q(B, e_1) = ret_q(B, e_2)$$

Thus $e_1 \vee e_2$ is also retrieval complete.

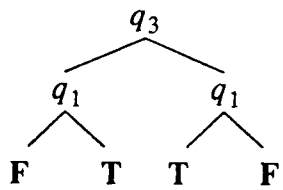
To show the second result, note that the set of reachable paths in $e_1 \wedge e_2$ is a subset of the set of reachable paths in $e_1$, and therefore, it follows that $ret_q(B, e_1 \wedge e_2) \subseteq ret_q(B, e_1)$. Similarly, since $e_1 \wedge e_2 = e_2 \wedge e_1$, then $ret_q(B, e_1 \wedge e_2) \subseteq ret_q(B, e_2)$. Finally, since both $e_1$ and $e_2$ are retrieval complete, then $e_1 \wedge e_2$ is also retrieval complete. ■

The retrieval problem presented in Example 5.17 above may be solved by constructing the corresponding retrieval complete decision cluster. First, the corresponding characterizing terms for both $q_1\langle x\ 'y\rangle = q_2\langle y\ 'x\rangle$ and $q_1\langle x\ 'y\rangle = q_3\langle y\ 'x\rangle$, is obtained.

$e_1 =$

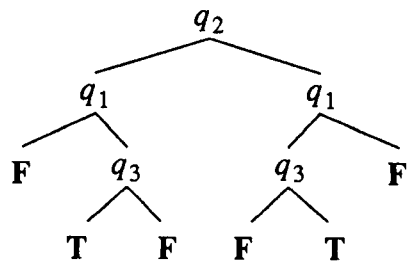



$e_2 =$

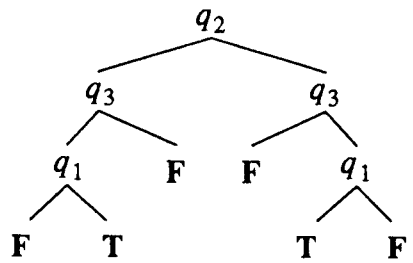



Since both $e_1$ and $e_2$ are functional, the characterizing term $e_1 \wedge e_2$, is also functional.

$e_1 \wedge e_2 =$




Decision delaying $q_1$ yields the term




which is a decision cluster on $q_1$ in $\{q_2, q_3\}$, since $q_1$ is both reachably semi-essential and definite. Expressing the decision cluster in algebraic form yields:

$$q_1 :\rangle\ q_2\langle 2 \Rrightarrow q_3\langle 2\ '\ _\rangle\ '\ 1 \Rrightarrow q_3\langle _\ '\ 1\rangle\rangle.$$

Note that the resulting decision cluster is the characteristic identity for this decision theory, which, in fact, is retrieval complete. The q-retrieval of $[q_3 \diamond 1]$ results in $\{2\}$. Similarly, notice that retrieving $[q_2 \diamond 1, q_3 \diamond 2]$ on the original functional identities resulted in $\{2\}$ for $e_1$ and in $\{1\}$ for $e_2$. The q-retrieval of the same conjunct on the retrieval complete decision cluster is empty as the corresponding branch is unreachable.

The procedure above describes a method for generating a retrieval complete decision cluster from a group of functional identities. In practice, however, one is given a whole set of identities and it is important to determine the decision expression of minimal size such that, given a subset of decisions in the theory, the expression is minimally retrieval

complete with respect to that subset. Such a cluster is said to be minimally retrieval complete.

**Theorem 5.24**

Let $D = (Q, E)$ be a decision theory.

There exists a valid identity $e'$, functional on $q \in Q$, such that if $R$ is a set of conjunctions of branch instructions in $Q - \{q\}$, then

(i) Given any valid identity, $e$, functional on $q$ and a conjunction of branch instructions, $B \in R$, then

$$ret_q(B, e') \subseteq ret_q(B, e).$$

(ii) If $e$ satisfies (i), i.e. $e$ is retrieval complete, then $e \Rightarrow e'$.

**Proof.**

The proof for this theorem is given in two steps. First, it is shown that there exists at least one identity $e'$ which satisfies (i) above, e.g. the characteristic identity. Then, it is shown that among those identities that satisfy (i), there exists one which is minimal in the implication preorder, namely $e'$, which is unique up to decision equivalence.

(i) Let $e_1, ..., e_m$ be the set of valid identities functional on $q$. By Lemma 5.21, the identity $e_0 = e_1 \wedge ... \wedge e_m$ is also valid and functional on $q$. Moreover, since the set of reachable paths in $e_0$ is always contained in the set of reachable paths of any identity $e \in \{e_1, ..., e_m\}$, then $e_0$ is also retrieval complete and, thus, it satisfies (i) above.

(ii) Let $e_1, ..., e_k$ be the set of valid identities functional on $q$ satisfying (i) above. Since $e_1, ..., e_k$ are all retrieval complete then, by

Lemma 5.23, $e_1 \vee ... \vee e_k$ is also a retrieval complete decision cluster (and hence functional on $q$). Now, given any $e \in \{e_1, ..., e_k\}$, it is obvious that $e \vee (e_1 \vee ... \vee e_k) = e_1 \vee ... \vee e_k$, i.e.,$e \Rightarrow e_1 \vee ... \vee e_k$. Therefore, $e' = e_1 \vee ... \vee e_k$. ■

An identity $e'$, with the properties of the theorem above, is said to be *minimally retrieval complete*. Similarly, the corresponding decision cluster is said to be a *minimally retrieval complete decision cluster*. If $E'$ be the set of identities in $E$ functional on some decision $q$, then the set of identities in $E'$ forms a lattice that looks like that of Figure 5.4

The previous result shows that there exists at least one retrieval complete decision cluster in every decision theory. The result also presents a method for finding a retrieval complete decision cluster given a set of functional identities. A more practical technique for finding minimally retrieval complete decision clusters is described next.

Suppose now, that the set R contains only conjunctions of branch instructions on decisions from the set $\{q_1, ..., q_n\} \subseteq Q$. The approach described above can be used to find a minimally retrieval complete decision cluster on a set R of this form. The first step is to obtain the characteristic identity, $e_0$. The second step is to decision delay every decision $q_k$ in $e_0$ that is not in $\{q_1, ..., q_n\}$. Then, $q_0$ is migrated until the reachable paths in resulting antecedent subtree involve only decisions in $\{q_1, ..., q_n\}$ as shown in Figure 5.5. The decision cluster obtained from the antecedent subtree is minimally retrieval complete in $\{q_1, ..., q_n\}$. The argument for this goes as follows: since $e_0$ is the characteristic identity, it implies every valid functional identity in the theory. Assuming that there exists one such functional identity $e'$ which is minimally retrieval complete, then it is obvious that $e_0 \vee e' = e'$ and thus, $e_0 \Rightarrow e'$. In fact, if the retrieval of a conjunct $B$ on the characteristic identity is empty,then it follows that it must be also empty on every retrieval complete decision cluster.

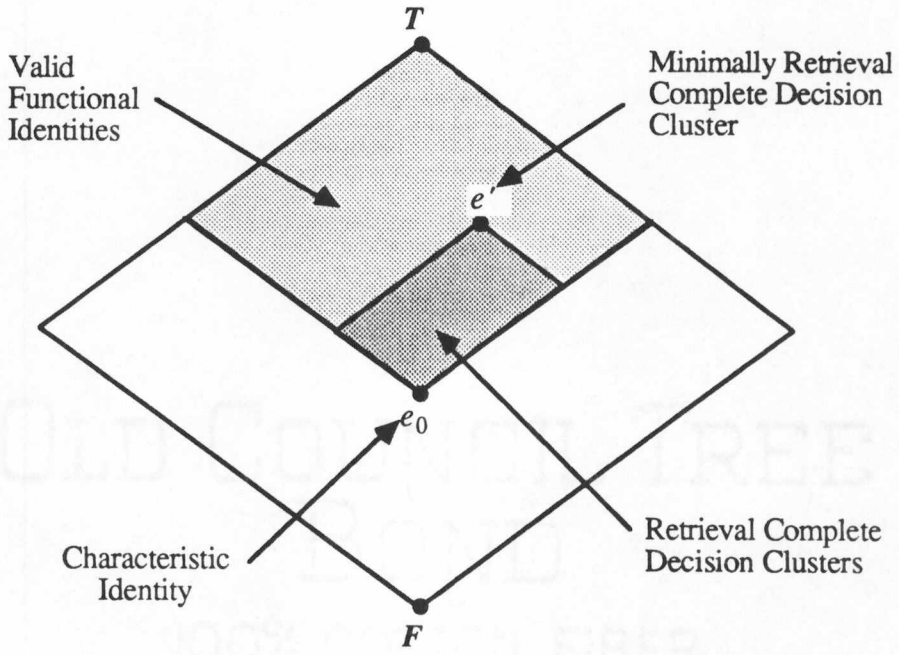



**Figure 5.4** The lattice of identities functional on some decision $q$.

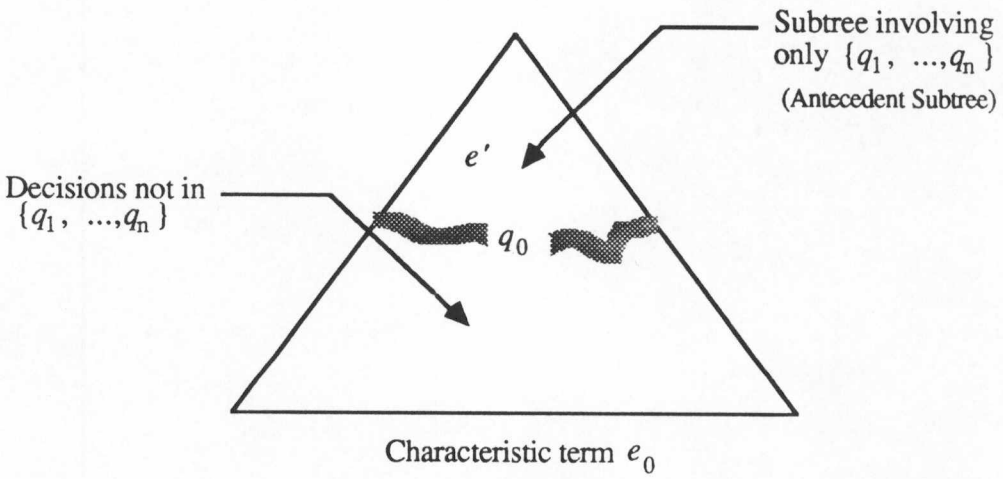



**Figure 5.5** Minimally retrieval complete term $e'$ obtained from the characteristic term $e_0$.

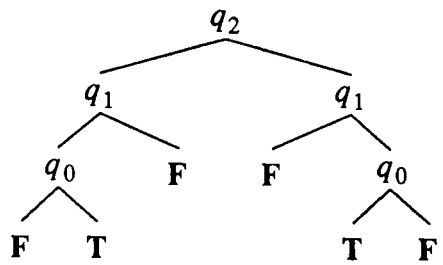
**Example 5.25**

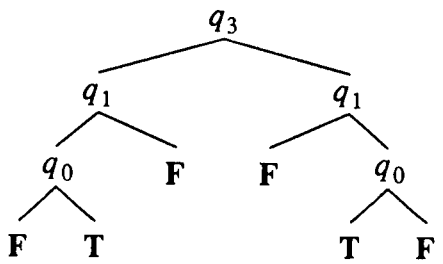
Given the decision theory $D = (\{q_0, q_1, q_2, q_3\}, E)$, where

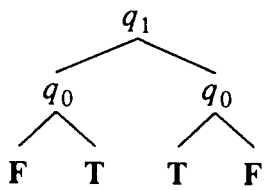
$$E = \{q_2\langle q_1\langle x_1\ 'x_3\rangle\ 'q_1\langle x_4\ 'x_2\rangle\rangle = q_0\langle x_2\ 'x_1\rangle,$$
$$q_3\langle q_1\langle x_1\ 'x_4\rangle\ 'q_1\langle x_3\ 'x_2\rangle\rangle = q_0\langle x_2\ 'x_1\rangle,$$
$$q_1\langle x_1\ 'x_2\rangle = q_0\langle x_2\ 'x_1\rangle\},$$

find a minimally complete decision cluster on $q_0$ in $\{q_1\}$.

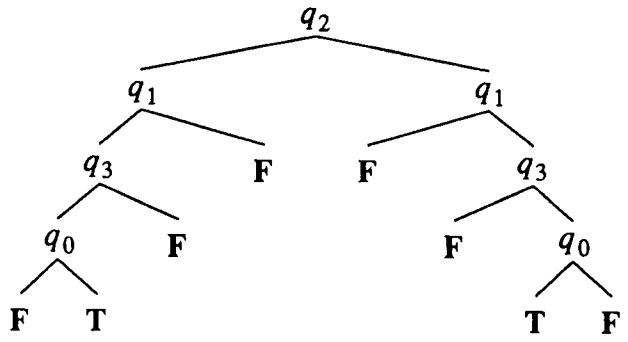
First, notice that all the identities in $E$ are functional on $q_0$. In fact, expressed as characterizing terms, it is easy to observe that they are all retrieval complete in $\{q_1\diamond 1, q_1\diamond 2\}$. The q-retrieval of $[q_1\diamond 1]$ results in $\{1\}$ and the q-retrieval of $[q_1\diamond 2]$ results in $\{2\}$ in all three cases.



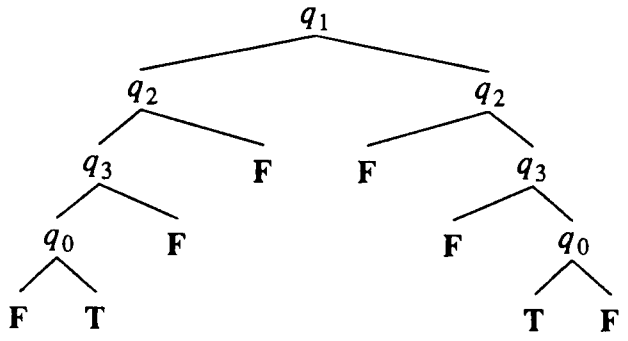



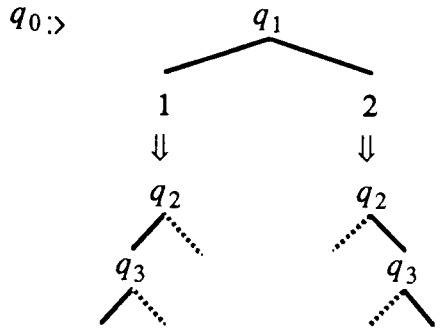
Now, combining all the identities into a characterizing identity results in:




Decision delaying $q_2$ past $q_1$ produces the following term:




from which the following decision cluster is obtained by migrating the conclusions to $q_0$.




The antecedent subtree is a minimally complete decision cluster on $q_0$ in $\{q_1 \diamond 1, q_1 \diamond 2\}$. ■

# CHAPTER 6

# EXPERIMENTAL RESULTS

This chapter discusses a particular implementation of the techniques for manipulating decision expressions which have been presented here. The goal of this implementation effort was to show that although the optimization of decision trees is NP-hard, the algorithms developed in this investigation can provide optimal solutions to useful and realistic problems in procession times which are very reasonable.

The chapter is organized as follows. The first section gives an overview of the software implementation. The second section discusses the transition from rules to decision trees in detail as this helps in the presentation of the examples shown. The third section presents the results of using the tools implemented by the author to several applications at the same time that it provides an insight to how the algorithms can be applied to realistic problems.

## 6.1 Software Implementation

The techniques for manipulating decision expressions developed in the course of this research effort have been implemented by the author into a series of software tools which, collectively, are referred to as Decision Tree Designer (DTD). The DTD was implemented in Prolog using Quintus Prolog Development Environment. The software developed was tested on a Digital Equipment Corporation VAXStation. It should also be mentioned here that the same code was run on a VAX 8650 computer and a Sun Microsystems SUN 3/160

workstation. All the cpu times reported here, unless otherwise stated, are related to the VAXStation implementation.

The DTD is shown in block diagram form in Figure 6.1. Starting from either a production rule or decision tree based representation of a problem, the DTD can automatically generate a decision tree solution which is computationally optimal with respect to some reasonable cost criterion. The user can alternatively elect to explore, interactively, all of the possible optimal solutions to the problem and choose the tree or set of trees that are best suited for the particular application. Finally, from that optimal representation of the solution, the system (by means of an optional software module not shown in Figure 6.1) can automatically generate a high-level language (e.g. C) source code implementation which can be ported to a target computer. The design cycle is completed by being able to transform an optimal decision tree into and equivalent production rule representation.

As shown in Figure 6.1, the Decision Tree Designer is made up of three programs:

a) A program to transform a set of rules into a decision tree.

The input to this module is a set of production rules expressed in implication form. These are represented internally as Prolog terms of the form:

rule(LABEL) :: $q_1$<?>[$a_1$] & ... & $q_n$<?>[$a_n$] => q<?>[c].

which can be interpreted as follows:

IF decision "$q_1$" is answered "$a_1$",

...

AND decision "$q_n$" is answered "$a_n$",

THEN the action of "q" must be "c".

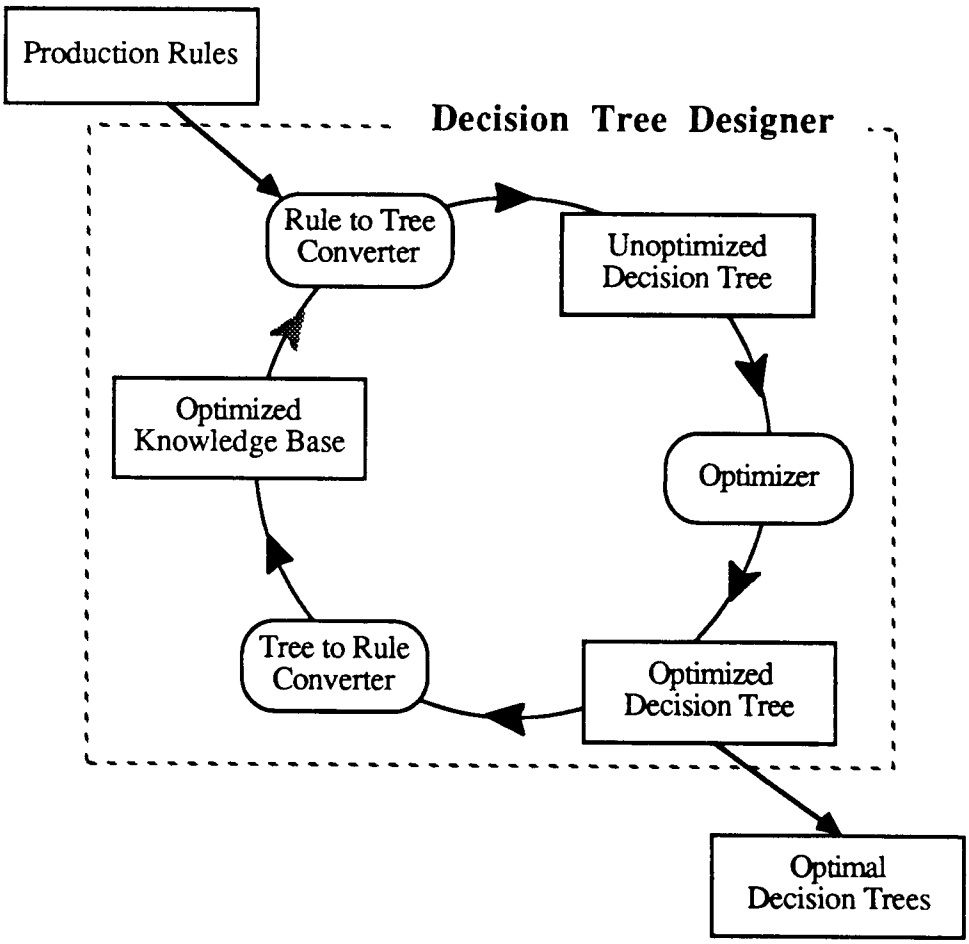



**Figure 6.1** Block diagram for the Decision Tree Designer (DTD).

The output of this module is a decision tree represented by a Prolog structure of the form:

tree(LABEL):: q :>

$q_1$: [$a_{11}$ >> $q_2$: [$a_{21}$ >> <: $c_1$, $a_{22}$ >> <: $c_2$],

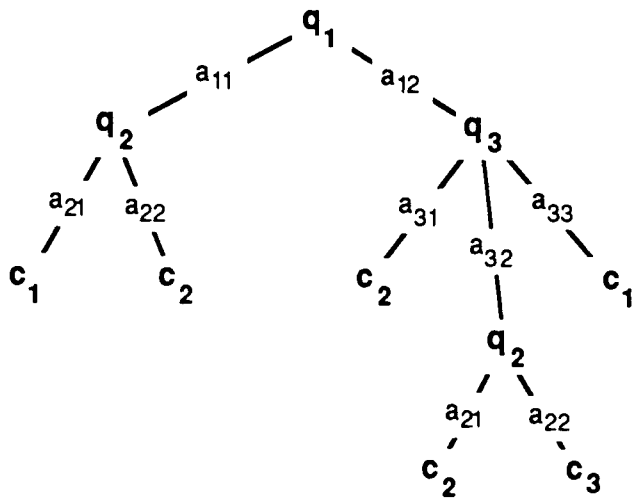
$a_{12}$ >> $q_3$: [$a_{31}$ >> <: $c_2$,

$a_{32}$ >> $q_2$: [$a_{21}$ >> <:$c_2$,$a_{22}$ >> <: $c_3$],

$a_{33}$ >> <: $c_1$]

].

which can be displayed in a more tree-like form as:




b) A program to optimize a decision tree / find all the irreducible forms. The optimization algorithm used in this program utilizes the concepts presented in Chapters 4 and 5 to extend the techniques discussed in Chapters 2 and 3 to handle decision trees with identities.

The algorithm is coded in a way that minimizes the storage required to generate all optimal trees. Basically, the program uses the

separations planning strategy to remember the exact shape of all irreducibles that have been generated so far. Thus, the algorithm keeps in memory only that tree that was generated last, replacing it with a new tree the next time an irreducible is generated.

c) A program to transform a decision tree into an equivalent set of production rules.

In the tree-to-rules converter software, each path in the tree is considered as an individual rule. Then, each rule is reduced to a *prime spine* and added to the set of production rules if it is not already present. A prime spine in a decision tree is defined as a conjunction of terms that has a unique retrieval on the tree. In other words, all the paths that "agree" with the conjunction of terms in the spine lead to the same conclusion. Furthermore, a prime spine is not contained by any of its equivalent spines.

Although the output of this stage is a set of production rules that is decision equivalent to the input tree, it may not correspond exactly to the original set of rules from which the tree was generated. However, both forms of representation are guaranteed to be semantically equivalent.

This chapter presents the result of using the DTD tool on several sample problems. Each problem is introduced as an example to illustrate some of the potential uses for the techniques developed in this research. For each example, the corresponding cpu "run time" is reported to demonstrate that, although the problem of optimizing decision trees is NP-hard, the algorithms of this paper can be implemented on computers with conventional architectures to solve very useful applications with reasonable computation speeds.

## 6.2 Generation of Characterizing Terms from Rules

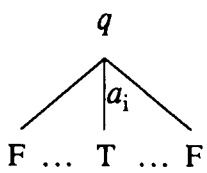
However, before introducing the examples, a technique developed by the author to generate characterizing terms from production rules is presented in this section. Production rules are logical statements of the form:

IF condition-1 AND ... AND condition-$n$ THEN conclusion. (1)

The rule (1) can be represented by the following algebraic formula:

$$q_1 \diamond a_1 \wedge q_2 \diamond a_2 \wedge \ldots \wedge q_n \diamond a_n \vdash q_0 \diamond a_0 \tag{2}$$

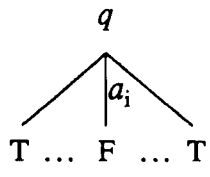
where the pair $q_i \diamond a_i$ represents the truth value of the ith condition in the logical statement (1). Assuming that the possible outcomes of a decision are mutually exclusive, the statement $q \diamond a_i$ is read as follows: $q \diamond a_i$ is true if the outcome of the decision $q$ is the codomain value $a_i$. In terms of decision expressions, the conditional $q \diamond a_i$ is equivalent to the following term:




(3)

Notice that the outcome of the decision expression (3) is true if the decision $q$ results in $a_i$ and false otherwise.

Similarly, $\neg (q \diamond a_i)$ is true only if the outcome of the decision $q$ assumes any codomain value except for $a_i$. Thus, the term in a decision theory which corresponds to the conditional $\neg (q \diamond a_i)$ is given by

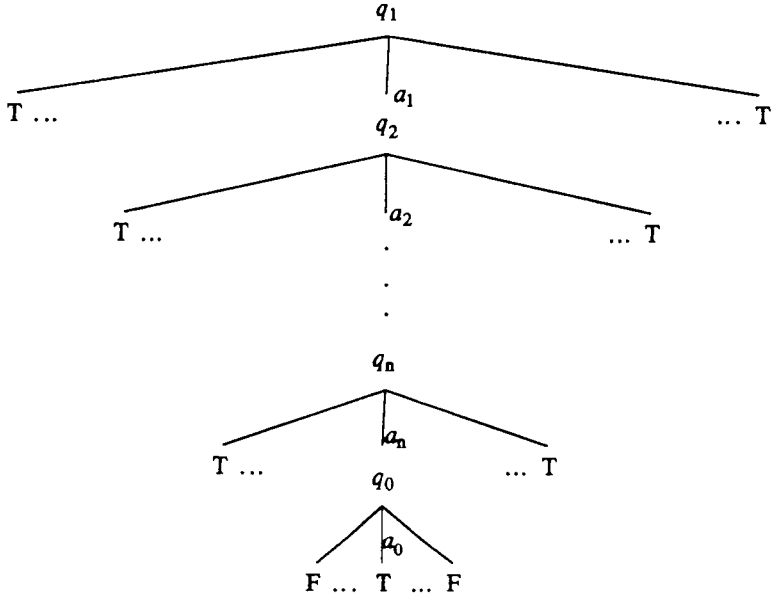



(4)

The removal of the implication in formula (2) above yields the expression given by the following disjunction of conditionals:

$$\neg (q_1 \diamond a_1) \vee \neg (q_2 \diamond a_2) \vee \ldots \vee \neg (q_n \diamond a_n) \vee q_0 \diamond a_0. \tag{5}$$

Once in that form, it is obvious that a characterizing term equivalent to the rule above can be obtained by computing the join of the individual conditionals expressed as decision trees. Thus the corresponding characterizing term which results has the form:




Finally, in order to obtain the logical equivalent to a set of rules in terms of a decision expression one can form, for each one of the rules, the corresponding characterizing term. Computing the meet of these terms (which corresponds to a characteristic term) yields the desired result.

## 6.3 Examples

The first example presented in this section discusses a very simple decision-making problem and how, from a rule based specification of the problem, one can arrive at a very efficient decision tree representation of the solution. The second example presents an application involving a simplified character recognition system and discusses how the algorithm to find all the irreducible forms of a tree reduces the problem of finding an optimal solution by several orders of magnitude. The third example illustrates how the extension of these algorithms to handle decision clusters can be used to generate all the possible irreducible trees which are consistent with a set of examples describing a particular decision process. Also, a brief discussion on how these techniques can be extended to develop a system for interactive knowledge acquisition is included. The fourth example involves inferring an optimal adaptive experiment (a sequential decision procedure for locating faults in an electronic circuit) from a given fault table involving six binary tests. And finally, the fifth example illustrates a problem which was solved using the techniques developed here, but for which, the implementation of alternative search methods is simply not possible.

**Example 6.1**

Consider the following problem (due to Cockett [14]). George has three daughters: Sue, Pam and Joy.

- Sue and Joy have brown hair.
- Pam's hair is red.
- Sue always wears a skirt, while Joy never does.
- Pam may or may not wear a skirt.

One day, the owner of a candy store found out that one of George's daughters had stolen some candy from his store. The owner of the store knew George, the girl's father, but not the girl's name. So, he went to George and ask for his money. George paid the owner, and asked him the girl's name, but the owner did not know.

How can George find out which of his daughters should be punished?

The logical specification of the problem can be described by the following set of *prime* rules.

**Rule 1:** If the girl wears a skirt and has brown hair then her name is Sue.

**Rule 2:** If the girl does not wear a skirt and has brown hair then her name is Joy.

**Rule 3:** If the girl has red hair then her name is Pam.

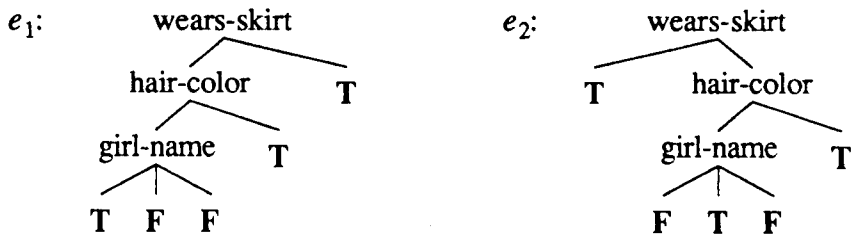
An algebraic representation for these rules yields:

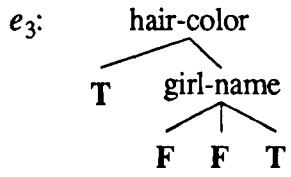
**Rule 1:** wears-skirt ◇ yes ∧ hair-color ◇ brown ⊢ girl-name ◇ Sue.

**Rule 2:** wears-skirt ◇ no ∧ hair-color ◇ brown ⊢ girl-name ◇ Joy.

**Rule 3:** hair-color ◇ red ⊢ girl-name ◇ Pam.

Obviously, the goal of this decision process is to determine the girl's name. For each one of these rules, one can generate a characterizing term. The resulting characterizing terms $e_1$, $e_2$ and $e_3$, corresponding respectively to rules 1, 2 and 3 above, can be expressed algebraically as follows:



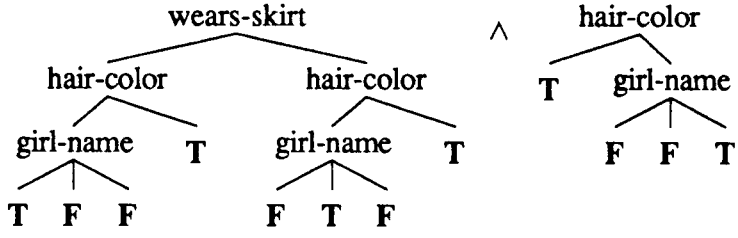



where the decision girl-name has codomain {*Sue*, *Joy*, *Pam*}, wears-skirt has codomain {*yes*, *no*} and hair-color has codomain {*brown*, *red*}. The branches below the decisions in the terms above are labeled from left to right, in the same order as their corresponding codomain lists.

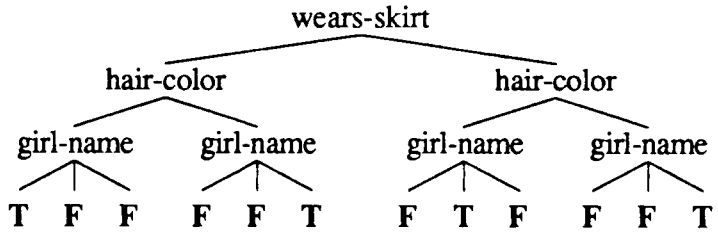
Once in characterizing term form, we can generate a decision cluster whose retrievals contain the retrievals of all three identities as follows.

First, we obtain the characteristic term $e_1 \wedge e_2 \wedge e_3$.

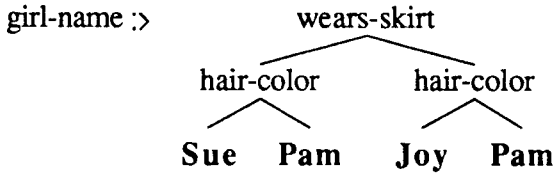
$e_1 \wedge e_2 \wedge e_3 =$



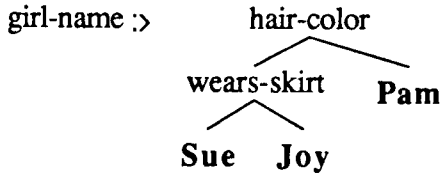

$e_1 \wedge e_2 \wedge e_3 =$




from which we obtain the corresponding decision cluster:



The resulting decision cluster is simply-reduced. However, it is not an irreducible. Transposing wears-skirt and hair-color and idempotent reducing, we obtain:




which is the only irreducible form for the cluster.

Finding the irreducible form for this problem required 560 msec. of cpu time. ■

**Example 6.2**

Suppose we want to design a system capable of recognizing a subset of the alphanumeric characters based on four basic features: the number of holes in the character, the position of the hole (if any), the number of bays in the character and the direction of the bay (if any). The alphanumeric characters in question are: 1, 6, 9, 0, A, C, P and U. Figure 6.2 shows how these features are used to describe the shape of objects in binary images.

Let the four features be represented by the following decisions: *number of holes* (#holes) with codomain {*zero*, *one*}, *number of bays* (#bays) with codomain {*zero*, *one*}, *position of hole* (hole-pos) with codomain {*top*, *bottom*} and *direction of bay* (bay-dir) with codomain {*north*, *west*, *south*, *east*}. A set of rules describing the characters in question is shown in Figure 6.3.

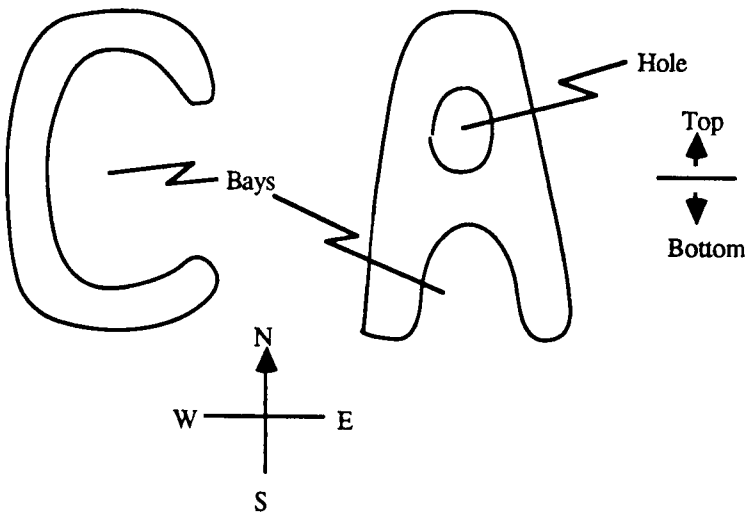
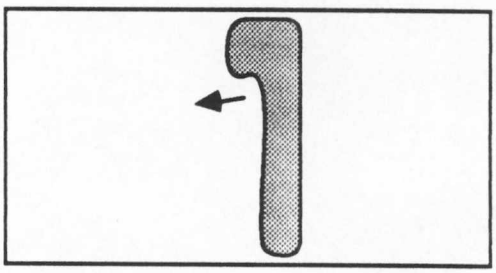
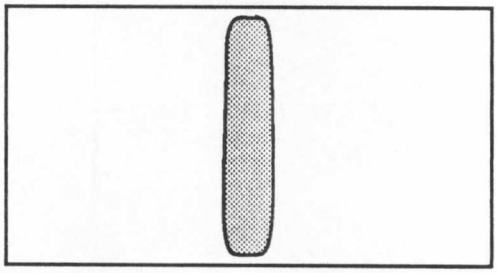



**Figure 6.2** Features used to describe the characters.

Rule: ONE_1.

IF number of holes is zero
AND number of bays is one
AND direction of bay is west
THEN character is 1.



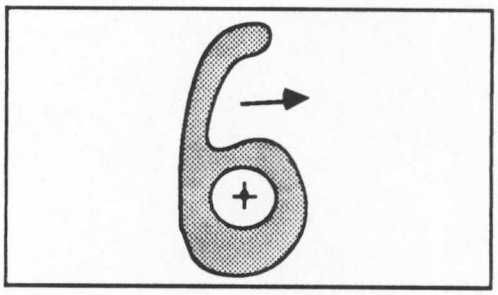


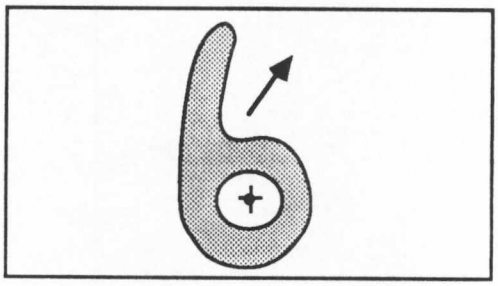
Rule: ONE_2.

IF number of holes is zero
AND number of bays is zero
THEN character is 1.

Rule: SIX_1.

IF number of holes is one
AND position of hole is bottom
AND number of bays is one
AND direction of bay is east
THEN character is 6.





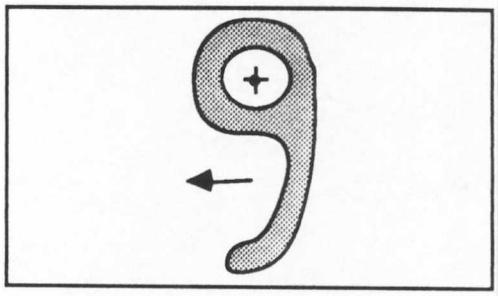
Rule: SIX_2.

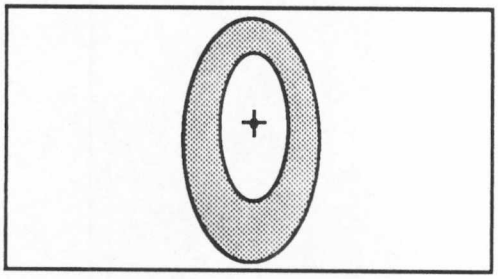
IF number of holes is one
AND position of hole is bottom
AND number of bays is one
AND direction of bay is north
THEN character is 6.

**Figure 6.3** Rules for recognizing the characters 1, 6, 9, 0, A, C, P, U.

Rule: NINE.

IF number of holes is one
AND position of hole is top
AND number of bays is one
AND direction of bay is west
THEN character is 9.



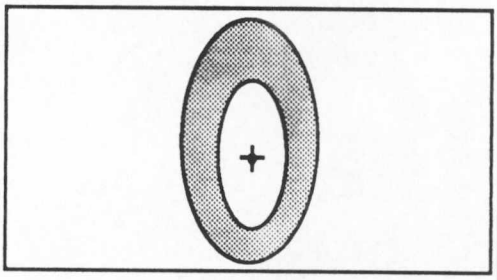


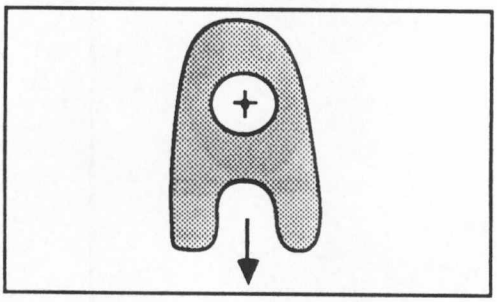
Rule: ZERO_1.

IF number of holes is one
AND position of hole is top
AND number of bays is zero
THEN character is 0.

Rule: ZERO_2.

IF number of holes is one
AND position of hole is bottom
AND number of bays is zero
THEN character is 0.





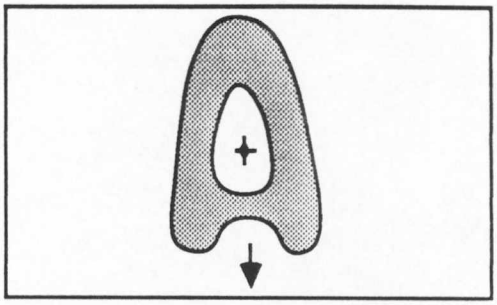
Rule: A_1.

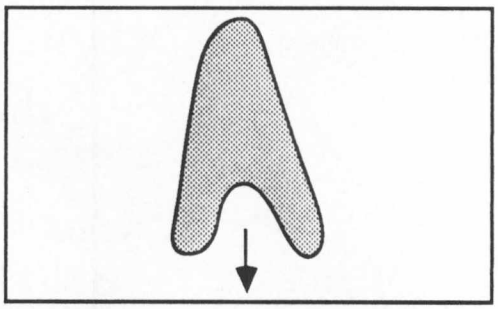
IF number of holes is one
AND position of hole is top
AND number of bays is one
AND direction of bay is south
THEN character is A.

**Figure 6.3** (Cont.)

Rule: A_2.

IF number of holes is one
AND position of hole is bottom
AND number of bays is one
AND direction of bay is south
THEN character is A.



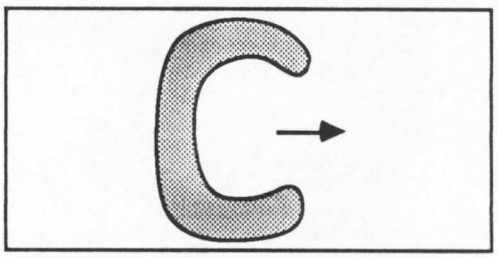


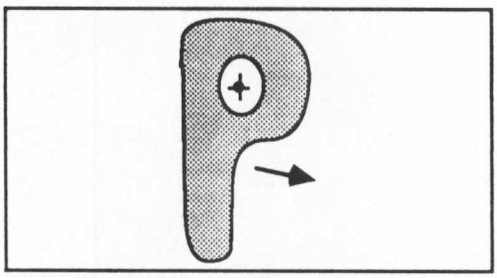
Rule: A_3.

IF number of holes is zero
AND number of bays is one
AND direction of bay is south
THEN character is A.

Rule: C.

IF number of holes is zero
AND number of bays is one
AND direction of bay is east
THEN character is C.





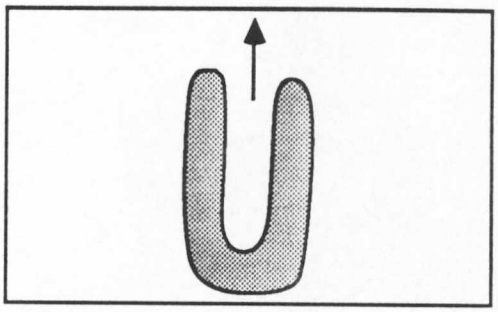
Rule: P.

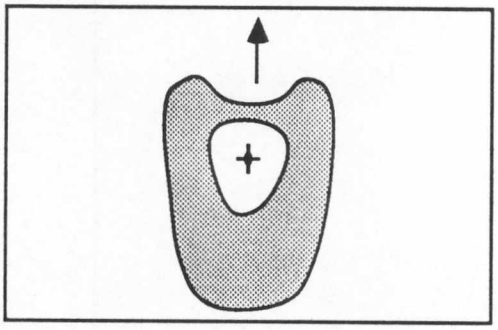
IF number of holes is one
AND position of hole is top
AND number of bays is one
AND direction of bay is east
THEN character is P.

**Figure 6.3** (Cont.)

Rule: U.

IF number of holes is zero
AND number of bays is one
AND direction of bay is north
THEN character is U.





Rule: UNREADABLE_1.

IF number of holes is one
AND position of hole is top
AND number of bays is one
AND direction of bay is north
THEN character is unreadable.

Rule: UNREADABLE_2.

IF number of holes is one
AND position of hole is bottom
AND number of bays is one
AND direction of bay is west
THEN character is unreadable.

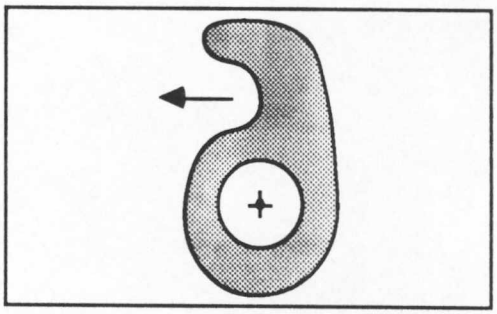


**Figure 6.3** (Cont.)

From that set of rules, a procedure similar to the one used in the example above yields the decision cluster in Figure 6.4. The resulting decision tree represents a formulation of the knowledge in the rules of Figure 6.3 that can be used to classify the eight characters.

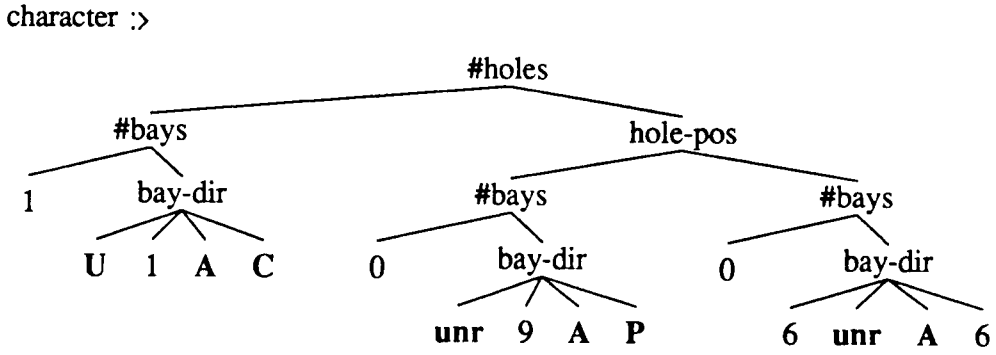



**Figure 6.4** Decision cluster for recognizing the characters 1, 6, 9, 0, A, C, P and U based on number of holes (#holes), hole position (hole-pos), number of bays (#bays) and bay direction (bay-dir).

In fact, the description of this problem can be represented by any one of 22,464 possible decision clusters, all of which are decision equivalent to the rules in Figure 6.3. However, only nine of these clusters are irreducible. Furthermore, of the nine irreducible decision trees corresponding to this example, only the three forms shown in Figure 6.5 are unconditionally stable. That is, the optimality of these forms are not constrained by any conditions on the relative costs of the decisions in the tree. The remaining six forms (all of which have the attribute *position of hole* at the root) are conditionally stable. That is, they are optimal only when the relative costs of the decisions in the tree are related by specific conditions.

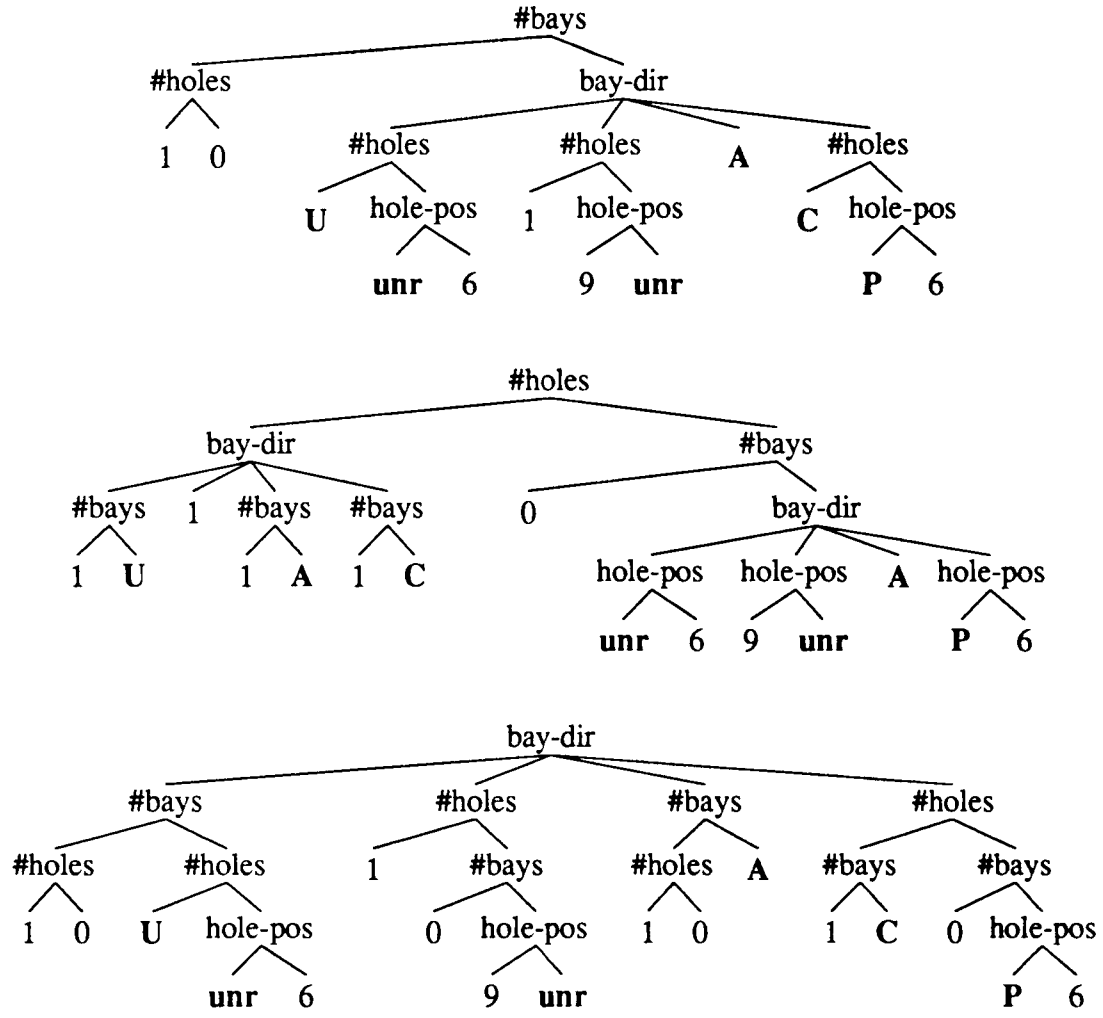



**Figure 6.5** Stable irreducible forms corresponding to the character recognition decision cluster.

All nine irreducible forms for the tree in this example were generated in less than 1.5 minutes of cpu time (an average of 10 seconds per irreducible). ■

**Example 6.3**

The data used for this example was adapted from an example by J. R. Quinlan [36]. The task is to classify a day as suitable to play golf or not given a set of attributes which describe the weather on that day. The assumption is that the examples are described in terms of attributes having discrete and mutually exclusive values. Thus, the weather can be described by the following attributes:

*outlook*, with values {sunny, overcast, rain}

*temperature*, with values {cold, warm}

*humidity*, with values {high, low}

*windy*, with values {true, false}

If we let the notation $x \diamond y$ represent the case when the attribute $x$ has value $y$, then the weather on a particular day can be described by the following conjunction of attributes:

$$\textit{outlook} \diamond \text{overcast} \wedge \textit{temperature} \diamond \text{warm} \wedge$$
$$\textit{humidity} \diamond \text{high} \wedge \textit{windy} \diamond \text{false}.$$

Table 6.1 shows a set of particular days in which the weather has been described in terms of the attributes *outlook*, *temperature*, *humidity* and *windy*. In addition, each day has been classified as either a *play* or a *don't play* day depending on whether that day was suitable to play golf or not.

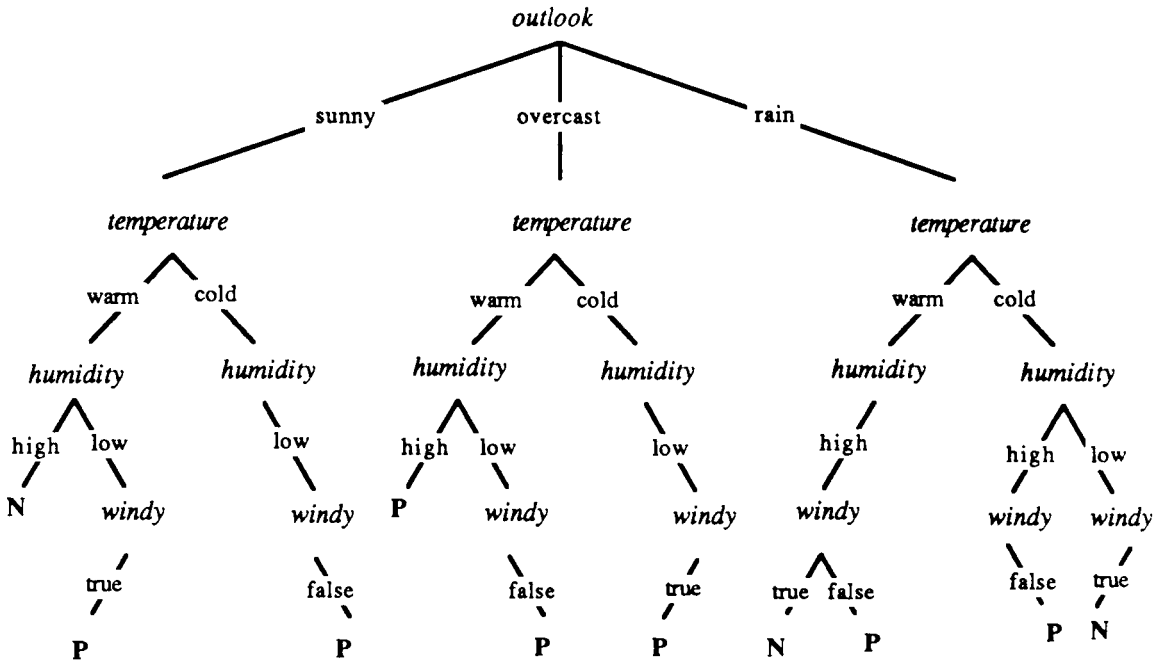
**Table 6.1** Example of days in which a decision to play golf or not has been made in terms of the weather.

| *outlook* | *temperature* | *humidity* | *windy* | *decision* |
|---|---|---|---|---|
| rain | warm | high | true | don't play |
| rain | cold | low | true | don't play |
| overcast | warm | high | true | play |
| overcast | warm | high | false | play |
| rain | warm | high | false | play |
| overcast | cold | low | true | play |
| sunny | warm | low | true | play |
| sunny | warm | high | true | don't play |
| sunny | warm | high | false | don't play |
| overcast | warm | low | false | play |
| rain | cold | high | false | play |
| rain | cold | high | false | play |
| sunny | warm | high | false | don't play |
| sunny | cold | low | false | play |

A decision tree which correctly classifies all the entries in Table 1 can be constructed by backward chaining on rules derived from the examples in the table. For instance, the first entry in the table corresponds to a rule given by the following logical statement:

$$\textit{outlook} \diamond \text{rain} \wedge \textit{temperature} \diamond \text{warm} \wedge \textit{humidity} \diamond \text{high} \wedge \textit{windy} \diamond \text{true} \vdash \textit{decision} \diamond \text{don't play}.$$

Using the correspondence between rules and characterizing terms, and following the approach outlined in the previous examples, we get the following decision tree, where missing branches indicate the absence of an example with that set of attributes and the labels **P** and **N** at the leaves stand for *play* and *don't play* respectively.




Once a decision tree has been generated from the training set, all the irreducible forms of the tree can be generated. In total, there are 63 irreducible decision trees which correctly classify all the given examples. However, only 48 of them are stable.

In the figures that follow, we have omitted the labels from the branches and assumed the following convention:

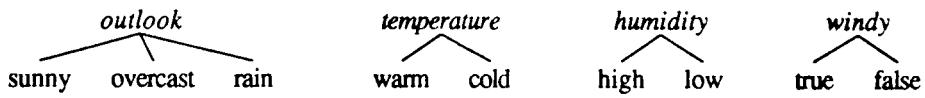


Figure 6.6 shows the irreducible forms which have the decision *outlook* at the root. There are 16 different irreducibles corresponding to the four different forms that can occur when the *outlook* is sunny, and the four different forms that can occur when the *outlook* is rain. The outcome when *outlook* is overcast is always the same: *play*.

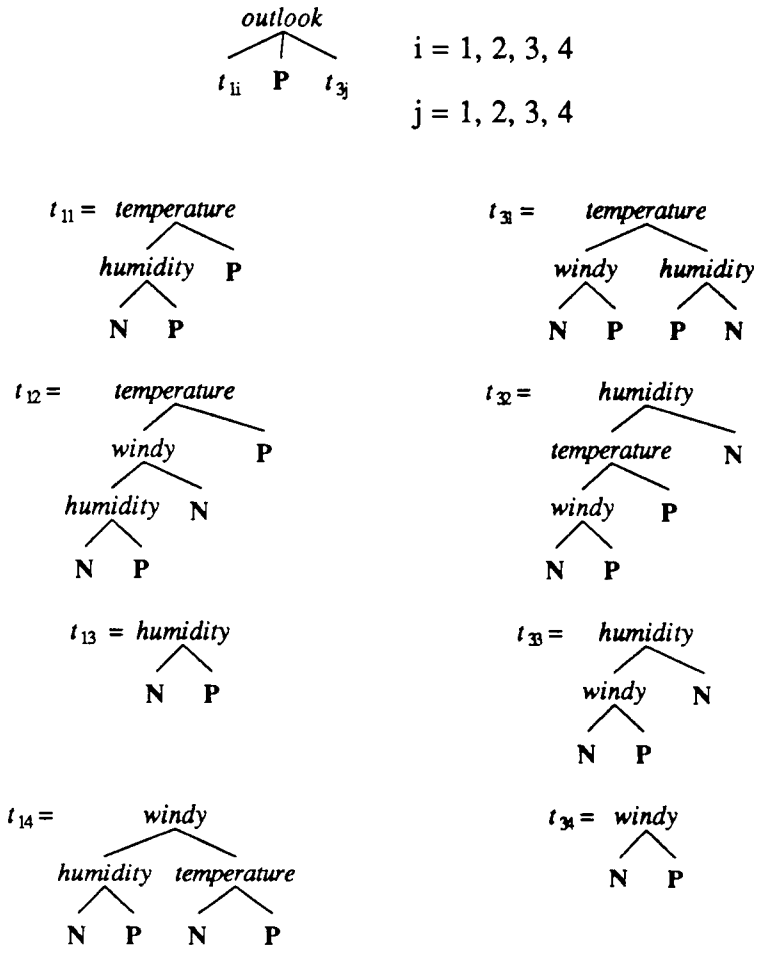



**Figure 6.6** Irreducible forms with *outlook* at the root.

Note, however, that the decision tree *outlook* $\langle t_{12}$ ' **P** ' $t_{31}\rangle$ is unstable since the subtree $t_{12}$ is optimal only when the cost of evaluating *windy* is less than the cost of evaluating *humidity* (otherwise the subtree $t_{13}$ is always better), while the subtree $t_{31}$ assumes that the opposite condition is true (otherwise $t_{34}$ is always better). Similar analysis reveals that *outlook* $\langle t_{12}$ ' **P** ' $t_{32}\rangle$, *outlook* $\langle t_{12}$ ' **P** ' $t_{33}\rangle$, *outlook* $\langle t_{14}$ ' **P** ' $t_{31}\rangle$, *outlook* $\langle t_{14}$ ' **P** ' $t_{32}\rangle$ and *outlook* $\langle t_{14}$ ' **P** ' $t_{33}\rangle$ are also unstable. Thus, there are only 10 irreducible forms with *outlook* at the root which are stable. Figure 6.7 shows the irreducible forms which have the decision *temperature* at the root. Even though there are five different forms for each of the two outcomes of *temperature* (warm and cold), only 20 of the 25 different combinations are irreducibles. The decision trees *temperature* $\langle t_{12}$ ' $t_{25}\rangle$, *temperature* $\langle t_{14}$ ' $t_{21}\rangle$, *temperature* $\langle t_{14}$ ' $t_{22}\rangle$, *temperature* $\langle t_{15}$ ' $t_{21}\rangle$ and *temperature* $\langle t_{15}$ ' $t_{22}\rangle$ are not irreducible. Of the irreducible forms, two are unstable: *temperature* $\langle t_{15}$ ' $t_{23}\rangle$ and *temperature* $\langle t_{15}$ ' $t_{24}\rangle$. Thus, there are only 18 stable irreducible forms with *temperature* at the root. Figure 6.8 shows the irreducible forms which have the decision *humidity* at the root. Even though there are four different subtrees that can follow when the *humidity* is high, and five when the *humidity* is low, only 15 of the 20 possible combinations are irreducible. Note that *humidity* $\langle t_{14}$ ' $t_{21}\rangle$ is transposition equivalent to *temperature* $\langle t_{11}$ ' $t_{24}\rangle$ of Figure 6.7. Similarly, *humidity* $\langle t_{14}$ ' $t_{22}\rangle$ is transposition equivalent to *temperature* $\langle t_{11}$ ' $t_{23}\rangle$ of Figure 6.7. In addition, there are three forms which are not irreducible: *humidity* $\langle t_{11}$ ' $t_{23}\rangle$, *humidity* $\langle t_{12}$ ' $t_{25}\rangle$, and

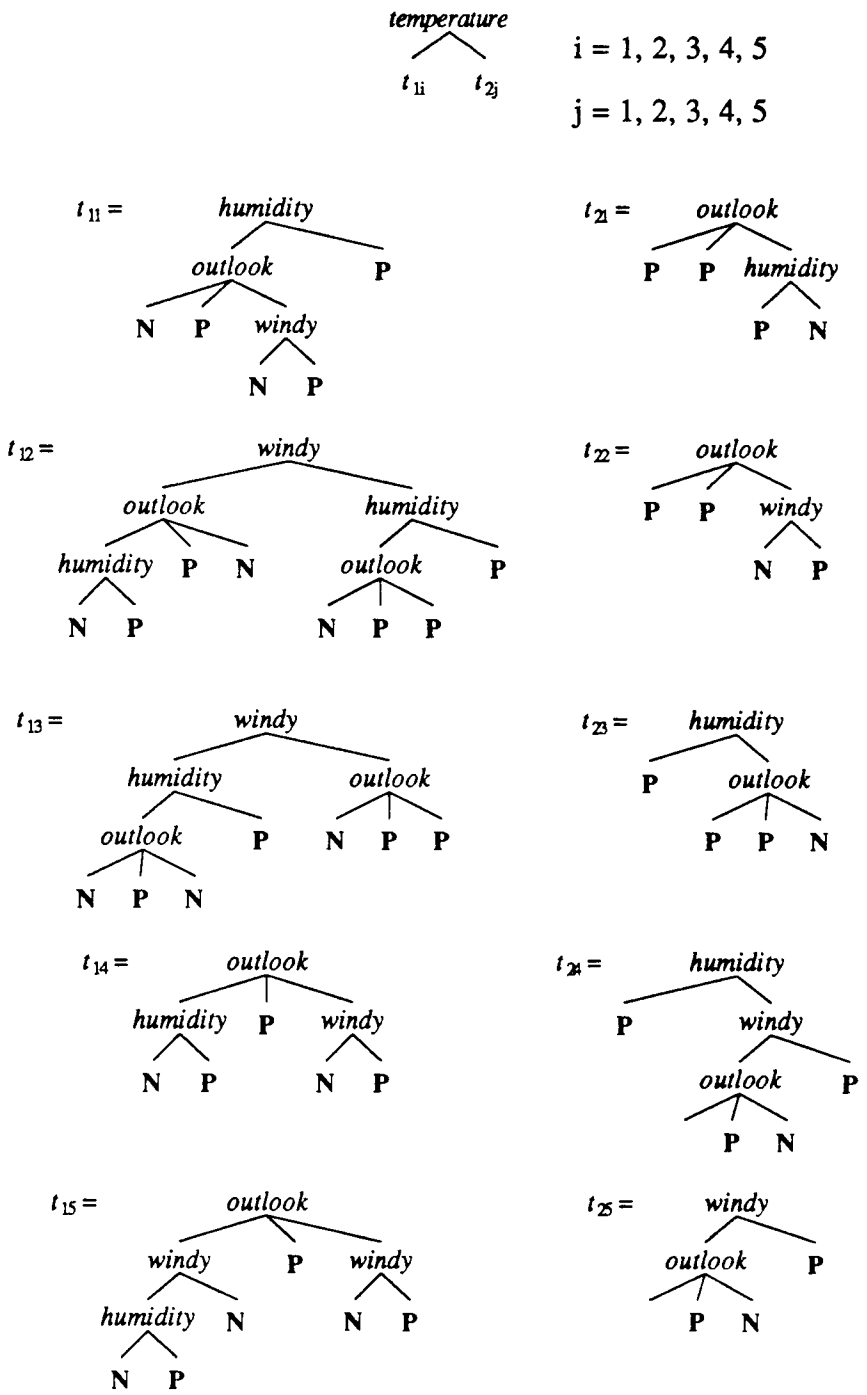



**Figure 6.7** Irreducible forms with *temperature* at the root.

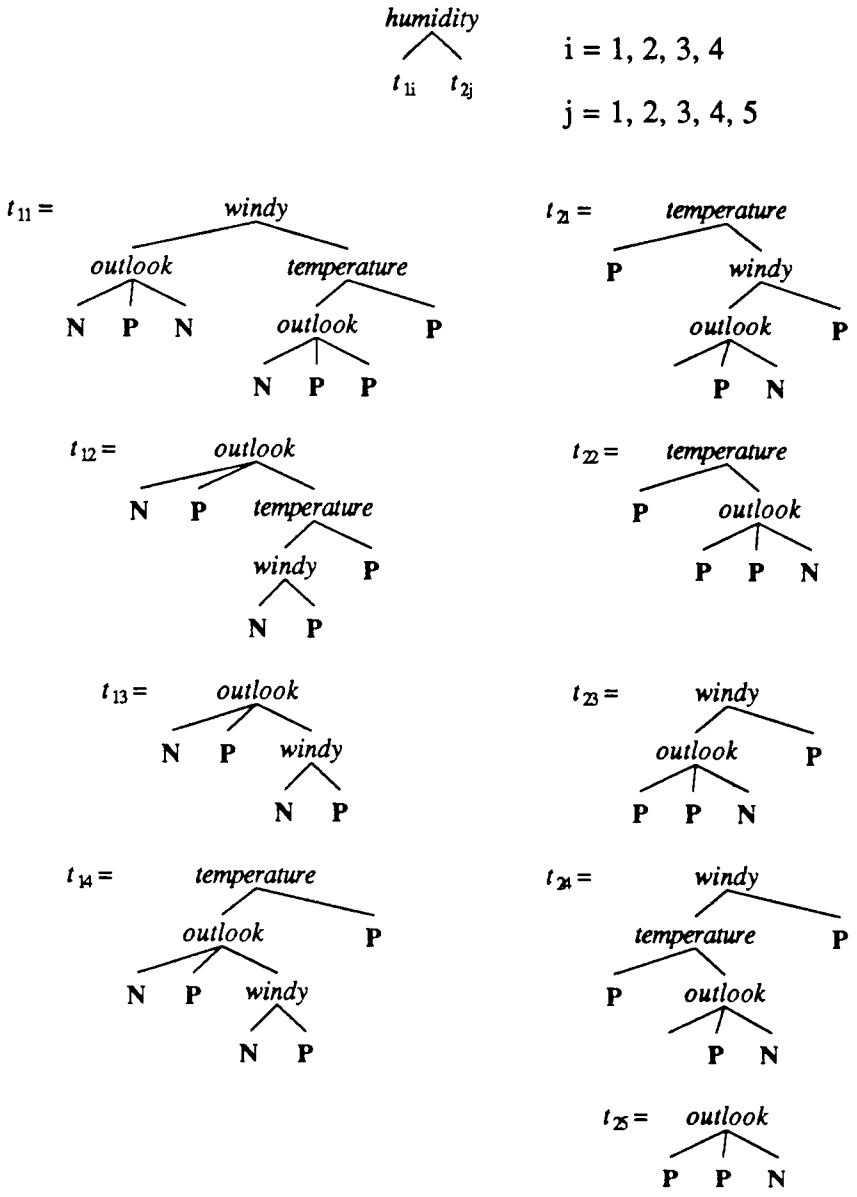
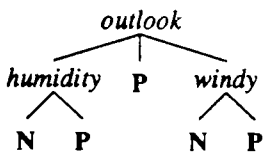



**Figure 6.8** Irreducible forms with *humidity* at the root.

*humidity* $\langle t_{13}$ ' $t_{25}\rangle$. And, of the irreducible forms, four are unstable: *humidity* $\langle t_{12}$ ' $t_{23}\rangle$, *humidity* $\langle t_{12}$ ' $t_{24}\rangle$, *humidity* $\langle t_{14}$ ' $t_{23}\rangle$ and *humidity* $\langle t_{14}$ ' $t_{24}\rangle$. Thus, only 11 irreducible forms with ***humidity*** at the root are stable.

Figure 6.9 shows the irreducible forms which have the decision *windy* at the root. There are three different subtrees that can follow if it is *windy*, and six if it is not, but only 12 of the 18 possible combinations are irreducible. The decision tree *windy* $\langle t_{12}$ ' $t_{22}\rangle$ is transposition equivalent to the tree *temperature* $\langle t_{13}$ ' $t_{25}\rangle$ of Figure 6.7. Similarly, *windy* $\langle t_{13}$ ' $t_{24}\rangle$ is transposition equivalent to *humidity* $\langle t_{11}$ ' $t_{24}\rangle$ of Figure 6.8. There are three combinations that are left out because they are not irreducible: *windy* $\langle t_{11}$ ' $t_{25}\rangle$, *windy* $\langle t_{11}$ ' $t_{26}\rangle$, *windy* $\langle t_{12}$ ' $t_{21}\rangle$, and *windy* $\langle t_{13}$ ' $t_{23}\rangle$. Finally, three of the irreducible forms are unstable: *windy* $\langle t_{12}$ ' $t_{23}\rangle$, *windy* $\langle t_{12}$ ' $t_{24}\rangle$ and *windy* $\langle t_{12}$ ' $t_{25}\rangle$. Thus, only 9 irreducible forms with *windy* at the root are stable.

Of these irreducible forms, only one is unconditionally stable:




This result agrees with that obtained by Quinlan using his ID3 algorithm. Given that the optimality of this form is not constrained by any conditions it is sensible then to use this form as the classification scheme for deciding when to play golf.

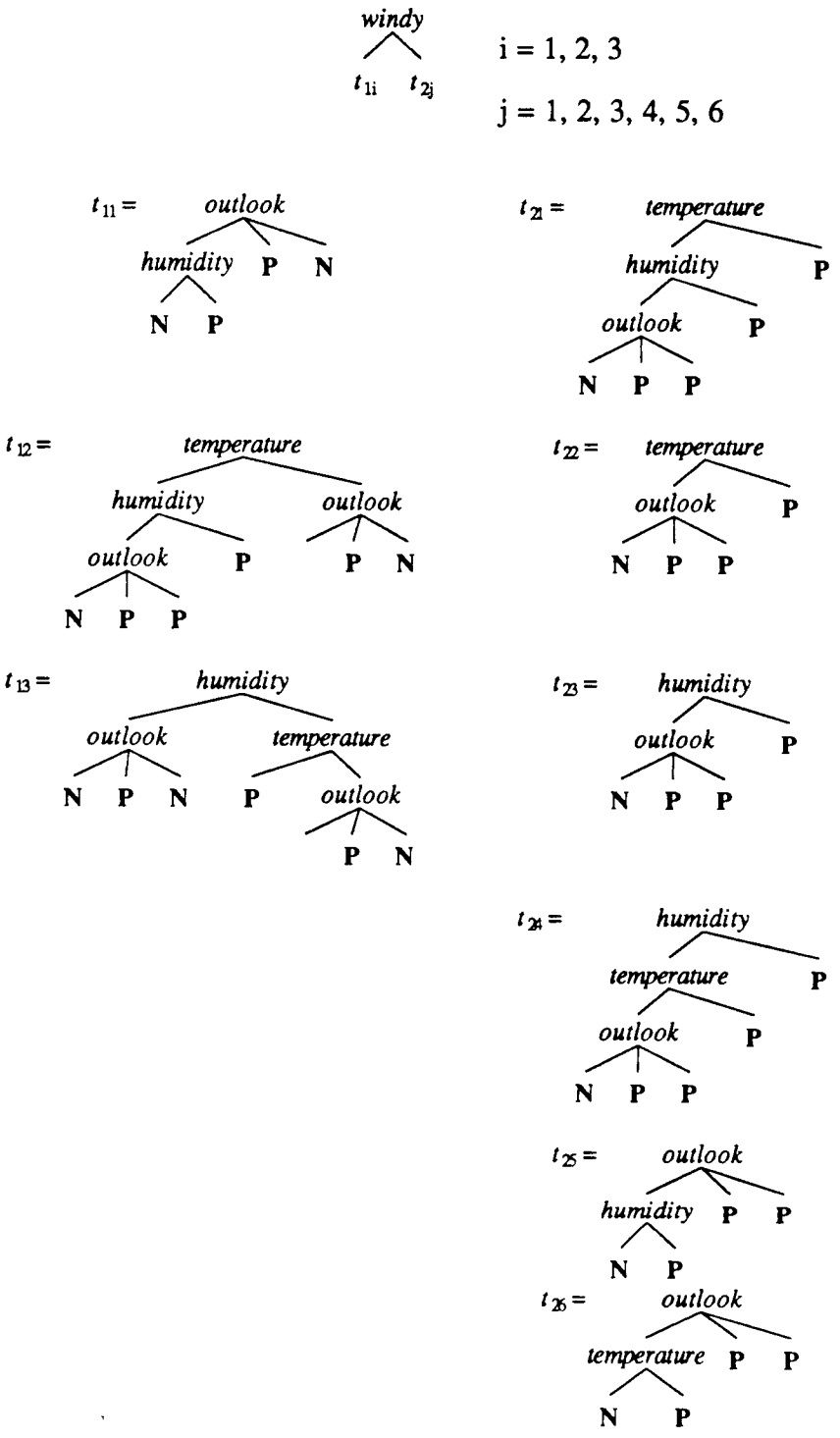



**Figure 6.9** Irreducible forms with *windy* at the root.

A total of 2.8 minutes of cpu time were required to generate all 48 stable irreducible forms consistent with the training examples given in Table 6.1. (Each irreducible form was obtained after an average of 3.5 seconds of cpu time.) ■

**Example 6.4**

Suppose that an optimal adaptive experiment (a sequential decision procedure for locating faults in an electronic circuit) is to be designed from a given fault table involving six binary tests. An example with this description was presented by Kohavi in [26]. This class of problems has frequently been singled out because of their complexity. In fact, in the discussion, Kohavi makes the statement that "in general, there is no algorithm for constructing the minimal tree, and an exhaustive trial and error is necessary." For instance, the number of decision trees corresponding to this problem exceeds $1.65 \times 10^{13}$. A computer program capable of producing a thousand trees per second would still require over 500 years to generate all of them.

In spite of the combinatoric nature of the problem, the techniques developed by this investigation were able to generate all the stable irreducible trees (totaling 439) in less than one hour of cpu time (an average of 8 seconds per irreducible). The rules used in this experiment are shown in Figure 6.10. One of the irreducible forms obtained by the DTD is shown in Figure 6.11. In particular, that decision tree was chosen because that is the optimal solution which Kohavi developed heuristically in [26]. One of the values, however, of using syntactic techniques is that they can also determine that

If the circuit passes tests 1, 2, 3, 4, 5 and 6, then it is fault-free.

If it passes tests 1, 3 and 4, and fails tests 2, 5 and 6, then fault 1 is present.

If it passes tests 1, 2, 3, 5 and 6, and fails test 4, then fault 2 is present.

If it passes tests 1, 3 and 5, and fails tests 2, 4 and 6, then fault 3 is present.

If it passes tests 1, 2, 4, 5 and 6, and fails test 3, then fault 4 is present.

If it passes tests 4, 5 and 6, and fails tests 1, 2 and 3, then fault 5 is present.

If it passes tests 2, 3 and 4 and fails tests 1, 5 and 6, then fault 6 is present.

**Figure 6.10** Rules for sequential testing procedure.

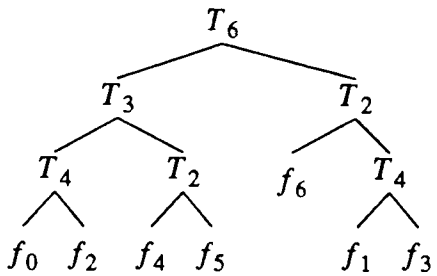



**Figure 6.11** An optimal sequential testing procedure.

the conditions for optimality for that particular tree are that the cost of test 2 is less than the cost of test 1 and that the cost of test 4 is less than the cost of test 5. The test for stability reduced the number of reasonable irreducible trees to be examined. Another interesting observation is that every one of the optimal solutions for this problem have some constraint associated with it. Thus, any knowledge as to the relative costs of the tests used in the process could be used to narrow down the number of candidate solutions even further. ■

**Example 6.5**

Finally, a sample expert system for solving a routing problem was developed using the DTD. The expert system consisted of 20 rules involving 24 different decisions. Since the decisions used were binary, the upper bound on the size of the tree which could be developed by backward chaining is in the order of $2^{24}$ or 16 million nodes. However, the actual size of the tree was considerably smaller and the system, running on a DEC VAX 8650 computer in a multi-user environment, was able to optimize the tree in about 5 minutes of cpu time.

By comparison, to solve the same problem by means of dynamic programming would have required

$$3^{24} + 24 \cdot 2 \cdot 3^{23} + 2^{24} \approx 4.8 \text{ terabytes of memory,}$$

provided that a single byte of memory is enough to store a node in the graph which is built by that method. Since, no facility with that kind of storage capacity was available, an implementation of that method was not attempted. In theory, however, dynamic programming would have required in the

order of $4.5\times10^{12}$ operations to solve this problem. Assuming that each operation takes only one nanosecond (highly unrealistically, since the calculations involved are not simple), it would still take 1.25 hours for the process to finish. ■

# CHAPTER 7

# CONCLUSIONS

The optimization techniques that were developed as the result of this investigation provide a methodology for the automatic generation of optimal decision trees. Moreover, the complexity of these techniques constitute an improvement over that of existing algorithms. The fact that the techniques developed are syntactic in nature, facilitates their use in automatic optimization applications where the knowledge of the semantics of the problem and the form of the cost criterion may not be available.

The recent development of an algebra for the manipulation of decision trees has allowed the implementation of an algorithm for generating all the irreducible forms of a decision tree. However, each irreducible is optimal only under certain conditions. These conditions provide a measure of the stability of the irreducible. In the absence of specific costing information, the more demanding the conditions for the stability of an irreducible, the less generally useful the tree. One of the examples presented in Chapter 6 illustrates a syntactic approach to decision tree inference in which all the irreducible decision trees which are consistent with respect to a given set of examples are generated, and the stability conditions are used to guide the selection of a reasonable inference tree.

There are several advantages to using the approach described above for knowledge acquisition. Each irreducible tree which is generated by this process can be viewed as an inferred knowledge cluster that provides some insight into the structure of the knowledge being acquired. Furthermore, this syntactic approach to learning can be implemented in an incremental manner.

There are several topics for further research in this area. One that deserves special attention is that of extending the techniques developed here to optimization of acyclic programs. In order to do this, the techniques have to be extended to include variables. Some research in this area is currently in progress. Also, the impact of parallel architectures on these techniques should be investigated.

# LIST OF REFERENCES

## LIST OF REFERENCES


[1] A. J. Bayes, "A Dynamic Programming Algorithm to Optimize Decision Table Code," *Austral. Comput. Journal*, vol. 5, no. 2, pp. 77-79, 1973.

[2] Y. Breitbart and A. Reiter, "A Branch-and-bound Algorithm to Obtain an Optimal Evaluation Tree for Monotonic Boolean Functions," *Acta Inf.*, vol. 4, pp. 311-319, 1975.

[3] Y. Breitbart and A. Reiter, "Algorithms for Fast Evaluations of Boolean Expressions," *Acta Inf.*, vol. 4, pp. 107-116, 1975.

[4] E. Cerny, D. Mange, and E. Sanchez, "Synthesis of Minimal Binary Decision Trees," *IEEE Trans. on Comput.*, vol. TC-28, no. 7, pp. 472-482, 1979.

[5] K. Chen, "Knowledge Representation Trees," Tech. Rep., Univ. of North Carolina at Charlotte, Department of Computer Science, 1984.

[6] K. Chen and Z. Ras, "An Axiomatic Theory of Information Trees," in *Proceedings of the 1985 CISS*, pp. 636-640, 1985.

[7] J. R. B. Cockett, "Coalgebraic Decision Theory," in *Mathematical Problems in Computation Theory*, (G. Mirkowska and H. Rasiowa, eds.), Banach Centre Publications, 1987.

[8] J. R. B. Cockett, "Decision Expression Optimization," *Fundamenta Informaticae*, vol. X, pp. 93-114, 1987.

[9] J. R. B. Cockett, "Discrete Decision Theory: Completeness," Tech. Rep. CS-87-69, Univ. of Tennessee Knoxville, Department of Computer Science, 1987. To appear in *Theoretical Computer Science*.

[10] J. R. B. Cockett, "Discrete Decision Theory: Manipulations," *Theoretical Computer Science*, vol. 54, pp. 215-236, 1987.

[11] J. R. B. Cockett, "The Theory of Decision Processes," Tech. Rep. CS-84-58, Univ. of Tennessee Knoxville, Department of Computer Science, 1984.

[12] J. R. B. Cockett and J. A. Herrera, "Decision Tree Reduction," Tech. Rep. CS-87-68, Univ. of Tennessee Knoxville, Department of Computer Science, 1987.

[13] J. R. B. Cockett and J. A. Herrera, "Prime Rules Based Methodologies Give Inadequate Control," in *Proc. of the ACM SIGART ISMIS*, (Knoxville, TN), pp. 441-449, Oct. 22-24 1986.

[14] J. R. B. Cockett, "Aspects of Expert Systems," (1987). To appear in *The Application of Avanced Computer Concepts and Techniques in Control Engineering*, NATO Avance Study Institute.

[15] P. R. Cohen and E. A. Fiegenbaum, *The Handbook of Artificial Intelligence*, vol. III, Los Altos, California: William Kaufman, 1984.

[16] G. R. Dattatreya and L. N. Kanal, "Adaptive Pattern Recognition with Random Costs and its Application to Decision Trees," *IEEE Trans. on Systems, Man and Cybernetics*, vol. SMC-16, pp. 208-218, March/April 1986.

[17] G. R. Dattatreya and L. N. Kanal, "Decision Trees in Pattern Recognition," in *Progress in Pattern Recognition 2*, (L. N. Kanal and A. Rosenfeld, eds.), pp. 189-239, Amsterdam: North Holland, 1985.

[18] G. R. Dattatreya and V. V. S. Sarma, "Bayesian and Decision Tree Approaches for Pattern Recognition Including Feature Measurement Costs," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. PAMI-3, pp. 293-298, May 1981.

[19] M. Garey, "Optimum Binary Identification Procedures," *SIAM Journal Applied Mathematics*, vol. 23, pp. 173-186, 1972.

[20] A. Gill, *Applied Algebra for the Computer Sciences*, Englewood, New Jersey: Prentice-Hall, 1976.

[21] Y. X. Gu, Q. R. Wang, and C. Y. Suen, "Application of Multi-layer Decision Tree in Computer Recognition of Chinese Characters," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. PAMI-5, pp. 83-89, Nov 1983.

[22] J. Halpern, "Evaluating Boolean Functions With Random Variables," *Int. J. Syst. Sci.*, vol. 5, no. 6, pp. 545-553, 1974.

[23] E. G. Hauska and P. E. Swain, "The Decision Tree Classifier: Design and Potential," in *Proc. 2nd. Symp. Machine Processing of Remotely Sensed Data*, 1975.

[24] J. A. Herrera and J. R. B. Cockett, "Finding all the Optimal Forms of a Decision Tree," Technical Report CS-86-66, Univ. of Tennessee Knoxville, Department of Computer Science, 1986.

[25] L. Hyafil and R. L. Rivest, "Constructing Optimal Binary Decision Trees is NP-Complete," *Information Processing Letters*, vol. 5, pp. 15-17, May 1976.

[26] Z. Kohavi, *Switching and Finite Automata Theory*, New York: McGraw-Hill, 1978.

[27] Y. K. Lin and K. S. Fu, "Automatic Classification of Cervical Cells Using a Binary Tree Classifier," in *Proc. 5th. Int. Conf. Pattern Recognition*, pp. 570-574, Dec 1980.

[28] A. Martinelli and U. Montanari, "Optimizing Decision Trees Through Heuristically Guided Search," *Communications of the ACM*, vol. 21, pp. 1025-1039, Dec 1978.

[29] S. W. Meisel and D. A. Michalopoulos, "A Partitioning Algorithm with Application in Pattern Classification and the Optimization of Decision Trees," *IEEE Transactions on Computers*, vol. TC-22, no. 1, pp. 93-103, 1973.

[30] R. S. Michalski, "Designing Extended-Entry Decision Tables and Optimal Decision Trees Using Decision Diagrams," Technical Report UIUCDCS-R-78-898, Univ. of Illinois at Urbana-Champaign, Department of Computer Science, 1978.

[31] R. S. Michalski, "Variable-valued Logic and its Application to Pattern Recognition and Machine Learning," in *Computer Science and Multiple Valued Logic. Theory and Applications*, (D. C. Rine, ed.), pp. 506-534, New York: North Holland, 1977.

[32] B. M. E. Moret, "Decision Trees and Diagrams," *ACM Comp. Surv.*, vol. 14, no. 4, pp. 593-623, 1982.

[33] B. M. E. Moret, M. G. Thomason, and R. C. Gonzalez, "The Activity of a Variable and its Relation to Decision Trees," *ACM Transactions on Programming Languages and Systems*, vol. 2, no. 4, pp. 580-595, 1980.

[34] H. J. Payne and W. S. Meisel, "An Algorithm for Constructing Optimal Binary Decision Trees," *IEEE Trans. Computers*, vol. C-26, pp. 905-916, Sep 1977.

[35] J. R. Quinlan, "Inductive Inference as a Tool for the Construction of High Performance Programs," in *Machine Learning*, (R. S. Michalski, T. M. Mitchell, and J. Carbonell, eds.), Palo Alto, California: Tioja, 1984.

[36] J. R. Quinlan, "Induction of Decision Trees," *Machine Learning 1*, pp. 81-106, 1986.

[37] L. T. Reinwald and R. M. Soland, "Conversion of Limited-entry Decision Tables to Optimal Computer Programs II: Minimum Storage Requirement," *Journal of ACM*, vol. 14, pp. 746-756, Oct 1967.

[38] L. T. Reinwald and R. M. Soland, "Conversion of Limited-entry Decision Tables to Optimal Computer Programs I: Minimum Average Processing Time," *Journal of ACM*, vol. 13, pp. 339-358, July 1966.

[39] E. M. Rounds, "A Combined Nonparametric Approach to Feature Selection and Binary Decision Tree Design," *Pattern Recognition*, vol. 12, pp. 313-317, 1980.

[40] Q. Y. Shi, "A Method for the Design of Binary Tree Classifiers," in *Proc. IEEE Conf. Image Processing and Pattern Recognition*, pp. 21-26, Aug 1981.

[41] K. Shwayder, "Combining Decision Rules in a Decision Table," *Comm. ACM*, vol. 18, pp. 476-480, 1975.

[42] A. Thayse, "P-functions: A New Tool for the Analysis and Synthesis of Binary Programs," *IEEE Trans. Comput.*, vol. TC-30, no. 2, pp. 126-134, 1981.

[43] A. Thayse, M. Davio, and J. P. Deschamps, "Optimization of Multiple-valued Decision Algorithms," in *Proc. of Int. Symp. on Multivalued Logic*, (Rosemont, PA), pp. 171-178, 1978.

[44] J. A. Vrba and J. A. Herrera, "Expert System for Automatic Decision Making in Target Detection and Classification," Final Report, U.S. Air Force Contract No. F04704-86-C-0138 (USAF/BMO), March 1987.

[45] J. A. Vrba and J. A. Herrera, "Expert System for Automatic Generation of Decision Tree Classifiers for Videotape Reading," Final Report, U.S. Army Contract No. DDAD10-86-C-0006 (US Army/WSMR), April, 1987.

[46] J. A. Vrba and J. A. Herrera, "Expert System for Pilot Decision Aiding," Final Report, U.S. Air Force Contract No. F33615-86-C-1088 (USAF/AFWAL/AAAS), January, 1987.

[47] Q. R. Wang and C. Y. Suen, "Analysis and Design of a Decision Tree Based on Entropy Reduction and Its Application to Large Character Set Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. PAMI-6, pp. 406-417, July 1984.

[48] S. Watanabe, "Pattern Recognition as a Quest for Minimum Entropy," *Pattern Recognition*, vol. 13, no. 5, pp. 381-387, 1981.

[49] K. C. You and K. S. Fu, "An Approach to the Design of a Linear Binary Tree Classifier," in *Proc. 3rd. Symp. Machine Processing of Remotely Sensed Data*, pp. 3-A-1-3-A-10, Purdue Univ., June-July 1976.

# VITA

Juan Antonio Herrera-Ball, the son of Juan Antonio Herrera Pérez and Gladys de las Mercedes Ball de Herrera, was born in Caracas, Venezuela on October 30, 1960. He attended High School at Colegio Santiago de León de Caracas and graduated in July 1978.

He entered the University of Tennessee, Knoxville, in January 1979. In August 1982, he graduated with high honors and received both a Bachelor of Science degree in Electrical Engineering and a Bachelor of Arts degree in Mathematics.

In September 1982, he was admitted to the Graduate School at the University of Tennessee, Knoxville. He obtained a Master of Science degree in Computer Science in March 1985, and a Doctor of Philosophy degree in Electrical Engineering in March 1988.

From February 1983 to August 1987, Herrera was a research assistant to Dr. R. C. Gonzalez in the Department of Electrical and Computer Engineering at the University of Tennessee, Knoxville. While there, he was involved in the development of an automatic license plate reader, the design and implementation of a decision tree-based alphanumeric character recognition system and various algorithms for pattern recognition and digital image processing applications.

In September 1987, Herrera joined Perceptics Corporation as a scientist in the Artificial Intelligence Applications Division. There, he has been conducting research in the areas of discrete decision theory, syntactic optimization of decision expressions, knowledge acquisition, expert system design and automatic software generation.

The author is a member of Tau Beta Pi, Eta Kappa Nu and several professional societies including I.E.E.E., A.C.M. and A.A.A.I.