THE UNIVERSITY OF CALGARY

Principled Induction from Feature Values

by

Daniel Jaliff

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA
JUNE, 1992

THE UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled, "Principled Induction from Feature Values" submitted by Daniel Jaliff in partial fulfillment of the requirements for the degree of Master of Science.

Mildred L. G. Shaw.

Supervisor, Dr. Mildred L. G. Shaw
Department of Computer Science

T. R. B. Cockett

Dr. Robin Cockett
Department of Computer Science

Dr. Brian R. Gaines
Department of Computer Science

Dr. Jack MacIntosh
Department of Philosophy

Date 29-June-92

ISBN 0-315-79151-9

Abstract

Principled induction is defined in this thesis as the process of arbitrarily selecting a valid description of a set of examples, and gradually simplifying this description until it is minimal in the preorder defined by the available simplification operators. If the preorder is well chosen, principled induction is a computationally feasible method for inducing descriptions. Algorithms are presented that perform principled induction of decision trees and rules from examples. These algorithms and empirical results obtained using them are shown to support the claim that principled induction is a clear and effective representation of the problem of learning concept descriptions from examples.

Acknowledgments

Numerous people from the Department of Computer Science contributed to the completion of this thesis. Brian Gaines encouraged me to undertake this project and was always supportive of my work. Rosanna Heise helped me formulate my research goals, and gave freely of her time for technical discussions. Debbie Leishman, Dave Maulsby, and Sonja Branskat read portions of the early drafts of the thesis; their feedback brought much clarity to the concepts I attempted to convey (but what obscurity remains is my own). John Aldwinckle, Todd Simpson, John Lewis, Bruce MacDonald, and Thong Phan provided valuable comments and suggestions during many discussions. The tutelage supplied by Gastón Groisman and Eric Schenk enabled me to negotiate safely the intricacies and pitfalls of LaTeX.

One person's contribution was absolutely essential; I am enormously grateful to Robin Cockett for his patient, committed, and insightful guidance.

Dedication

לרחמים דראב, ז"ל.

To Rahamim Darab, his memory be blessed.

Contents

List of Tables

List of Figures

Chapter 1

Principled induction in machine learning

This work investigates some of the methods that enable artificial systems to induce concept descriptions from examples defined by vectors of feature values. Each example is an instantiation of a template $(a_1, a_2, \ldots, a_n, c)$, where every a_i is a particular feature (e.g., color) that describes an entity, and c denotes a class membership. The task is to produce descriptions of the concepts c, to be used in determining the class membership of other entities taken from the same domain.

The first section of this chapter defines the notion of principled induction in the context of machine learning. This principled approach affords a clear and effective representation of the problem of learning concept descriptions from examples. In the second section, a number of well known algorithms are examined under the light of principled induction. The rest of the chapter formulates the goals of this thesis and gives an overview of the document.

Subsequent chapters propose algorithms that follow closely the definition of principled induction, and compare these algorithms to some of the methods reviewed in Section 1.2.

1.1 The elements of principled induction

Michalski's (1991) inferential theory of learning defines machine learning as a process of knowledge transformations: the creation or improvement of knowledge representations. The knowledge contained in the examples is represented in a convenient

fashion, and then transformed to achieve a particular goal. In the case of symbolic learning systems such as the ones to be discussed in this thesis (as opposed to neural net systems, for example), the theory requires that these knowledge transformations be performed in "explicit" and "conceptually comprehensible" steps.

1.1.1 Induction

To induce concept descriptions from training examples means to hypothesize descriptions that agree with the examples. A successful inductive conclusion from the examples is a hypothesis that explains them. The following formal definition is adapted from one by Genesereth and Nilsson (1987).

Definition 1.1 Let Γ be a set of statements in logic that embody some background knowledge about the domain from which the examples are taken. Let Δ be a database of observations about the domain (e.g., a set of examples), and let Ψ be a hypothesis, expressed in the same language as Δ and Γ. We say that Ψ is an *inductive conclusion* from Γ and Δ if the following conditions hold:

1. Δ is provable from Γ and Ψ: $\Gamma, \Psi \vdash \Delta$.
2. Ψ is consistent with Γ and Δ: $\Gamma, \Delta, \Psi \nvdash F$, where F denotes the logical constant *false*.

If the entities in Δ are described by attribute values, Ψ is an inductive conclusion, or a *valid description*, if it correctly determines the class of every entity in Δ.

While this definition does not provide a method for obtaining a valid description, it is easily verified that substituting Δ for Ψ satisfies both conditions. However, $\Psi = \Delta$ seems unsatisfactory as an induced description because it could only function

as a look-up table, and it does not appear to serve any transformation or learning goal.

This arguments raises some fundamental issues: the need to order hypothesized descriptions by some criterion, and the identification of conceptually comprehensible steps that can be used to transform a particular hypothesis into one that meets some predetermined goal.

1.1.2 Ordering descriptions

The relationship among various valid descriptions of a set of examples is best captured by a *preorder*: a relation that is reflexive, transitive, but not necessarily antisymmetric. The last condition allows for distinct descriptions that are nevertheless equivalent in the preorder.

Post (1960) examined the concept of *simplicity* in scientific hypotheses; he stated that simplicity is a quality whose desirability goes beyond a subjective, aesthetic appraisal — it also embodies the concept of generality, or breadth of application. A simple hypothesis is also a "good mnemonic." From this point of view, a simple hypothesis is of significant value in artificial systems; it has lower storage requirements than a less simple hypothesis, and is easier for a human user to understand.

Sober (1975) put forward a more formal definition of simplicity: A hypothesis H_1 is more simple than H_2 if H_1 requires less additional information than H_2 to answer a particular question to which both are relevant. This formulation induces an *informativeness preorder* on a set of hypotheses. If the hypotheses are concept descriptions generated from vectors of feature values, then H_1 is simpler if it needs to reference fewer of the entity's features than H_2 in order to determine an entity's class membership.

Sober's formulation is consistent with Post's intuition. If a simple hypothesis needs to reference fewer of an entity's features, then it is more compact than others. It has greater breadth of application, since it is not restricted by the need to reference irrelevant attributes. Post correctly equated the problem of finding the simplest hypothesis to explain a phenomenon to that of finding an optimal coding for the phenomenon.

However, the task of finding such an optimal coding constitutes an insurmountable hurdle from the computational point of view. The optimization of decision trees and decision rules, two representations commonly used in symbolic machine learning systems to represent induced descriptions, is NP-Complete (Hyafil & Rivest, 1976; Wegener, 1987).

One common way of generating a preorder on a description space is to define a mapping from the set of possible descriptions to the non-negative real numbers. For example, the optimization problem for decision trees can be defined as the goal of minimizing the sum of the path lengths in the tree (Hyafil & Rivest, 1976). This mapping defines a total preorder on the decision trees that describe a particular domain, such that there exists an optimal element — one that is minimal with respect to the entire order. But since the task of finding this optimal element is NP-Complete, it is infeasible for all but the smallest instances of the problem.

The approach used in some of the induction methods to be reviewed in Section 1.2 is to build descriptions from scratch, augmenting them by making locally correct (or "greedy") choices. This bypasses the computational difficulty of true optimization, and typically results in good suboptimal solutions.

Another approach, used in the algorithms presented in later chapters, is to generate a partial preorder on the descriptions, randomly select a class of comparable

descriptions, and find one that is minimal in the preorder. The effect is similar to that of the greedy approach; any minimal element in the preorder can be expected to be a good, though not provably optimal solution. The search for this minimal element is a local search and thus does not explore the entire space of possible descriptions.

It must be noted that in spite of the ideas of Post and Sober, the preorder imposed on the valid descriptions of a set of examples is not fixed, since optimization goals may vary from situation to situation (Gaines, 1977). In general, it is convenient to speak of a *reduction preorder* ($\leq_{red}$) on the space of valid descriptions, without committing to the particular features of the relation.

Definition 1.2 A *description space* is a logical calculus of descriptions (with operations and rules of inference) together with a reduction preorder ($\leq_{red}$) on descriptions.

1.1.3 Transforming descriptions

Having discussed the concept of a reduction preorder, it is now possible to turn to the task of transforming descriptions in order to achieve a particular goal. We begin with a definition of generalization by Niblett (1988).

Definition 1.3 Given background knowledge Γ and statements S_1 and S_2, S_1 is a *generalization* of S_2, or is *more general* than S_2 ($S_1 \geq_g S_2$), if $\Gamma, S_1 \vdash S_2$

This definition states that wherever S_2 is applicable, S_1 is also applicable; any observations that are explained by S_2 are also explained by S_1. Requiring that the transformation of a valid description result in another description that is a generalization of the original one guarantees that there will be no loss in the breadth of applicability.

Definition 1.4 Given description Ψ of observations Δ with respect to background knowledge Γ, and a reduction preorder $\leq_{red}$ on the descriptions of Δ, t is said to be a *valid transformation* if

1. Ψ is a valid description implies that $t(\Psi)$ is also a valid description,
2. $t(\Psi) \geq_g \Psi$, and
3. $t(\Psi) \leq_{red} \Psi$.

A valid transformation is a generality preserving mapping of the set of valid descriptions into itself. It transforms a valid description into another that is at least as general, and at least as reduced, as the original description. The task of formulating valid transformations is extremely awkward if one starts from a given reduction preorder, and then attempts to define transformations that preserve the preorder, as well as the validity and generality of descriptions.

The strategy used in later chapters is to start from operations that preserve the validity and generality of descriptions, and also improve the descriptions by some criterion — or at least do not make them any worse. These operations can then be used to generate a reduction preorder on the space of valid descriptions, and be treated as valid transformations.

Definition 1.5 A reduction preorder is *transformational* (written as $\leq_{redt}$) if it is generated by the operations and rules of inference of a description space. A description space is *transformational* if its reduction preorder is transformational.

A set of generality and validity preserving operations on a set of descriptions becomes a set of valid transformations when the reducing preorder used is the transformational reducing preorder induced by the operations. Then it becomes easy to

identify elements that are minimal in the preorder: an element Ψ is minimal in $\leq_{redt}$ if and only if there is no transformation t such that $t(\Psi) \leq_{redt} \Psi$ and $\Psi \not\leq_{redt} t(\Psi)$.

The concept of principled induction may now be formulated.

Definition 1.6 Ψ is a *principled inductive hypothesis* from background knowledge Γ and observations Δ if

1. Ψ is an inductive hypothesis from Γ and Δ by Definition 1.1 (e.g., Ψ is in the description space for Δ).
2. Ψ is minimal in the transformational reduction preorder for the description space.

The following definition gives a general procedure for obtaining principled inductive hypotheses.

Definition 1.7 Given observations Δ, background knowledge Γ, a transformational description space for Δ with operations and rules of inference that are valid transformations, and a transformational reduction preorder $\leq_{redt}$ defined by the valid transformations, Ψ is obtained by *principled induction* if:

1. Ψ_0 is some arbitrary valid description (an inductive hypothesis by Definition 1.1).
2. Ψ is obtained from Ψ_0 by application of valid transformations, and Ψ is minimal in $\leq_{redt}$.

In other words, principled induction is the process of selecting an arbitrary valid description of the observations, and applying valid transformations until a minimal element in the preorder is obtained. The intent of stipulating that the initial valid description be selected arbitrarily is to allow for the use of an easily obtained valid

description. In practical terms, principled induction is the process of producing, inexpensively, an accurate working hypothesis, and then improving that hypothesis by small, well defined steps, until no further simplification is possible.

1.2 Some induction algorithms

As part of an introduction to principled induction, it is useful to survey some well known algorithms that induce concept descriptions from vectors of attribute values. These algorithms are described in some detail, and the descriptions are enriched with simple examples. The reader who is already familiar with these algorithms will benefit from the portions of the discussion that analyze them in terms of whether they can be characterized as forms of principled induction. The time complexity of two of the algorithms is also examined, as these measures will be used for comparisons in later chapters.

1.2.1 Version space

The version space learning algorithm, due to Mitchell (1978), is based on a reflexive, antisymmetric partial order on the space of all possible descriptions in a language. This ordering is indeed a generality ordering, in the sense of Definition 1.3. The entire relation can be represented as a graph, called the *version space* graph.

Example 1.1 Given a domain in which entities are described by the features *color* (one of $\{blue, red\}$) and *size* (one of $\{small, large\}$), the version space graph of concept definitions in the domain is as shown in Figure 1.1. An arrow from node v_1 to node v_2 indicates that v_1 is more general than v_2. Thus, the top element in the graph, $(*, *)$, which represents a description in which neither of the features is

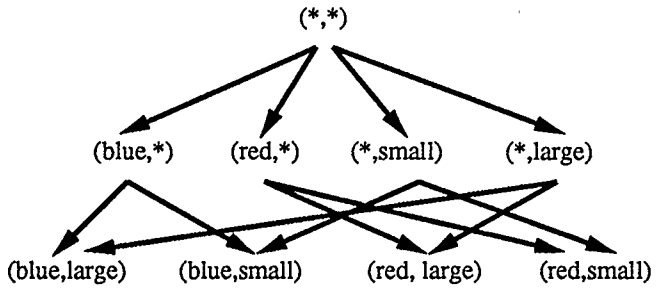

Figure 1.1. The version space of concepts in *color* and *size*.

instantiated, is more general than any other description. The four minimal elements in the relation are the descriptions with both features instantiated. As long as nothing is known about the concept to be learned (e.g., no examples have been made available), any node in the version space graph is a possible description of the concept.

For the purpose of learning concept descriptions, the version space V is not represented explicitly, but rather by two boundary sets of nodes:

1. The most general boundary set: $G = \{g | \neg\exists v \in V, v \neq g \wedge v \geq_g g\}$.

2. The most specific boundary set: $S = \{s | \neg\exists v \in V, v \neq s \wedge s \geq_g v\}$.

The algorithm uses positive and negative examples of the concept to be learned (the *target concept*) to specialize the G set and to generalize the S set. The examples are presented incrementally, and the boundary sets are modified to create a new version space, whose nodes constitute the set of possible concept definitions that are complete and consistent with respect to the examples that have been presented. If and when the boundary sets meet, $G = S = \{v\}$, and v is the concept description.

The G set is initialized to the most general element in the graph, and the S set is initialized to the first positive example. With every additional positive example

e, G is updated by discarding any elements that are no more general than e. The specific boundary set S is updated to the least general elements in the graph that are more general than e. Positive examples are used to ensure that the S set is as general as possible.

If the example e is negative, any elements in S that are more general than e must be dropped, and G must be replaced by the set of least general graph nodes that are not more general than e. This ensures that the G set is only as specific as necessary to avoid covering negative examples, so that all of the possible definitions in the new version space are consistent with respect to the available observations.

Example 1.2 Consider the task of learning the concept "blue" in the description space depicted in Figure 1.1. Let the positive examples be (*blue,large*) and (*blue,small*), and let there be a single negative example (*red,small*). The boundary sets are initialized to $G = \{(*,*)\}$ and $S = \{(blue,large)\}$. Let the next example be the negative one, (*red,small*). The current S set does not contain any elements that are more general than (*red,small*), so what remains is to replace G by the set of least general nodes that are not more general than the example. The single element of the current G is more general than (*red,small*), so the set is replaced by $\{(blue,*),(*,large)\}$. At this point the G and S sets define a version space with three nodes, or possible concept descriptions: $\{(blue,large),(blue,*),(*,large)\}$. The second positive example, (*blue,small*) is not covered by (*,large), so the latter node must be dropped from the G set, resulting in $G = \{(blue,*)\}$. The only element in the S set is not more general than (*blue,small*), so S is updated to $\{(blue,*)\}$. The boundary sets have met, and the learned concept is "blue, of any size", which may be interpreted as "blue".

It is easy to show that this algorithm learns concept descriptions by principled induction. The goal of the transformation process is to reduce the size of the set of possible concept descriptions, such that a single possible description remains. The initial guess Ψ_0 is obtained at little cost by creating an S set with an arbitrary positive example as its only member. The updatings of the boundary sets preserve the validity of the possible concept descriptions in the new version space — that is precisely their nature. Generality is preserved by updating the S set with nodes that are more general than the ones they replace.

The updating operations induce a preorder on the valid version spaces. This preorder may be thought of as one of *tightness*: if an example triggers any change at all in the version space, the updated version space will have a G set that is at most as general as the previous G set, and — as discussed in the previous paragraph — an S set that is at least as general as the previous one.

It could be argued that since the learning algorithm is strictly incremental, in the sense that it maintains consistency and generality with respect to one example at a time, a given state of the version space cannot be judged to be a valid description of all the examples. For example, the initial version space, defined on the basis of the first positive example, is not necessarily consistent with respect to future negative examples.

However, consider the version space after k examples have been processed. To perform principled induction from those k examples, it is sufficient to select the current version space as Ψ_0. Since no valid transformations — or steps of the algorithm — can be performed without additional examples, the current version space is minimal in the preorder, and is the result of the principled induction task. The previous, incremental processing of the k examples guarantees that the readily

available initial guess is in fact minimal in the preorder.

1.2.2 Learning Boolean functions

The task of simplifying a Boolean function is the same as learning a concept description from a set of examples made up of bit vectors for which the function is known to have value "1". While this is only a restricted form of the problem of inducing decision rules, the Quine-McCluskey simplification method (McCluskey, 1956; Quine, 1955; Quine, 1952) is a clear example of principled induction and should be examined in this survey as well.

The input to the algorithm is a set of N positive examples v defined in binary attributes $X = \{x_1, \ldots, x_n\}$. The output is a set of vectors, each of which is defined in some subset of X. The set is interpreted as a disjunction of conjunctive clauses.

In dealing with binary attributes whose values may be "0" or "1", it is customary to represent $x = 1$ by x and $x = 0$ by $\bar{x}$. Thus the output set $\{x_1\bar{x_2}, x_3\}$ is interpreted as the description "x_1 takes the value one and x_2 takes the value 0, or x_3 takes the value 1." Each element v in the output set is interpreted as a conjunctive decision rule of the form $v \Rightarrow 1$. For example, $x_1\bar{x_2}$ is interpreted as

$$(x_1 = 1) \wedge (x_2 = 0) \Rightarrow 1.$$

Initially, the algorithm looks for all pairs of examples that differ on exactly one attribute x_i. For every such pair (v_1, v_2), a new example v' is generated, such that v' is defined in $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$: v' is of length $n - 1$, and is equal to both v_1 and v_2 on every x_j, $j \neq i$. The examples v_1 and v_2 are marked as "used" to indicate that they have participated in the generation of a new vector; this does not preclude them from appearing in other pairs. The process is repeated with the all the new vectors of length $n - 1, n - 2, \ldots$ until no pairs of vectors of the same length can

be found that differ on exactly one attribute. Notice that vectors of length $n-1$ or less may be defined on entirely different attributes, and may therefore not be comparable. The output is the set of vectors that remain unmarked.

#	x_1	x_2	$x_1 + x_2$
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	1

Table 1.1. A table for the Boolean function "or" (+).

Example 1.3 The Boolean function "or" is defined in Table 1.1. The positive examples are numbers 2 ($\bar{x_1}x_2$), 3 ($x_1\bar{x_2}$), and 4 (x_1x_2). Examples 2 and 3 differ on both attributes, so they may not be paired to generate a simplified vector. On the other hand, 2 and 4 differ on the value of x_1; they are be used to generate a vector x_2, and then marked as used. The same is true of 3 and 4, which generate x_1. The two vectors of length one are defined on different attributes, so no further simplification is possible. Both vectors are unmarked, so they appear in the output set $\{x_1, x_2\}$, interpreted as two decision rules: If attribute x_1 has value 1, the Boolean function "or" has value 1; else if x_2 has value 1, the function has value 1.

It must be noted that the output of this algorithm is not optimal. All the algorithm guarantees is that every conjunction is *prime*: if any attribute were dropped, the conjunction would cover some case defined in the original n attributes that was not included in the original set of positive examples. In order to truly optimize the output, it is necessary to identify and remove any redundant conjunctions — that is, conjunctions that only cover examples which are also covered by other examples. The

removal of redundant conjunctions, and hence the optimization of Boolean function definitions, is NP-Complete (Wegener, 1987).

In order to understand why this simplification algorithm is an example of principled induction, one must note that the reduction of two vectors that differ on one attribute to a shorter vector without that attribute preserves the generality and consistency of the entire set of unmarked vectors — the potential output. The new vector is in fact more widely applicable than either of the two original ones, and it is consistent because the eliminated attribute is binary. The reduction operation induces a reducing transformation preorder on the valid sets of unmarked vectors, and constitutes a valid transformation with respect to that preorder.

1.2.3 Top-down induction of decision trees

Concept definitions can be represented as decision trees, where every internal node in the tree represents a query about the value of a particular attribute, and every leaf represents a class or concept name. The decision tree of Figure 1.2 is a representation of the Boolean function "or" described in Table 1.1.

The classical approach to inducing decision trees is to build them from the top down, using local criteria to select the test attribute at every new node (Breiman, Friedman, Olshen & Stone, 1984; Quinlan, 1986b).

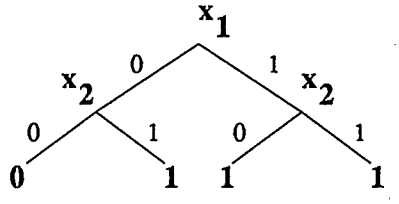

Figure 1.2: A decision tree representation of the "or" function, described in Table 1.1.

Given a set S of entities that belong to different classes, one would like to select as

the first attribute — the root of the decision tree — one that divides S into subsets $S_1, \ldots, S_n$, such that the subsets are more homogeneous than S. More explicitly, suppose that the most frequent class among the N entities of S is c, of which there are k cases. Then the frequency of c in S is $\frac{k}{N}$. Let c_i be the most frequent class in each S_i. One would like the average frequency of the c_is to be higher than $\frac{k}{N}$; this means that the subsets are "closer" to containing only elements of a single class. The process is repeated on every S_i, until the resulting subsets are perfectly homogeneous.

The crux of this method is the selection of an adequate attribute on which to split the set of examples. While the choice of an attribute is irrelevant to the correctness of the tree representation of the concepts, or classes, contained in S, there are choices that will produce more compact representations.

In order to select a test that will lead to a simpler representation, the attributes that are available to partition S are rated on their *impurity* (Breiman et al., 1984). An impurity function $i(S)$ assigns a nonnegative value to a set of examples and a possible test attribute a. This function must take on its maximum value when every class that is present in S occurs with the same frequency, and its minimum value when only one class occurs in S.

In order to rate an attribute $a \in A$, where A is the set of attributes available for partitioning S, a is used to partition the examples into $S_1, \ldots, S_n$, and the impurity of every S_i is calculated. These impurities are weighted by the sizes of the subsets and averaged. The attribute that is selected is the one that minimizes the weighted average of the impurity measures.

One of the impurity functions proposed by Breiman et al (1984) is

$$i(S) = -\sum_{i]1}^{k} p(c_k) \cdot log(p(c_k))$$

where $c_1, \ldots, c_k$ are the classes that occur in S. This is the *entropy* function defined

by Shannon (Shannon & Weaver, 1948); it is used with 2 as the logarithmic base in the tree induction systems ID3 (Quinlan, 1986b) and C4.5 (Quinlan, 1990b).

Example 1.4 In this example, the ID3 algorithm is used to induce a tree from the data of Table 1.1. Partitioning on x_1 would result in subsets $\{1,2\}$ and $\{3,4\}$, corresponding to the examples with values 0 and 1, respectively, on x_1. The first subset has one example of class 1 and one of class 0; the entropy of the set is

$$-(0.5 \cdot log_2(0.5) + 0.5 \cdot log_2(0.5)) = -(-0.5 - 0.5) = 1.$$

The second subset has two examples of class 1, so its entropy is 0.
Partitioning by x_2 would produce two subsets with the same entropies, so the choice is irrelevant. Let us select x_1; the subset $\{3,4\}$ requires no further partitioning, since both entities belong to class 1. A leaf node is created and labeled with that class. The subset $\{1,2\}$ still requires partitioning. The only remaining attribute is x_2, and

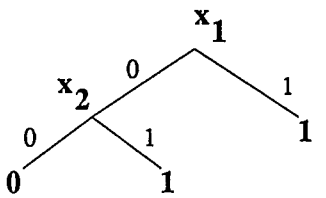

Figure 1.3. The tree induced top-down from Table 1.1.

splitting on this feature produces two singleton subsets, obviously with entropy 0. The induced tree is shown in Figure 1.3.

The top-down approach is clearly not principled induction. There is no initial guess as such; the unfinished tree does not qualify as a valid description of S. The tree is "grown" by specialization, rather than by generality preserving simplification. It is true that the finished tree could be used as the initial valid description of a

principled induction process; however, as discussed below, the cost of producing this initial guess is far from negligible.

To analyze the cost of this approach to tree induction, consider a set S of training examples, described by the set of attributes A. In order to select some $a \in A$ to partition S, the set must be partitioned by every a, and the impurities of the resulting subsets must be measured. Therefore, every member of S must be examined on the value of every attribute at a cost of $|S| \cdot |A|$. That is in effect the cost of constructing every level of the tree; since the number of levels is bounded by the number of attributes, the time complexity of the entire process is $O(|S| \cdot |A|^2)$. Moreover, the average running time of the algorithm can be expected to approximate this upper bound.

Trees induced by this method are not guaranteed to be optimal because test attributes are selected by minimizing the average heterogeneity of the subsets resulting from a partition, which is a purely local criterion. True optimization would require that these selections be made by examining the finished decision trees that would arise from the various selections of test attributes for the current node of the tree.

1.2.4 Statistical induction of decision rules

PRISM (Cendrowska, 1987) is an algorithm that adapts the statistical approach of ID3 to induce sets of decisions rules. Like the Quine-McCluskey algorithm, PRISM produces a disjunction of conjunctive decision rules. PRISM, however, is capable of learning from examples described by n-ary attributes.

A more significant difference between the Quine-McCluskey algorithm and PRISM is that the latter grows rules by specialization; this disqualifies it from being a principled induction algorithm.

Decision rules are created by finding tests, or attribute-value pairs, of the form $(attribute = value)$. Cendrowska treats these pairs as discrete messages in an information system. The information, in bits, contributed by such a pair (abbreviated as $(a = x)$) to the classification of an entity as a member of class c, is defined by

$$I(c|(a = x)) = log_2(\frac{p(c|(a = x))}{p(c)}).$$

The interpretation of the fraction in the right hand side is: "the probability that the class is c after it is known that the entity has value x on feature a, divided by the probability that the entity belonged to class c before the value of its feature a was known." Rules are grown by adding to them attribute-value pairs that maximize the information function.

In order to build a set of rules from training examples from classes $c_1, \ldots, c_n$, PRISM does the following, for each c_i.

1. Selects the attribute-value pair $(a = x)$ that maximizes the information function, adding the pair as a test to the conjunctive rule for class c_i. It then creates the subset of entities that meet this condition.

2. Repeats step 1 until the subset contains only members of class c_i, and then removes the remaining subset from the original training set.

3. Repeats steps 1 and 2 until no entities of class c_i remain in the training set.

As anticipated above, PRISM is not a principled induction algorithm. There is no initial valid description; the final description is grown by progressive specialization from an empty rule set, with no intermediate valid descriptions.

This algorithm is somewhat less efficient that top-down tree induction. For every rule, every entity could be examined on every attribute with every test that is added

to the rule. Since a rule can be as long as the number of features, given a set of attributes A, the cost of creating rule from examples S is $|S| \cdot |A|^2$. Furthermore, the number of rules is bounded by number of examples, so the overall time complexity is $O(|S|^2 \cdot |A|^2)$. However, unlike with the tree induction method, this is in fact a very pessimistic estimate, and the average time can be expected to be better.

1.2.5 Other algorithms

The algorithms described above exemplify some of the basic approaches to the induction of descriptions from attribute vectors, and serve as positive and negative examples of principled induction.

There are other effective induction algorithms worthy of mention. For example, the program AQ11 succeeded in learning a set of rules for the diagnosis of plant pathology, that performed better than rules generated by human experts from the same recorded cases described in terms of feature values (Michalski & Chilausky, 1980). The system's basic method is not unlike that used in PRISM: decision rules are augmented with conjuncts that cover many positive and few negative examples of the target concept. The basic AQ11 algorithm has been extended in systems such as AQ15 (Michalski, Mozetic, Hong & Lavrac, 1986) and Einstein (Webb, 1991); these extensions enable the algorithm to cope with attributes whose values come from continuous domains (as opposed to discrete attributes), and with partially corrupted examples. INDUCT (Gaines, 1991) is an extension of PRISM that performs well on corrupted data. However, like the basic methods that underlie them, these effective algorithms fail to qualify as forms of principled induction.

1.3 Goals of this thesis

The goal of this thesis is to argue that the principled approach to induction presented in this chapter provides a clear and effective representation of the problem of learning concept descriptions from examples. The principled induction algorithms presented in later chapters are not intended to provide solutions that are necessarily more efficient or effective than those reviewed in the previous section, but rather to support this claim.

It will be shown how differences in the preorders used for principled induction affect the efficiency of the induction process and the quality of the results. These differences highlight the tradeoff between the cost of the principled induction process and the degree of optimization achieved.

Furthermore, experiments comparing the performance of principled induction to the statistically based method of Section 1.2.3 shed some light on the value of statistical criteria when induction is performed from corrupted data.

1.4 Overview of the chapters

Chapter 2 reviews an algebraic theory for the manipulation of decision trees and an efficient algorithm for simplifying decision trees. Chapter 3 explains the application of the tree reduction algorithm to the task of learning concept definitions by principled induction from vectors of feature values. It is shown in that chapter that the original tree reduction algorithm fails to achieve the degree of simplification that can be expected of statistical methods. A new preorder is defined on the terms of a decision theory to support a modified tree reduction algorithm that achieves greater simplification.

Chapter 4 discusses the application of these methods to the task of inducing concept definitions from incomplete and noisy data, and examines the value of statistical criteria when inducing decision trees in noisy domains. Some empirical results are presented to support the effectiveness of the principled induction algorithm presented in Chapter 3, and the extensions discussed in the current chapter.

Chapter 5 presents an algorithm for inducing compact definitions in the form of prime decision rules, which are extracted by principled induction from a decision tree representation of the training examples.

The last chapter summarizes the thesis and discusses its contribution.

Chapter 2

Algebraic decision theory

This chapter presents an algebraic formulation of decision trees, developed by Cockett (1987a, 1987b, 1988). The formulation is used to define operations on decision trees, which in turn generate a preorder on the trees. An algorithm is defined, using those operations, to find minimal elements in that preorder (Cockett & Herrera, 1990). In Chapter 3 the algorithm will be used to perform principled induction of decision trees.

2.1 Discrete decision theory

This section presents an algebraic theory that can be used to model decision processes in which every atomic test is *discrete*: it has a predetermined, finite number of outcomes.

Definition 2.1 A *decision* q is an operation, with an associated arity $n \geq 2$ (written as $arity(q) = n$), that partitions a set S into n disjoint subsets:

$$q : S \mapsto S + \ldots + S,$$

that is, q partitions a set S into $S_1, \ldots, S_n$, such that

$$\bigcup_{i=1}^{n} S_i = S$$

and for $1 \leq i, j \leq n$, $i \neq j$ implies that $S_i \cap S_j = \emptyset$.

The following example shows how this definition of a decision models the atomic component of a decision process.

Example 2.1 $S = \{e_1, e_2, e_3\}$, where each element is in turn a set of n expressions $(q \diamondsuit j)$: q is a decision that appears exactly once in a given e_i, and $1 \leq j \leq arity(q)$. These expressions simply associate the value j with decision q. Let $arity(q_1) = 2$; the arities of $q_2, \ldots, q_n$ may remain unspecified. Suppose that $e_1 = \{(q_1 \diamondsuit 1), (q_2 \diamondsuit 3), \ldots (q_n \diamondsuit 5)\}$, $e_2 = \{(q_1 \diamondsuit 2), (q_2 \diamondsuit 7), \ldots, (q_n \diamondsuit 3)\}$, and $e_3 = \{(q_1 \diamondsuit 1), (q_2 \diamondsuit 7), \ldots, (q_n \diamondsuit 4)\}$. It is useful to think of q_1 as a gate with one input channel, and $arity(q_1)$ output channels, each labeled with a number from $\{1, \ldots, arity(q_1)\}$ (see Figure 2.1). Decision q_1 redirects each e_i in a way that is determined by the value associated with q in e_i.

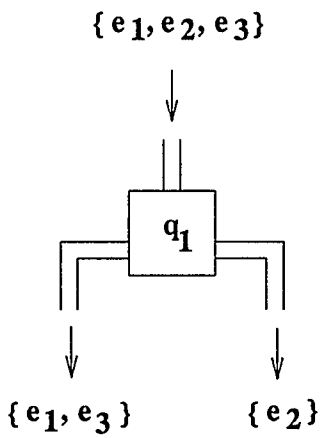

Figure 2.1. The interpretation of decision q_1 with arity two.

In the following definition, decisions are composed to form terms, which are expressions analogous to decision trees.

Definition 2.2 Given a set of decisions Q and a stock of variables $x_1, x_2, \ldots, x_n, \ldots$, a *term* defined in Q and the variables is an expression t formed as follows:

- If t is one of $x_1, x_2, \ldots, x_n, \ldots$, it is a term.

- If $t_1, \ldots, t_n$ are terms, q is a decision in Q, and $arity(q) = n$, then $q\langle t_1 | \ldots | t_n \rangle$ is a term. In such a term, $t_1, \ldots, t_n$ are the *arguments* or *subtrees* of q.

A term t also partitions a set into disjoint subsets, by the following rules:

- If $t = q\langle t_1 | \ldots | t_n \rangle$, then partition the set with q, redirecting the disjoint subsets into the corresponding t_is; that is, redirect the output from channel i into t_i.
- If $t = x$, the partition is complete.

Intuitively, the variables in the term are possible outcomes of the decision or classification process. For every variable that occurs in the term, there is a "bucket" labeled with the variable name. As shown in Figure 2.2, elements of the set that are redirected to a variable x are deposited in the bucket labeled "x". This corresponds to our understanding of decision processes as represented by decision trees. Every

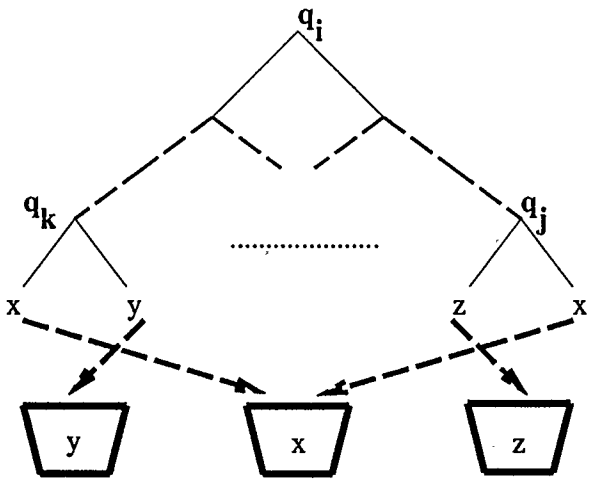

Figure 2.2. The interpretation of a term in a decision theory.

internal node of the tree can be interpreted as a "gate" such as shown in Figure 2.1, labeled with the corresponding decision. However, the graphical representation of Figure 2.1 has been deliberately replaced by the standard graphical representation

of a decision tree. The arcs of the tree correspond to the channels, and its nodes represent occurrences of the variables in the decision term. Although a variable x may appear on several terminal nodes, all occurrences of x are linked to a common bucket.

It must be noted that instead of discussing the classification of isolated events, these definitions deal with the partition of sets of events into disjoint subsets: if an event is deposited in a bucket labeled x, then it cannot also appear in a bucket labeled y. Formally, to classify a single event, it is necessary to ask in which bucket would the event be deposited if the entire set were partitioned with the decision expression. In practice, it is sufficient to trace the application of the decision term to that particular element.

Example 2.2 Consider the problem of determining the appropriate dosage of a medication on the basis of the patient's age group. If the patient is young, the dose should be one tablet; if he is middle aged, it should be two tablets; and if he is old, it should be one and a half tablets. This can be represented in a decision theory $D = (agegroup, \{\})$ with variables $\{1, 2, 1.5\}$, where *agegroup* has domain $\{young, middleaged, old\}$; this domain is an ordered set, each of whose elements corresponds to one of $\{1, \ldots, arity(q)\}$. The appropriate term is $agegroup\langle 1|2|1.5\rangle$, shown in Figure 2.3 If the query *agegroup* returns *young*, then the term is to be

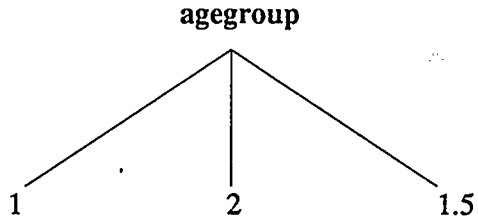

Figure 2.3. Decision tree corresponding to $agegroup\langle 1|2|1.5\rangle$.

taken in its first argument, or subtree, which yields the variable term 1. Formally, an event with the value *young* on test *agegroup*, is placed by the term into a subset that contains the class of individuals for whom the dosage is one tablet.

Throughout this work the following will be used interchangeably: *tree* and *term*; *subtree* and *argument*; *decision* and *test*; and *variable*, *leaf*, and *terminal node.*

The following definition completes the theoretical framework required to describe discrete decision processes.

Definition 2.3 A *decision theory* $D = (Q, E)$ is an algebraic theory with a set of decisions Q and a set of identities E. The terms defined in Q and a stock of variables are called the terms of D (written as $terms(D)$). The identities in E are of the form $t_i = t_j$, where $t_i, t_j \in terms(D)$.

In addition to any identities in E, every decision q in a decision theory must satisfy the following identities:

D.1: Idempotence

$$q\langle x|\ldots|x\rangle = x.$$

D.2: Distribution

$$q_1\langle x_1|\ldots|x_{i-1}|q_2\langle y_1|\ldots|y_m\rangle|x_{i+1}|\ldots|x_n\rangle =$$

$$q_2\langle q_1\langle x_1|\ldots|x_{i-1}|y_1|x_{i+1}|\ldots|x_n\rangle|\ldots|q_1\langle x_1|\ldots|x_{i-1}|y_m|x_{i+1}|\ldots|x_n\rangle\rangle.$$

D.3: Repetition

$$q\langle x_1|\ldots|x_{i-1}|q\langle y_1|\ldots|y_n\rangle|x_{i+1}|\ldots|x_n\rangle = q\langle x_1|\ldots|x_{i-1}|y_i|x_{i+1}|\ldots|x_n\rangle.$$

An additional identity was introduced by Chen and Ras (1985):

D.4: Transposition

$$q_1\langle q_2\langle x_{1,1}|\ldots|x_{1,n}\rangle|\ldots|q_2\langle x_{m,1}|\ldots|x_{m,n}\rangle\rangle$$

$$= q_2\langle q_1\langle x_{1,1}|\ldots|x_{m,1}\rangle|\ldots|q_1\langle x_{1,n}|\ldots|x_{m,n}\rangle\rangle.$$

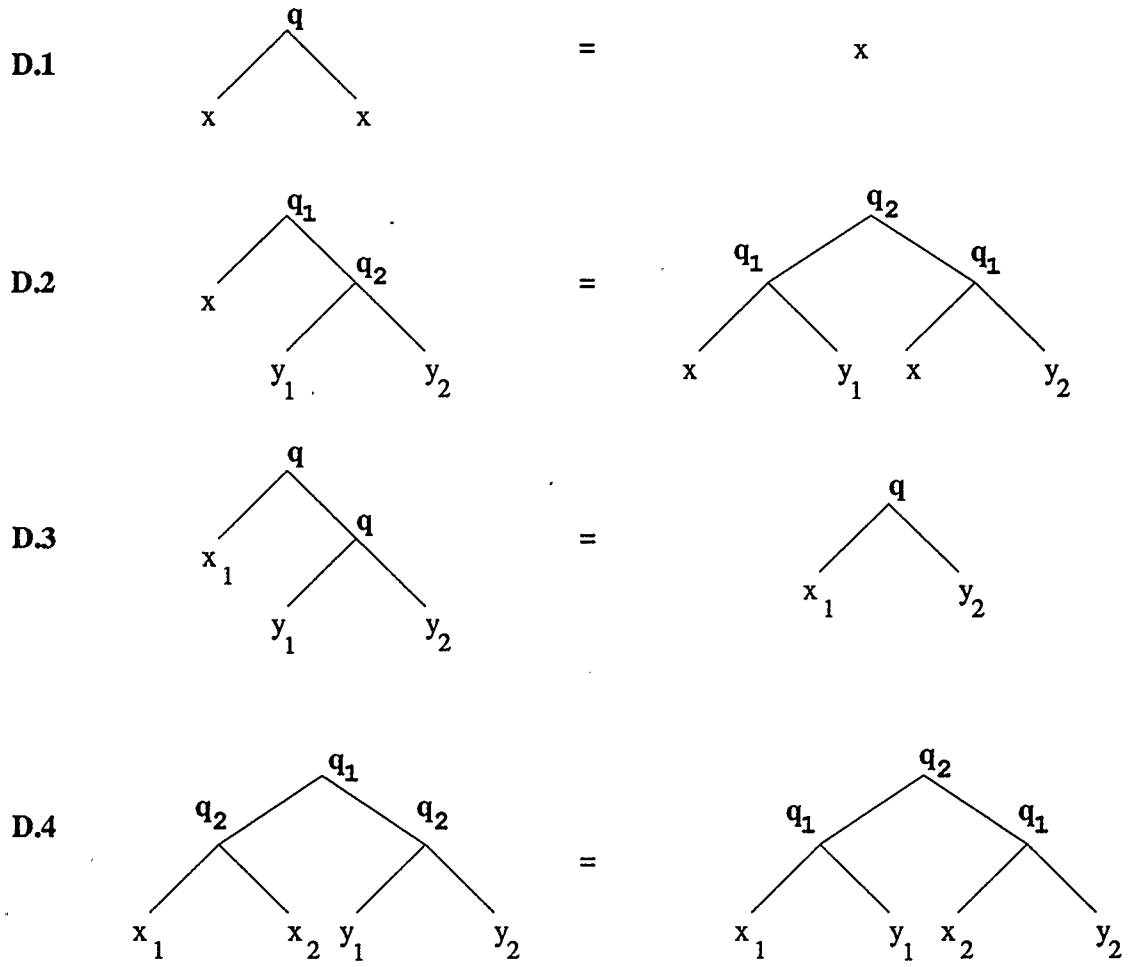

Figure 2.4. Identities **D.1–4** as they apply to binary trees.

These identities, as they apply to binary trees, are illustrated in Figure 2.4. Their semantics correspond closely to intuitions about decision processes. The idempotence identity states that if every possible answer to a query results

in the same conclusion, then the conclusion may be safely drawn without reference to the query. Using the buckets analogy, it is pointless to continue partitioning the set if all of its elements are bound to end up in the same bucket. The repetition identity states that if the same query is posed at two consecutive stages of a decision process, then it may be assumed that the answer is equal both times — and that the second occurrence of the query may be omitted. The buckets corresponding to the variables of the second occurrence of the repeated decision that are removed by the elimination of repetition are going to have no contribution from those occurrences of the variables. The two remaining identities, **D.2** and **D.4**, specify how changes in the ordering of two decisions affect the structure of the process, given that the order in which the decisions are taken must not affect the contents of the buckets.

Cockett (1987b) showed that **D.1–3** constitute a set of axiom schemes for the terms of a decision theory $D(Q, E)$: if every $q \in Q$ satisfies these identities, then so does every term in the decision theory. It was also shown, in the same work, that **D.4** can be derived from **D.2** and **D.3**, and that **D.3** can be obtained from **D.1** and **D.4**. Hence **D.1**, **D.2**, and **D.4** also constitute a set of axiom schemes.

The identities **D.1–4** can be rewritten as a set of equations that express their applicability to composite terms. Some additional notation is required: If W is a composite term with variables (from left to right) $x_1, x_2, \ldots, x_n$, the term will also be denoted by $W\langle x_1|x_2|\ldots|x_n\rangle$.

DW.1: Idempotence for terms

$$W\langle x|\ldots|x\rangle = x.$$

DW.2: Distribution for terms

$$W_1\langle x_1|\ldots|x_{i-1}|W_2\langle y_1|\ldots|y_m\rangle|x_{i+1}|\ldots|x_n\rangle =$$

$$W_2\langle W_1\langle x_1|\ldots|x_{i-1}|y_1|x_{i+1}|\ldots|x_n\rangle|\ldots|W_1\langle x_1|\ldots|x_{i-1}|y_m|x_{i+1}|\ldots|x_n\rangle\rangle.$$

DW.3: Repeat reduction for terms

$$W\langle x_1|\ldots|x_{i-1}|W\langle y_1|\ldots|y_i|\ldots|y_n\rangle|\ldots|x_{i+1}|\ldots|x_n\rangle$$

$$= W\langle x_1|\ldots|x_{i-1}|y_i|\ldots|x_{i+1}|\ldots|x_n\rangle.$$

DW.4: Transposition for terms

$$W_1\langle W_2\langle x_{1,1}|\ldots|x_{1,n}\rangle|\ldots|W_2\langle x_{m,1}|\ldots|x_{m,n}\rangle\rangle$$

$$= W_2\langle W_1\langle x_{1,1}|\ldots|x_{m,1}\rangle|\ldots|W_1\langle x_{1,n}|\ldots|x_{m,n}\rangle\rangle.$$

Giving the equations **DW.1** and **DW.3** a direction in the obvious simplification direction results in rewriting rules which allow the definition of various special forms for decision trees.

Definition 2.4 A term t is *idempotence reduced* if there is no decision that can be eliminated using **DW.1**. Any tree can be transformed into an idempotence reduced form ($id(t)$) by successive applications of **DW.1**.

Definition 2.5 A term t is *repeat reduced* if there is no decision that can be eliminated using **DW.3**. Any tree can be transformed into a repeat reduced form ($rp(t)$) by successive applications of **DW.3**.

Definition 2.6 A term t is *simply reduced* if it is put in repeat reduced form, and the result is then put in idempotence reduced form ($id(rp(t))$).

Equations **DW.1–4** also allow us to define several equivalence relations between terms of a decision theory.

Definition 2.7 Terms t_1 and t_2 in the same decision theory are *structurally equivalent* ($t_1 \equiv t_2$) if one is an identical copy of the other.

Definition 2.8 Terms t_1 and t_2 in the same decision theory are *decision equivalent* ($t_1 \equiv_D t_2$) if one can be obtained from the other by some sequence of applications of **DW.1,3,4** (or **DW.1,2,4**).

Definition 2.9 Terms t_1 and t_2 in the same decision theory are *transpose equivalent* ($t_1 \equiv_T t_2$) if one can be obtained from the other by **DW.4**.

2.2 The reasonable preorder on terms

The equations defined in the previous section can be used to generate a preorder on the terms of a decision theory. This is a reduction preorder in the sense mentioned in Chapter 1.

Definition 2.10 The *reasonable preorder*, $\leq_r$, on the terms of a decision theory is the least preorder [1] that is

Monotonic: If $t_1 \leq_r t_2$, $q\langle \ldots | t_1 | \ldots \rangle \leq_r q\langle \ldots | t_2 | \ldots \rangle$.

Idempotent reducing: $t \leq_r q\langle t | \ldots | t \rangle$.

[1] As mentioned in Chapter 1, a *preorder* is a relation that is reflexive and transitive, but not necessarily antisymmetric. A *least preorder* can be formed by computing the transitive and reflexive closure of a relation.

Repeat reducing:

$$q\langle x_1|\ldots|x_{i-1}|y_i|x_{i+1}|\ldots|x_n\rangle \leq_r$$

$$q\langle x_1|\ldots|x_{i-1}|q\langle y_1|\ldots|y_i|\ldots|y_n\rangle|x_{i+1}|\ldots|x_n\rangle.$$

Transposition invariant:

$$q_1\langle q_2\langle x_{11}|\ldots|x_{1n}\rangle\ldots q_2\langle x_{m1}|\ldots|x_{mn}\rangle\rangle$$

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

If $a \leq b$ and $b \leq a$ are in a preorder, a and b are *preorder equivalent.* Cockett (1987a) showed that two terms of a decision theory are preorder equivalent if and only if they are transpose equivalent. If they are transpose equivalent, then they are preorder equivalent by the definition of $\leq_r$. If they are preorder equivalent, they must be transpose equivalent, because the other inequalities that define $\leq_r$ are asymmetric.

This can be seen as idempotence and repeat reduction shorten the average path length in the term, and only transposition preserves it. If the preorder is interpreted as an ordering on the cost of the terms of a decision theory, then transposition equivalence partitions the terms into cost classes.

The following definition provides an additional characterization of minimal elements in the reasonable preorder.

Definition 2.11 A term t is *irreducible* if every t' such that $t' \equiv_T t$ is simply reduced.

It is not difficult to see that a term is irreducible if and only if it is minimal in the reasonable preorder. If every transpose of t is simply reduced, it means that idempotence and repeat reduction cannot be applied to any t' that is preorder

equivalent to t; since the reasonable preorder is a transitive closure, this implies that there is no term that is strictly smaller than t in the preorder.

However, Definition 2.11 is more useful in a practical sense, as it provides a procedural characterization of minimal elements. This is the characterization used later to formulate an algorithm to find a minimal element in the reasonable preorder.

Example 2.3 The tree of Figure 2.5(i) is irreducible, since it is simply reduced, and so is its only transpose, shown in Figure 2.5(ii). On the other hand, the tree of

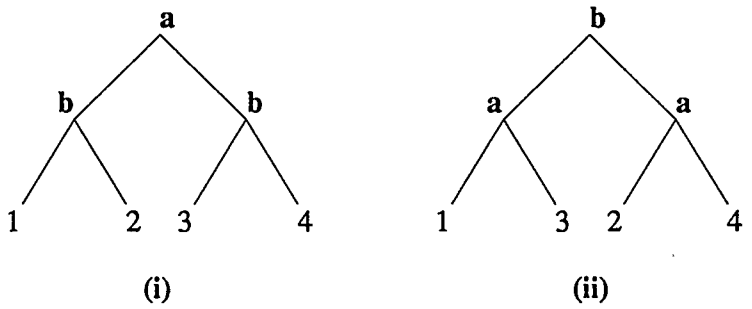

Figure 2.5. **(i)** An irreducible tree, and **(ii)** its only transpose.

Figure 2.6(i) is simply reduced, but not irreducible, since its transpose (Figure 2.6(ii)) is not simply reduced. The transpose can be idempotence reduced to $b\langle 1|2\rangle$.

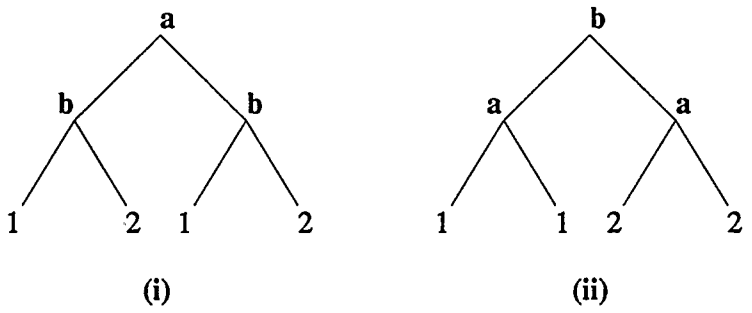

Figure 2.6: **(i)** A simply reduced tree that is not irreducible, and **(ii)** a transpose that is not simply reduced.

The relationship between the minimality and the cost of elements in the reasonable preorder deserves further examination. Taking an arbitrary term and descending from it in the preorder is not guaranteed to yield a minimal element that is optimal with respect to a predetermined criterion. This is so because the preorder is a partial ordering. The following discussion describes the connection between optimization criteria and the reasonable preorder.

Definition 2.12 A *reasonable criterion* is a real cost function of the form

$$cost : terms(D) \mapsto R^+$$

that preserves the reasonable preorder.

To prove that a cost criterion is reasonable, it is sufficient to show that if t_1 is obtained from t_2 by idempotence or repeat reduction, or transposition, or by monotonic composition, then $cost(t_1) \leq cost(t_2)$. Since the reasonable preorder is a transitive closure, this proves that if any $t_1 \leq_r t_2$, then $cost(t_1) \leq cost(t_2)$.

It is therefore easy to construct a proof that, for example, the height of a tree ($height(t)$) or its uniform size [2] ($usize(t)$) are reasonable cost criteria. By the same token, it can be shown that the number of nodes is a reasonable cost criterion for binary terms. This is not true of arbitrary terms; the term

$$q_1\langle q_2\langle x_1|x_2|x_3\rangle|q_2\langle x_4|x_5|x_6\rangle\rangle$$

has a total of nine nodes. Its only transpose,

$$q_2\langle q_1\langle x_1|x_4\rangle|q_1\langle x_2|x_5\rangle|q_1\langle x_3|x_6\rangle\rangle$$

[2]The number of leaves minus one.

has ten. Thus, the transposition of trees, which is treated in the preorder as "neutral" in terms of cost, can in fact disrupt a cost criterion that might at first hand appear to be reasonable.

The proof technique described above can also be used to show, with little effort, that linear combinations of reasonable cost criteria (with positive coefficients) and powers (greater than one) of reasonable cost criteria are also reasonable. Furthermore, there is a proof that any expected testing cost (e.g., an assignment of a positive cost to every decision in a tree, and a branching probability to every arc) is reasonable (Cockett & Herrera, 1990).

Although a minimal element of the reasonable preorder cannot be guaranteed to be optimal with respect to a predetermined reasonable cost criterion, it can be shown that every irreducible term is optimal with respect to some reasonable cost criterion. In particular, every irreducible term is optimal with respect to some expected testing cost (Cockett & Herrera, 1990).

2.3 Tree reduction to an irreducible form

If a repeat reduced term is not irreducible, then there must exist some transpose equivalent term that is not idempotence reduced. Hence the problem of transforming a repeat reduced term into an irreducible form is that of finding transposes that are not simply reduced, performing idempotence reduction, and repeating this process until the resulting term is irreducible.

2.3.1 The strategy for finding irreducibles

The method last discussed for finding irreducibles is impractical, given the large number of transposes a term may have. The worst case for the number of transposes of a

binary tree composed from decisions Q is given by the recurrence $R(|Q|) = |Q| \cdot R(|Q| - 1)^2$, where $R(0) = 1$ (Cockett, 1987b).

To obtain a more efficient method, the problem must be viewed from a different perspective. It follows from Definition 2.11 that if a term t is not irreducible , then there exists a decision in t that can be transposed to the level above the leaves such that idempotence reduction will be applicable. Conversely, to show that the term is irreducible, it is sufficient to show that there is no such decision in t. The following definitions will allow the formalization of this intuition.

Definition 2.13 A decision that occurs on every path from the root to a leaf of a term t is *semi-essential* in t.

While it is clear that any conclusion to be reached using a given tree will probe every decision that occurs semi-essentially, this does not imply that the decision cannot be eliminated from some of the paths by idempotence reduction.

Definition 2.14 A term t is *semi-essentially simple* when its only semi-essential decision is its root decision. It is said to be *factored* by q if the subtree at every occurrence of q is semi-essentially simple.

If a tree t is factored by q, then no occurrence of q in t can be transposed to a lower level. These definitions allow the formulation of the following proposition, from (Cockett, 1987a).

Proposition 2.1 A term t is irreducible if and only if for every decision q in t, t factored by q cannot be idempotence reduced.

To understand why this proposition is true, it is useful to examine how the result could be applied to the problem of finding an irreducible form of a term.

Consider a simpler version of the problem: Let $t = q\langle t_1|\ldots|t_n\rangle$ be such that $t_1,\ldots,t_n$ are known to be irreducible. By the proposition, to test t for irreducibility (or transform it into an irreducible), it is sufficient to factor t by its root decision q and test for idempotence.

In order to factor t, it is necessary to transpose q as far down the tree as possible. The problem becomes that of finding those decisions in t with which q may be legally transposed. The solution is provided by the following result from (Cockett, 1987a).

Lemma 2.2 (*Pulling up lemma*) A decision q is semi-essential in t if and only if there is a term t' with q at the root, such that $t' \equiv_T t$.

The factoring of t by its root decision q can be performed in the following manner. Find some decision $q' \neq q$ that is semi-essential in t. If there is none, then t is already factored by q. Otherwise, build $t' = q'\langle\ldots\rangle$, $t' \equiv_T t$ (called *pulling up* q' in t). The existence of t' is guaranteed by Lemma 2.2. Repeat the process for every subtree in t' that has q at the root.

The correctness of this process — and of Proposition 2.1 — hinges on the assumption that the order of pulling up does not affect the result. In other words, it is important to know that when factoring a term by its root decision q, the choice of semi-essential q' to be pulled up to the root does not in any way affect the presence (or absence) of idempotence when factoring by q is completed. The irrelevance of the order in which semi-essentials are pulled up is shown in the following lemma (Cockett, 1987a).

Lemma 2.3 If $t_1 \equiv_T t_2$ and both terms are factored by q, then

$$t_1 = W\langle q\langle z_{11}|z_{12}|\ldots|z_{1n}\rangle|\ldots|q\langle z_{m1}|z_{m2}|\ldots|z_{mn}\rangle|z_{m+1}|\ldots|z_t\rangle$$

$$t_2 = V\langle q\langle y_{11}|y_{12}|\ldots|y_{1n}\rangle|\ldots|q\langle y_{p1}|y_{p2}|\ldots|y_{pn}\rangle|y_{p+1}|\ldots|y_t\rangle$$

(where W and V are composite expressions) then $m = p$ and $t = s$, and there is a permutation π such that

$$W\langle x_1, x_2, \ldots, x_n\rangle \equiv_T V\langle x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}\rangle,\ z_{ir} \equiv_T y_{\pi(i)r},\ z_j \equiv_T z_{\pi(j)}.$$

Proof As $t_1 \equiv_T t_2$, the decision q_0 at the root of t_1 is semi-essential in t_2. Since t_2 is factored by q, q_0 must always occur before q in V. By Lemma 2.2, it may be moved to the root of t_2 by transposition, yielding t_2'. The same argument can be applied to the subtrees of q_0 in t_2' — which shows that $W \equiv_T V$ given π — and to the arguments of q.

2.3.2 The reduction algorithm

This section presents the algorithm for transforming a repeat reduced term into an irreducible, followed by a complexity analysis, as given by Cockett and Herrera (1990). As hinted in the previous discussion, algorithm is recursive: to reduce a term $t = q\langle t_1|t_2|\ldots|t_n\rangle$, first reduce $t_1, t_2, \ldots, t_n$ to $t_1', t_2', \ldots, t_n'$. Since the t_i's are irreducible, the only way to (possibly) achieve idempotence reduction in $q\langle t_1'|t_2'|\ldots|t_n'\rangle$ is by manipulating q to the level above the leaves.

Algorithm 2.1

`reduce`(t)

if $t = q\langle t_1|t_2|\ldots|t_n\rangle$ **then**

$(t_1', t_2', \ldots, t_n') \leftarrow$ map `reduce` over $(t_1, t_2, \ldots, t_n)$

$t' \leftarrow$ `factor`$(q\langle t_1'|t_2'|\ldots|t_n'\rangle)$

return(`idp-reduce`(t'))

else % t is a leaf %

return(t)

The routine `idp-reduce` (not given here) is one that applies idempotence reduction in the tree wherever possible. When idempotence reduction returns a leaf, this result must be considered for reduction higher up in the tree. The following algorithm is the one that factors a term by its root decision.

Algorithm 2.2

`factor`(t)

if $t = q\langle t_1|t_2|\ldots|t_n\rangle$ **then**

$SE \leftarrow$ `semi-essentials`$(t) - \{q\}$

if $SE = \{\}$ **then return**(t)

else $\%q_0 \in SE\ \%$

$q_0\langle t'_1|\ldots|t'_n\rangle \leftarrow$ `pull-up`(q_0, t)

$(r_1, \ldots, r_n) \leftarrow$ map `factor` over $(t'_1, \ldots, t'_n)$

return$(q_0\langle r_1|\ldots|r_n\rangle)$

else % t is a leaf %

return(t)

As discussed above, the factoring is achieved by finding a decision that is semi-essential in the term (other than the root decision), pulling it up, and repeating the process recursively for the subtrees of the resulting tree.

To find the semi-essentials of a term t, one need only add the root decision of t to the intersection of the semi-essentials of t's arguments.

Algorithm 2.3

`semi-essentials`(t)

if $t = q\langle t_1|t_2|\ldots|t_n\rangle$ **then return**$(\{q\} \cup \bigcap_{i=1}^{n}$ `semi-essentials`$(t_i))$

else return$(\{\})$ % t is a leaf %

Once the semi-essentials have been found, one of them, q_0, must be pulled up to the root of the term. This is done by creating a term $t' = q_0\langle t|\ldots|t\rangle$ and repeat reducing it. Since t is repeat reduced, the only decision with repeated occurrences in t' is q_0. The repeat reduction can be performed with `branch-reduce`(q_0, r, t'), where

the branch reduction leaves t' unchanged, except that every occurrence of q_0 in t' is replaced by q_0's r^{th} argument.

Algorithm 2.4

`pull-up`(q, t)

for $i = 1 \ldots arity(q)$

$t_i \leftarrow$ `branch-reduce`(q, i, t)

return$(q\langle t_1 | \ldots | t_{arity(q)} \rangle)$

Algorithm 2.5

`branch-reduce`(q, r, t)

if $t = q_0\langle t_1 | t_2 | \ldots | t_n \rangle$ **then**

if $q_0 = q$ **then return**(t_r)

else for $i = 1 \ldots n$

$t'_i \leftarrow$ `branch-reduce`(q, r, t_i)

return$(q_0\langle t'_1 | \ldots | t'_n \rangle)$

else return(t) % t is a leaf %

The complexity of reduction is analyzed in detail by Cockett and Herrera (1990). The relevant results are summarized here.

`semi-essentials` The number of semi-essentials in a term is bounded by its minimum height (the length of the shortest path from the root to a leaf). The work at each node consists of intersecting the semi-essentials returned by the subtrees. The cost of intersecting two sorted lists is of the order of the sum of the lists; the work at term $t = q\langle t_1 | \ldots | t_1 \rangle$ is bounded by

$$2 \cdot \sum_{i=1}^{n} minheight(t_i).$$

The sum of this work over the entire term is bounded by $k_1 \cdot usize(t)$, where $usize(t)$ is the uniform size of t.

`pull-up` The cost of pulling up a term is that of performing branch reductions, and this is bounded by $k_2 \cdot usize(t)$, the size of the tree.

`factor` The work required to factor a tree by its root decision is to pull up semi-essentials until the root is at the level above the leaves. This requires applying `pull-up` at every level of the tree; the complexity is $O(usize(t) \cdot height(t))$.

`idp-reduce` The cost of performing recursive idempotence reduction on a tree is that of traversing the entire tree, plus the comparisons required to check leaves for equality. The latter summand is smaller than the number of leaves. Idempotence reduction is $O(usize(t))$.

`reduce` Finally, the complexity of reducing the entire tree is $O(usize(t) \cdot height^2(t))$. This is the cost of of factoring every subtree in the tree.

2.4 Summary

A algebraic formulation of discrete decision theory provides the foundation for defining operations to manipulate decision trees. These operations can are used to define a preorder on decision trees, and are then composed to obtain an efficient algorithm for finding minimal elements in the preorder, or *irreducibles*. Given a repeat reduced input tree, this algorithm returns a tree that is decision equivalent to the input, and guaranteed to be no more costly than the input by the criteria used to define the preorder.

Furthermore, the trees returned by the reduction algorithm are optimal with respect to some reasonable cost criterion. The fact that the optimization criterion cannot be given as a parameter is what distinguishes this algorithm from true optimization.

Chapter 3

Principled induction of decision trees

This chapter shows how the tree reduction algorithm of Chapter 2 can be used to perform principled induction of decision trees.

The first section defines the algorithms required to obtain an initial valid description of a set of training examples, which is then reduced to a tree description that is minimal in the reasonable preorder. The complexity of this algorithm is compared to that of the top-down tree induction methods reviewed in Section 1.2, and the algorithm is shown to be a form of principled induction.

In the second section, a larger reasonable preorder is defined that supports a tree reduction algorithm which outputs more compact decision trees than Algorithm 2.1.

The third section compares the performance of the two reduction algorithms on two simple examples.

3.1 The basic principled induction algorithm

At this stage of the presentation, it is assumed that the training examples from which the tree is induced constitute a complete set; every possible configuration of vectors attainable with the given attributes is present in the set. Furthermore, all of the examples are assumed to be given in full and correct form; each example is defined on all of its features, and without errors. These assumptions will be relaxed in Chapter 4, where the method is extended to enable principled induction from incomplete or noisy training sets.

3.1.1 Generating an initial valid description

The first step in performing principled induction of a decision tree from a set of examples is to generate, at low cost, an initial valid description to be used as a starting point for the descent in the preorder.

Given a set of training examples $e \in S$, $e = (e_1, e_2, \ldots, e_n, e_{n+1})$, described by discrete attributes $Q = \{a_1, a_2, \ldots, a_n\}$, and uniquely classified by a value of $\delta \in \{\delta_1, \delta_2, \ldots, \delta_k\}$, where the elements of δ are variables. The following algorithm is used to build a tree that represents the structure of S.

Algorithm 3.1

`build-tree`(Q)

if $Q = \{\}$ **then return**("leaf")

else

Randomly select some $a_i \in Q$, with arity k

$Q' \leftarrow Q - \{a_i\}$

$t_1, \ldots, t_k \leftarrow$ map `build-tree` over $Q', \ldots, Q'$ % k copies of Q' %

return$(a_i\langle t_1 | \ldots | t_k \rangle)$

Algorithm 3.1 builds a skeletal tree, with its leaf labels undefined, by randomly selecting a test attribute from those that are still available. The implementation of this algorithm used to generate the examples throughout this thesis actually defines an arbitrary order on the attributes, such that `build-tree` produces a tree in which only a given attribute appears at each level, and the attribute appears at only one level. On average, this method of building the tree can be expected to produce much the same final results (after reduction) as a truly random selection of the test attributes.

The values of the variables that must appear at the leaves of `build-tree`(Q) are set by applying Algorithm 3.2 for every example $e \in S$.

Algorithm 3.2

`insert-event`(e, t)

if $t =$"leaf" **then return**(e_{n+1}) % the outcome δ associated with e %

else % $t = a_i\langle t_1|t_2|\ldots|t_{arity(a_i)}\rangle$ %

$p \leftarrow e_i$ % the value of e on attribute a_i %

$t'_p \leftarrow$ `insert-event`(e, t_p)

return$(a_i\langle t_1|\ldots|t_{p-1}|t'_p|t_{p+1}|\ldots|t_{arity(a_i)}\rangle)$

The assumption that S is a complete training set guarantees that all of the leaves will be labeled with a variable, and the assumption that it is noise-free guarantees that every $e \in S$ can be inserted, and that no two events that are equal on $(e_1, e_2, \ldots, e_n)$ may differ on the class indicator e_{n+1}.

Applying Algorithms 3.1 and 3.2 to a set of training examples S defined in discrete-valued features Q results in a decision tree that is a trivial but valid description of the classes in δ.

3.1.2 Reducing the valid description

The initial valid description can be input to the reduction algorithm (Algorithm 2.1) to produce a reduced valid description of the examples; the entire process is a form of principled induction. To show that this is the case, it is sufficient to note the following:

- The identities **D.1** (Idempotence), **D.3** (Repetition), and **D.4** (Transposition), taken as operators on valid descriptions, preserve generality. **D.4** merely

changes the order in which feature values are tested to determine a classification; this does not affect the breadth of applicability. **D.3** prevents repeated tests of the same attribute, so this does not affect the breadth of applicability either. When **D.1** is applied in the reducing direction (the use made in defining the reasonable preorder, and in the reduction algorithm), it functions as a generalization operator. The meaning of the tree $q\langle x|\ldots|x\rangle$ is "all of the events described by attribute q belong to class x, regardless of the value of q." The term can be idempotence reduced to x, which means "all of the events in the space belong to class x." Since the first statement is provable from the first, by the definition of generalization (Definition 1.3), idempotence reduction is a generalization operator.

- These identities constitute, therefore, a set of valid transformations with respect to the reasonable preorder.

- The reduction algorithm uses only these valid transformations to produce a tree that is irreducible, or minimal in the reasonable preorder.

3.1.3 Time and space complexity

As discussed in Chapter 2, the time complexity of reducing a term t is $O(usize(t) \cdot height^2(t))$. Assuming that the training set S is complete, $|S| \leq B^{|Q|}$, where Q is the set of decisions on which the elements of S are defined, and $B = max\{arity(q)|q \in Q\}$. Since the initial tree is only as high as the number of of attributes that describe the examples in S, $height(t) \leq |Q|$. Building the original decision tree is $O(|S| \cdot |Q|)$, hence the complexity of principled induction of decision trees is $O((B^{|Q|} - 1) \cdot |Q|^2)$.

This time complexity is somewhat lower than $O(|S| \cdot |Q|^2)$, that of the top-down method employed by ID3 and C4.5 (see Section 1.2.3). In practice, the recursive reduction process usually decreases $usize(t)$ dramatically by the time t must be factored by its root decision.

The space complexity is the same as that of ID3; in the worst case, ID3 can grow a full tree. However, since the algebraic method *must* grow the full tree before proceeding to prune it, the algorithms presented above could be impractical for inducing from training sets described by a large number of attributes, each with a high arity.

This problem can be avoided with a minor modification. The reader will recall that the reduction process is recursive; the subtrees are reduced, and only then is the tree factored by the root decision, when appropriate. It is therefore possible to build the tree from the bottom up, turning the subtrees into irreducibles before creating the parent tree. The additional cost is in the greater number of traversals of S, but this does not affect the complexity of the reduction process.

3.1.4 The sparseness of the reasonable preorder

The principled induction method described above can produce decision trees that are disconcertingly large. This is because the reasonable preorder is sparse, and the descent through this preorder can often lead to one of the larger minimal elements. Two non-comparable minimal elements may approximate the desired optimization criterion in vastly different degrees, and yet one is "just as irreducible" as the other.

The following example illustrates this phenomenon, and provides the motivation for enlarging the reasonable preorder to allow for fewer of those less satisfactory irreducibles.

a	b	c	δ	a	b	c	δ
1	1	1	3	2	1	1	3
1	1	2	3	2	1	2	2
1	2	1	3	2	2	1	3
1	2	2	1	2	2	2	1

Table 3.1. A complete training set for classification on δ.

Example 3.1 Consider the complete training set of Table 3.1, described by the

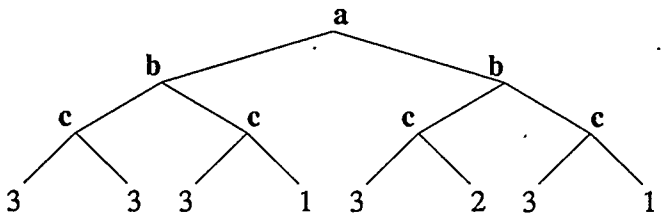

Figure 3.1. An initial tree description of the data in Table 3.1.

binary attributes $\{a, b, c\}$, each belonging to a class from $\delta = \{1, 2, 3\}$. Figure 3.1 shows an initial valid description of the examples. Applying the reduction algorithm yields the tree in Figure 3.2. Now consider the tree shown in Figure 3.3, which is a more compact solution to the induction problem than that of Figure 3.2. Decision a, which is at the root of the tree in Figure 3.2, has been eliminated from all but two of the paths.

It is somewhat disconcerting that the irreducible of Figure 3.2 should be optimal in any sense. In particular, the statement that there exists a reasonable cost criterion on which it is better than the smaller tree of Figure 3.3 provokes disbelief. Yet it is

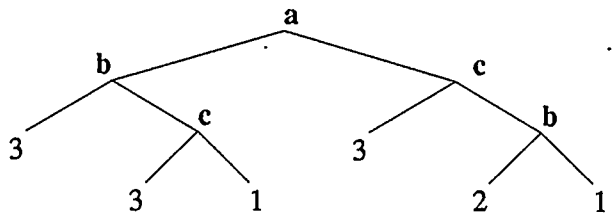

Figure 3.2. Irreducible tree for the data of Table 3.1.

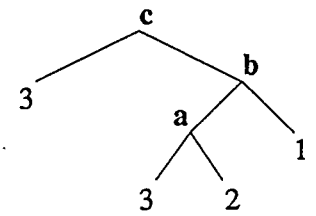

Figure 3.3. A more compact tree for the data of Table 3.1.

quite simple to define an expected testing cost criterion such that the larger tree has a lower cost. Let the testing costs of a, b, and c be 1, 1, and 5, respectively. For the

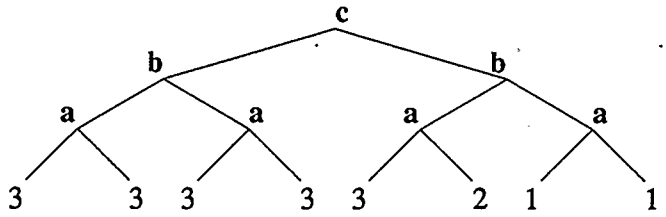

Figure 3.4: An initial valid description for which the reduction algorithm produces the tree of Figure 3.3.

sake of simplicity, assume that all branching probabilities in both trees are $\frac{1}{2}$. The expected testing cost of the first tree is

$$cost(t_1) = cost(a) + \frac{1}{2}(cost(b) + \frac{1}{2}cost(c)) + \frac{1}{2}(cost(c) + \frac{1}{2}cost(b)) = 5.5.$$

For the smaller tree, the cost is

$$cost(t_2) = cost(c) + \frac{1}{2}(cost(b) + \frac{1}{2}cost(a)) = 5.75.$$

It is interesting to note that if the initial valid description had been the tree of Figure 3.4, the reduction algorithm would have produced the irreducible of Figure 3.3. However, the trees of Figures 3.1 and 3.3 are not related in the reasonable preorder. The goal of the next section is to enlarge the reasonable preorder such that initial valid descriptions will be more likely to be related to the smaller minimals in the preorder.

3.2 Enlarging the reasonable preorder

Enlarging the minimal preorder is not in itself a difficult task. However, care must be taken to ensure that it does not become too large. An excessively large preorder could make the cost of any algorithm that traverses it to find a minimal element prohibitive. In particular, as discussed in Section 1.1.2, searching through a total preorder to find a minimal, or optimal element, is an NP-Complete problem.

The strategy used to define the enlarged preorder is to examine the basic tree reduction algorithm for a relatively inexpensive way of restricting the halting condition, such that more valid descriptions will be accessible from any given initial description. That investigation provides the motivation for a new identity on the terms of a decision theory, which is added to the generating conditions of the reasonable preorder to produce an enlarged preorder.

The reduction algorithm is then modified to return minimal elements in the new preorder.

3.2.1 The significance of semi-essentials in tree reduction

The recursive strategy of the reduction algorithm is as follows: to reduce a term $t = q\langle t_1|t_2|\ldots|t_n\rangle$, first reduce $(t_1, t_2, \ldots, t_n)$ to $(t'_1, t'_2, \ldots, t'_n)$, then attempt to fac-

tor $t' = q\langle t'_1|t'_2|\ldots|t'_n\rangle$ by q and search for possible idempotence reductions. The rationale is that since $(t'_1, t'_2, \ldots, t'_n)$ are irreducible, only factoring by q can yield further idempotence reductions in t'.

The precondition for factoring by q is that there be some decision $q_0 \neq q$ that is semi-essential in t'. The role of q_0 is to replace q at the root of the tree. If no such decision can be found, the tree is returned as an irreducible. The tree of Figure 3.2 was returned as an irreducible because when the time came to factor it by its root decision a, no semi-essentials could be found below that could replace a at the root. It is relevant to investigate the properties of semi-essential decisions that enable the basic reduction algorithm to output irreducible trees, and to inquire whether there exist other decisions in the tree that possess those properties.

Algorithm 2.2 (`factor`) factors a tree $t = q\langle\ldots\rangle$ by its root decision by finding a q_0 that is semi-essential in t and creating a tree $t' = q_0\langle t|\ldots|t\rangle$. It then proceeds to repeat reduce t'; since the arguments of the tree are already repeat reduced, the only decision that may be repeated in t' is q_0. Hence the repeat reduction can be accomplished by applying `branch-reduce` to the arguments of the root decision, each of which is t. This procedure leaves t unchanged, except for the subterms with q_0 at the root; these are replaced by their r^{th} argument. The factoring is completed by applying `factor` recursively to the arguments of t'.

The following example illustrates how pulling up a non semi-essential decision in an irreducible term with no semi-essentials other than its root decision leads to the loss of irreducibility.

Example 3.2 Consider the decision tree of Figure 3.5, with both of a's arguments irreducible, and no semi-essentials other than a. This tree is already factored by its root decision, and therefore irreducible. Pulling up decision d yields the tree of

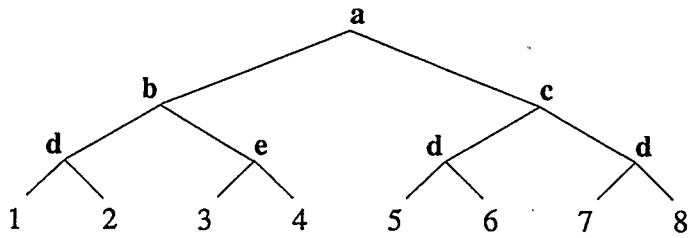

Figure 3.5. A tree that is factored by its root decision.

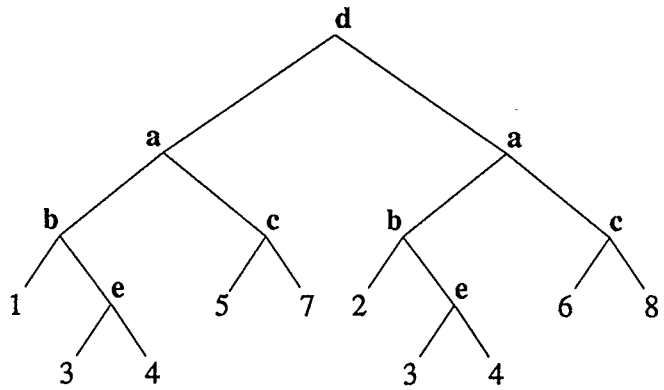

Figure 3.6. The tree of Figure 3.5 with decision d pulled up.

Figure 3.6. Note that the structure $a\langle b|c\rangle$ appears in both subtrees. It may be substituted by a composite structure W, which results in the tree of Figure 3.7. Figure 3.8 shows the result of transposing d and W, using equation **DW.4** (transposition for terms). It is easy to verify that the last tree is not irreducible: transposing d and e would make it possible to eliminate d by idempotence reduction.

It is not really necessary to perform the last transposition to find idempotence. The fact that the tree is not irreducible may be inferred from the structural equivalence of its arguments.

Lemma 3.1 If a decision tree is of the form $t = q\langle t_0|\ldots|t_0\rangle$, that is, all i arguments of the root decision are structurally equivalent, then t is not irreducible. In particular, factoring t by its root decision q causes idempotence at the leaves; the root decision

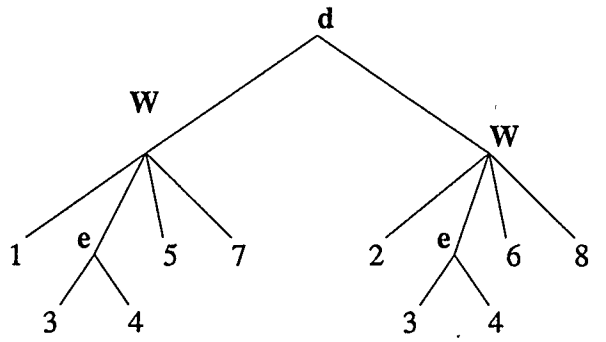

Figure 3.7. The tree of Figure 3.6, represented using composite term W.

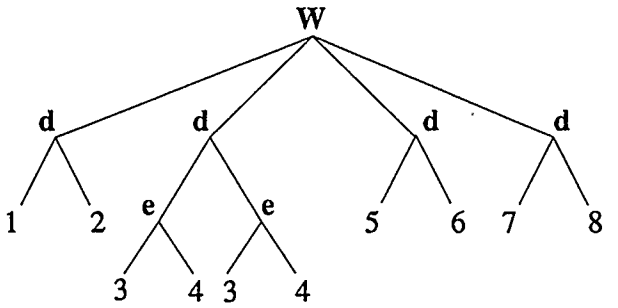

Figure 3.8. The tree of Figure 3.7 after transposition of d and W.

is redundant and can be eliminated from the tree altogether.

Proof By induction on the height of the tree.

What Example 3.2 shows is that semi-essential decisions are important in the reduction algorithm because pulling up a decision that is *not* semi-essential in a term — prompted by the fact that the term has no semi-essentials other than its root decision — leads to the loss of irreducibility. This is formalized in the following lemma.

Lemma 3.2 Let $t = q\langle t_1|\ldots|t_n\rangle$ be a repeat reduced term such that every t_i is irreducible, and the only semi-essential decision in t is q. If a non-semiessential decision q_0 that occurs in t is pulled up to the root of t, the resulting t' is not

irreducible. In particular, factoring t' by q_0 makes some occurrence of q_0 idempotent.

Proof Select an arbitrary path P in t, and let $r = q_i\langle r_1|r_2|\ldots|r_n\rangle$ be the lowest subtree along P such that q_0 appears in some but not all of the n arguments — such a path exists, because otherwise q_0 would be semi-essential in t. Let $t_1 = q_0\langle t|\ldots|t\rangle$, the result of pulling up q_0 in t. Subtree r occurs at the same height in every argument of q_0 in t_1. Let $t_2 = rpt(t_1)$, the result of repeat reducing t_1; let r_i be some subtree of r in which q_0 does not occur. The repeat reduction leaves r_i undisturbed in every copy of t. This r_i appears in identical form, and in the same position, in t_2. Note that the structure of t_2's subtrees is identical from their root (q) down to q_i. This structure can be replaced by a composite structure W; assuming that $arity(q_0) = k$, this results in the term

$$q_0\langle W\langle s_{1,1}|\ldots|s_{1,i-1}|r_i|s_{1,i+1}|\ldots|s_{1,m}\rangle|\ldots|W\langle s_{k,1}|\ldots|s_{k,i-1}|r_i|s_{k,i+1}|\ldots|s_{k,m}\rangle\rangle$$

in which the i^{th} argument of every occurrence of W is r_i. Transposing q_0 and W results in a term with W at the root; its i^{th} subtree is of the form $q_0\langle r_i|\ldots|r_i\rangle$. By Lemma 3.1, q_0 can be eliminated from this subtree. This implies that t_2 is not irreducible, as factoring by q_0 produces idempotence.

The result is discouraging in principle, since it appears to rule out non semi-essential decisions as potential replacement roots that will allow the factoring of a tree by its root decision. However, by isolating the property of semi-essentials that makes them suitable candidates for being pulled up to the root, it in fact it provides a valuable clue for finding decisions that are not semi-essentials, yet might be pulled up without leading to a loss of irreducibility.

3.2.2 Whiskers

The question that arises is whether there exist decisions that are not semi-essential, yet pulling them up in a tree and refactoring does not produce idempotence. To show that such decisions do indeed exist, it is necessary to isolate a special type of decision tree, the *whisker*.

Definition 3.1 A *whisker* is a term of the form $w = q\langle t_1|t_2|\ldots|t_n\rangle$, where each argument of t is either $t_i = x$ or of the form $t_i = q_1\langle r_{i1}|r_{i2}|\ldots|r_{im}\rangle$, and there exists some j, $1 \leq j \leq m$, such that for every $t_i \neq x$, $r_{ij} = x$. A whisker has three important components, as follows: q is called the *whisker root*, q_1 is called the *whisker decision*, and x is called the *whisker variable*.

Example 3.3 The term of Figure 3.9 is a whisker with $j = 2$, root a, decision b, and variable x. The subtrees y and z may or may not be variables. If they are, the whisker is called *shallow*.

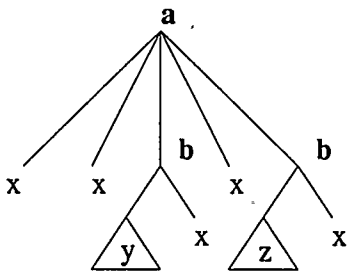

Figure 3.9. A sample whisker.

The following is an important operation on whiskers.

Definition 3.2 A *whisker rearrangement* (wr) is the operation composed of

1. distributing the root of a whisker past its variables, and
2. applying idempotence reduction.

The result of a whisker rearrangement is also a whisker.

Example 3.4 To rearrange the whisker of Figure 3.9, distribute b over a (Figure 3.10(i)) and apply idempotence reduction (Figure 3.10(ii)). In general, if w is a whisker, then $wr(wr(w)) = w$.

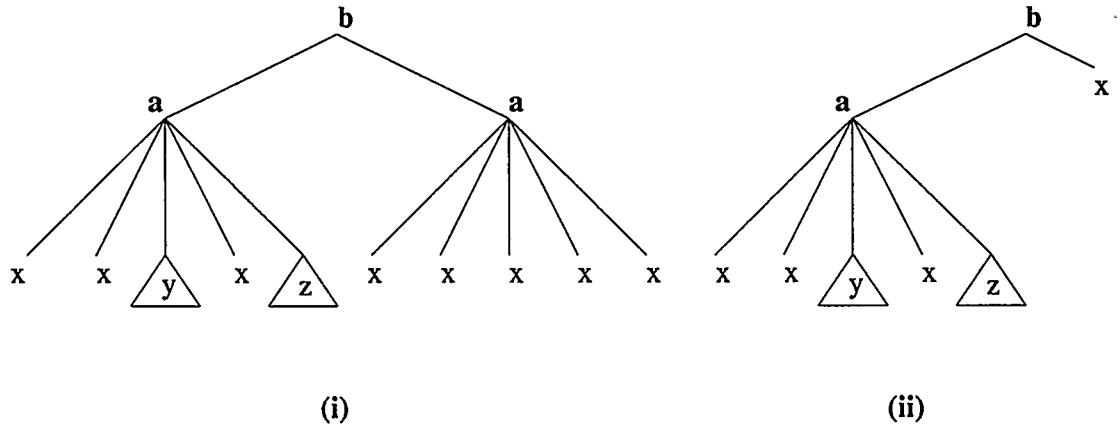

Figure 3.10: The whisker of Figure 3.9 after **(i)** distribution, and **(ii)** idempotence reduction.

Weak transposition

It is now possible to define a new identity for terms of a decision theory.

D.5: Weak transposition for whiskers

$$w = wr(w).$$

The validity of weak transposition follows from the definition of a whisker rearrangement in terms of distribution and idempotence reduction. Throughout the rest of this discussion, *weak transposition* and *whisker rearrangement* will be used interchangeably.

The effect of weak transposition

The effect of rearranging a whisker is that the whisker decision and whisker root exchange positions in the tree, making the former whisker decision semi-essential.

A shallow whisker is factored by its root, and *delayed* by its decision. The formal definition of a tree that is delayed by a decision was given by Cockett (1987a).

Definition 3.3 A term t is *decision delayed* by q if t is simply reduced and at each leaf either the last decision is q or q does not occur at all on the path to that leaf.

Another interpretation of weak transposition is that it transforms a shallow whisker w into another shallow whisker w' that is factored by the whisker decision of w and delayed by the whisker root of w.

The distinction between factoring and delaying a term by a decision is significant. Factoring a tree t by a decision q only allows transposing q with other decisions in t. In order to delay t by q, it is necessary to apply whisker rearrangement; note that distribution alone will not do, since a delayed tree must be simply reduced. Delaying t by q implies that q will be pushed as far down the tree as possible by transposition and whisker rearrangement.

It is important to point out that whether a tree is factored or delayed by q does not affect the possibilities of idempotently eliminating q. A tree can be delayed by q only if factoring it by q would not produce idempotence.

Weak transposition can be used jointly with the transposition identity to define an equivalence relation for terms of a decision theory.

Definition 3.4 The terms t_1 and t_2 are *whisker equivalent* ($t_1 \equiv_w t_2$) if one can be obtained from the other by some sequence of transpositions and whisker rearrangements.

3.2.3 The weak reasonable preorder

Weak transposition can be used to enlarge the reasonable preorder.

Definition 3.5 The *weak reasonable preorder* $\leq_{wr}$ on the terms of a decision theory is defined like the reasonable preorder of Definition 2.10, with the additional stipulation that it be weak transposition invariant:

$$wr(w) \leq_{wr} w$$

The preorder $\leq_{wr}$ is a proper superset of the reasonable preorder; it includes pairs of terms that are weak transposes, which are not in the reasonable preorder. Since the weak reasonable preorder is also a least preorder (e.g., a transitive and reflexive closure), it also includes pairs of terms that are whisker equivalent.

While it is easy to show that $\leq_{wr}$ preserves the height of the tree, just as $\leq_r$ does, the weak reasonable preorder preserves the uniform size of binary trees only.

Example 3.5 Consider the shallow whisker

$$q_1\langle x_1|q_2\langle x_1|x_2|x_3\rangle\rangle$$

with $arity(q_1) = 2$ and $arity(q_2) = 3$. Applying weak transposition produces

$$q_2\langle x_1|q_1\langle x_1|x_2\rangle|q_1\langle x_1|x_3\rangle\rangle.$$

The weak transposition increases the uniform size from 3 to 4, because a decision was distributed in front of another with lower arity.

This is somewhat discouraging, because it implies that descending in the weak reasonable preorder might actually increase the uniform size of the original term. However, this increase is bound not to be all that significant, for two reasons. If either x_2 or x_3 in the last example were equal to x_1, the weak transposition would not change the uniform size of the tree. The second reason is that arbitrary whisker rearrangements in a term can be expected to distribute a decision in front of another with higher arity, as often as in front of one with lower arity.

The following is a procedural characterization of the minimal elements in the weak reasonable preorder; it is analogous to the definition of irreducible trees (Definition 2.11).

Definition 3.6 A term t is said to be *whisker reduced* if every term that is whisker equivalent to it is simply reduced.

From this definition it follows that every whisker reduced tree is also irreducible; the converse is not true.

3.2.4 The strategy for finding whisker reduced terms

It is possible now to return to the task of characterizing the decisions that can be pulled up to the root of a tree, such that refactoring by them does not produce idempotence.

Definition 3.7 A decision q that occurs in term t is a *suitable root* for t if it can be made semi-essential in t by any sequence of whisker rearrangements.

It follows from this definition that any decision which is semi-essential in a term is also a suitable root for it.

The following lemma shows that suitable roots may be pulled up to the root of a tree that needs to be factored by its root decision without any deleterious effects. In order to ensure that this is the case, the pulling up is followed immediately by idempotence reduction of the entire tree.

Lemma 3.3 Let $t = q_0\langle t_1|\ldots|t_n\rangle$, with every t_i irreducible. If q is a suitable root for t, $q \neq q_0$, pulling up q to the root of t, idempotence reducing the resulting tree, and refactoring it by q will *not* produce any idempotence at the leaves.

Proof If q is semi-essential in t, the result is given by the fact that $t_1, \ldots, t_n$ are irreducible, and factoring by q would just give transposes of the t_is, which are also irreducible.

If q is not semi-essential in t, consider the lowest subtree $r = q_1\langle r_1|\ldots|r_n\rangle$ in t such that q appears in some but not every r_i. Since q is not semi-essential, at least one such subtree must exist in t. Recall that pulling up q to the root of t initially results in a term $t' = q\langle t|\ldots|t\rangle$, where the i^{th} copy of t is branch reduced by q and i. Refactoring by q will produce idempotence if in all copies of t, the r_is on which q does not appear are left intact by the idempotence reduction (see Lemma 3.2).

Since q is a suitable root for t, by definition it is possible to whisker rearrange q to the root of r. For this to be true, q must appear as the whisker decision on one of a chain of whiskers along the non-terminal r_is, of which r itself is highest in t. All of these whiskers must have the same whisker variable. In fact, every r_i in which q does not occur must be a terminal with the whisker variable as its label. When `pull-up`(q, t) is idempotence reduced, in the i^{th} copy of t such that i corresponds to the argument number of the whisker variable with q at its root, the entire chain of

whiskers will collapse. In particular, the idempotence reduction will also eliminate the t_is in which q did not occur.

This result can be used to prove the whisker-equivalence analog for the pulling up lemma (Lemma 2.2).

Lemma 3.4 Decision q is a suitable root for t if and only if there exists a term t' with q as its root decision, such that $t' \equiv_W t$.

Proof If q is a suitable root for t, let t_1 be t with q made semi-essential by whisker rearrangement. By Lemma 2.2, there exists a t' with q at the root that is transpose equivalent to t_1. It follows that t' is also whisker equivalent to t. Now let t be whisker equivalent to some t' with q as its root decision, and assume that q is not a suitable root for t. Then q is not semi-essential in t, since being semi-essential implies being a suitable root. By Lemma 2.2, in order to obtain t from t', q must at some point become semi-essential. Since transposition cannot affect semi-essentiality, and the only other available operation is whisker rearrangement, it must be possible to make q semi-essential in t by whisker rearrangement only. By the definition of a suitable root, this contradicts the assumption that q is not a suitable root for t.

Before formulating the algorithm for finding whisker reduced trees, it remains to prove a result analogous to Proposition 2.1

Proposition 3.5 A term t is whisker reduced if and only if for every decision q in t, delaying t by q cannot be idempotence reduced.

It has already been shown that pulling up a decision q that is a suitable root for $t = q_0\langle \ldots \rangle$, and idempotence reducing, preserves the irreducibility of the tree up to delaying by q_0 (recall that pulling up suitable roots, as opposed to semi-essentials, causes the tree to be delayed, as opposed to just factored, by q_0).

Once again, the validity of the proposition hinges on the irrelevance of the order in which decisions are pulled up.

Lemma 3.6 If $t_1 \equiv_w t_2$ and in both terms q has been pushed as down as close to the leaves as possible using transposition and whisker rearrangement, then both terms are delayed by q, then

$$t_1 = W\langle q\langle z_{11}|z_{12}|\ldots|z_{1n}\rangle|\ldots|q\langle z_{m1}|z_{m2}|\ldots|z_{mn}\rangle|z_{m+1}|\ldots|z_t\rangle$$

$$t_2 = V\langle q\langle y_{11}|y_{12}|\ldots|y_{1n}\rangle|\ldots|q\langle y_{p1}|y_{p2}|\ldots|y_{pn}\rangle|y_{p+1}|\ldots|y_t\rangle$$

(where W and V are composite expressions) then $m = p$ and $t = s$, and there is a permutation π such that

$$W\langle x_1, x_2, \ldots, x_n\rangle \equiv_w V\langle x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}\rangle,\ z_{ir} \equiv_w y_{\pi(i)r},\ z_j \equiv_w z_{\pi(j)}.$$

Proof As $t_1 \equiv_w t_2$, the decision q_0 at the root of t_1 is a suitable root for t_2 (Lemma 3.4). Since t_2 is delayed by q, q_0 must always occur before q in V. By Lemma 3.4, it may be moved to the root of t_2 by transposition and whisker rearrangement, yielding t_2'. The same argument can be applied recursively to the arguments of q_0 in t_2'. This restructuring determines the permutation π, and shows that $W \equiv_w V$. The same process can be applied to the arguments of q.

This result proves Proposition 2.1.

3.2.5 Whisker reduction

Only minor modifications are required to convert the algorithm for finding an irreducible into one for finding a whisker reduced term. The definition of these changes is followed by an analysis of their impact on the time complexity of the original algorithm.

The whisker reduction algorithm

First the pulling up algorithm is modified to accommodate non semi-essential suitable roots. Recall that it is necessary to idempotence reduce the tree following the pull-up.

Algorithm 3.3 `Whisker-branch-reduce` is obtained by composing `idp-reduce` and `branch-reduce`.

The most significant change is in the selection of the candidate decisions for pulling up. For the sake of efficiency it is convenient to separate semi-essential and non semi-essential decisions, and to express the algorithm as a query: it is known that q is not semi-essential in t; is it a suitable root for t?

Algorithm 3.4

`suitable-root?`(q,t)

  return$(\exists i \in \{1,\ldots,arity(q)\}|$ `suitable-root-branch?`$(q,t,i))$

`suitable-root-branch?`(q,t,r)

  if t is variable **return**$(false)$

  else if $q = q_0$ $\qquad\qquad\qquad\qquad$ $\%t = q_0\langle t_1|\ldots|t_n\rangle$ $\%$

  else

    if $\forall i \in \{1,\ldots,n\}, t_i$ is non-terminal **then**

      return$(\bigwedge_{i=1}^{n}$`suitable-root-branch?`$(q,t_i,r))$

    else

      return (`reduce(whisker-branch-reduce`$(q,t_1,r) = \ldots$

        $\ldots =$ `whisker-branch-reduce`$(q,t_n,r) = x))$

Recall that in order for a non semiessential decision to be a suitable root for t, pulling up q to the root of t must result in a tree from which q cannot be eliminated by delaying and idempotence reduction. This requires that in one of the copies of t

appearing as arguments of q, some path on which q does not appear be pruned at the bottom, so that refactoring by q will not yield a subterm with q at the root and all arguments structurally equivalent.

The query posed by `suitable-root-branch?`(q, t, r) is whether whisker branch reducing t by q on its r^{th} argument will cause the elimination of at least one of the subterms that would otherwise appear as structurally equivalent arguments of q (see proof of Lemma 3.3).

Let us examine the non trivial cases of this algorithm for correctness. If the term is of the form $t = q_0\langle t_1| \ldots |t_n\rangle$, with every t_i non terminal, the algorithm returns the conjunction of the query for the arguments: if q can be made semi-essential in each of the subtrees, then it can be made semi-essential in the parent tree. If some of the arguments are terminal, then in order for q to be a suitable root with branch r, all of the leaf children of t must be equal to, say x, and whisker branch reducing every non terminal argument with r must yield x. This is the condition for the terminal children of t to be "cleaned up" by idempotence reduction.

The following algorithm returns the semi-essentials of a t, if any, or a singleton set containing one suitable root of t. If none can be found, the algorithm returns the empty set.

Algorithm 3.5

```
suitable-roots(t)
    SE ← semi-essentials(t)  %t = q⟨...⟩%
    if SE ≠ {q} then return(SE − {q})
    else
        D ← decisions(t) − {q}
        for each q ∈ D
            if suitable-root?(q, t) then
                return({q})
        return({})
```

The following two algorithms complete the procedural definition of whisker reduction.

Algorithm 3.6 `Whisker-delay` is obtained by substituting the algorithms `whisker-branch-reduce` for `branch-reduce` and `suitable-roots` for `semi-essentials` in `factor` (Algorithm 2.2).

Algorithm 3.7 `Whisker-reduce` is obtained by substituting `whisker-delay` for `factor` in `reduce` (Algorithm 2.1).

The complexity of whisker reduction

The following proposition analyzes the time complexity of whisker reduction.

Proposition 3.7 The time complexity of `whisker-reduce` applied to full normal trees is $O(|B| \cdot (|B|^{|Q|} - 1)| \cdot |Q|^3)$, where B is the maximum arity of any $q \in Q$.

Proof The only truly different algorithms are `whisker-branch-reduce` and `suitable-roots`.

- The additional work done by `whisker-branch-reduce` over `branch-reduce` is checking for idempotence. Both the branch reduction and the checks for idempotence are bounded by the size of the tree, so this algorithm does not change the complexity.
- In the basic reduction algorithm, the work is dominated by the intersections required to compute the semi-essentials of a tree. The cost of finding a suitable root is that of finding the semi-essentials, plus checking whether each decision is a suitable root. The cost of `suitable-root-branch?` is bounded by $k \cdot usize(t)$; applying this for every branch of each decision is $O(|B| \cdot height(t) \cdot usize(t))$. This bound replaces $O(usize(t))$ in the complexity analysis of Chapter 2, yielding $O(|B| \cdot usize(t) \cdot height^3(t))$. Substituting $|Q|$ for $height(t)$ and $(|B|^{|Q|} - 1)$ for $usize(t)$ gives the desired result.

The order of whisker reduction is larger than that of standard reduction by a factor of $|B| \cdot |Q|$. The form of the algorithm guarantees that this additional cost will not be incurred unless no semi-essentials can be found. The effectiveness of this modification is evident in the small examples presented in the following section.

3.3 Some empirical results

This section presents three sample applications of `whisker-reduce`, all on complete training sets. First, Example 3.1 is recomputed using `whisker-reduce` instead of the basic reduction algorithm.

Example 3.6 When `whisker-reduce` is applied to the tree of Figure 3.1, the root's right subtree is reduced exactly as before, to $c\langle 3|b\langle 2|1\rangle\rangle$ (Figure 3.2). The left subtree is first reduced to $b\langle 3|c\langle 3|1\rangle\rangle$. The basic reduction algorithm can do no more with this

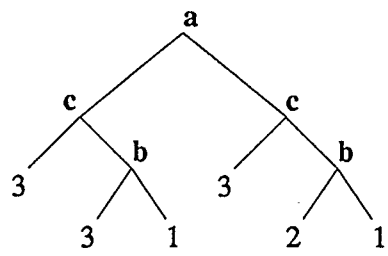

Figure 3.11: The term of Figure 3.1 after its arguments have been whisker reduced.

subtree, since it has no semi-essentials other than the root b. But `whisker-reduce` attempts to delay by b, finds suitable root c, and pulls it up, resulting in the tree of Figure 3.11. This tree has suitable root c (which is also a semi-essential); pulling up c and factoring by a, followed by idempotence reduction at the leaves, yields $c\langle 3|\langle b\ \langle a\ \langle 3|2\rangle\rangle|1\rangle\rangle$, the term of Figure 3.3.

#	a	b	c	d	δ	#	a	b	c	d	δ	#	a	b	c	d	δ
1	1	1	1	1	3	9	2	1	1	1	3	17	3	1	1	1	3
2	1	1	1	2	2	10	2	1	1	2	2	18	3	1	1	2	3
3	1	1	2	1	3	11	2	1	2	1	3	19	3	1	2	1	3
4	1	1	2	2	1	12	2	1	2	2	1	20	3	1	2	2	1
5	1	2	1	1	3	13	2	2	1	1	3	21	3	2	1	1	3
6	1	2	1	2	2	14	2	2	1	2	2	22	3	2	1	2	2
7	1	2	2	1	3	15	2	2	2	1	3	23	3	2	2	1	3
8	1	2	2	2	1	16	2	2	2	2	3	24	3	2	2	2	3

a:	1=young	2=pre-presbyopic	3=presbyopic
b:	1=myope	2=hypermetrope	
c:	1=not astigmatic	2=astigmatic	
d:	1=reduced tear production	2=normal tear production	
δ:	1=fit hard contact lenses	2=soft contact lenses	3=no contact lenses

Table 3.2. Data set on the prescription of contact lenses.

Example 3.7 Table 3.2 contains a data set for the prescription of contact lenses, taken from (Cendrowska, 1987). The columns labeled **a**, **b**, **c**, and **d** contain the

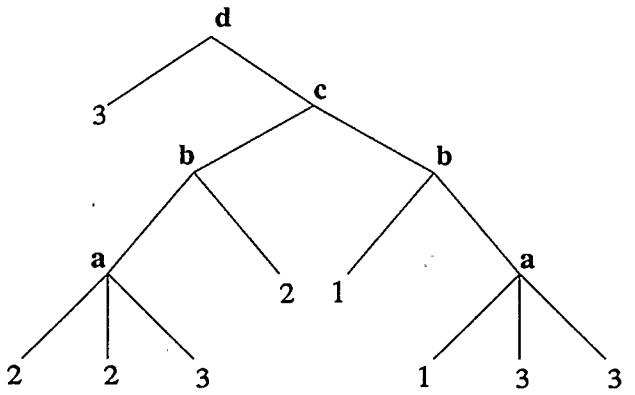

Figure 3.12: Tree obtained from the data of Table 3.2 by both ID3 and whisker reduction of an initial description.

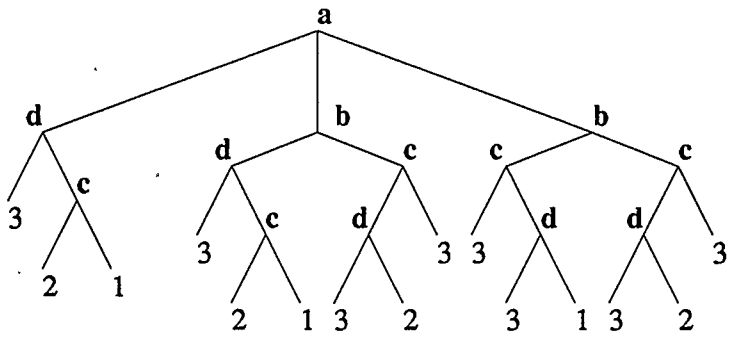

Figure 3.13. Tree for the contact lens data, using the basic reduction algorithm.

values of attributes, and the columns labeled δ contain the recommendations in the cases described by the attributes. Figure 3.12 shows the decision tree induced from the data by ID3 as reported by Cendrowska (1987). Building an initial tree with 24 leaves and applying `whisker-reduce` results in a tree that is structurally equivalent to that of Figure 3.12. The basic reduction algorithm returns a much larger tree, shown in Figure 3.13.

3.4 Summary

This chapter described the application of the tree reduction method of Chapter 2 to the principled induction of decision trees from complete sets of examples given as vectors of feature values. The analysis of tree reduction's time complexity was reexamined under the light of this application to show that it is somewhat better than that of the statistical method used in ID3 and C4.5.

It was shown that the sparseness of the reasonable preorder can result in relatively large minimal elements. The weak reasonable preorder, a superset of of the reasonable preorder, was defined to enable greater reduction in the size of the trees.

The complexity of the algorithm that produces whisker reduced trees — trees that are minimal in the weak reasonable preorder — was shown to be greater than that of the algorithm for irreducibles by a factor that is linear in the number of features and the maximum number of categories of any of those features. The form of the algorithm is such that the additional cost is incurred only when the less costly technique of the original reduction algorithm has proven fruitless.

Empirical results obtained from small sets of training examples were presented to illustrate the improvement obtained in the results by exploring the larger preorder. The examples highlight the tradeoff between the size of the preorder — the cost of finding minimal elements in that preorder — and the degree of simplification achieved. This tradeoff serves to validate the claim that principled induction is a clear representation of the problem of learning concept descriptions from examples.

Chapter 4

Induction from imperfect data

Induction tasks are rarely performed on perfect data such as were used in the previous chapter to demonstrate the principled induction of decision trees.

One frequent form of imperfection is the *incompleteness* of training data. In domains defined by sets of discrete features this translates into training examples that do not cover the entire space defined by the attributes.

Another type of imperfection is noise, which can be divided into *inconsistencies* and *undefined attribute values*. Inconsistencies appear in the form of examples with equal values on all attributes but different class descriptions, or, in the case of data sets that are also incomplete, in the form of full descriptions of examples which have not in fact been observed.

This chapter extends the methods of Chapter 3 to enable principled induction from imperfect data. The extended methods are shown empirically to produce results that are satisfactory in terms of their compactness and predictive accuracy. These results compare well with those obtained using the tree induction system C4.5 (Quinlan, 1990b), and validate the claim that the principled approach to inducing decision trees is an effective one. However, a new dimension is defined to rate decision trees induced from noisy data. It is shown that trees built top-down with statistical techniques are are better suited for later pruning using statistical criteria.

4.1 Incomplete training sets

The reduction of decision trees constructed from incomplete training data is complicated by the occurrence of leaves that lack a class label. Such a situation is illustrated in the following example.

Example 4.1 Consider the training examples

$$\{(a = 1, b = 1, class = t); (a = 2, b = 1, class = t); (a = 2, b = 2, class = f)\},$$

where tests a and b have arity two and the possible classes are t and f. These events can be represented by the decision tree of Figure 4.1. The second leaf from the left

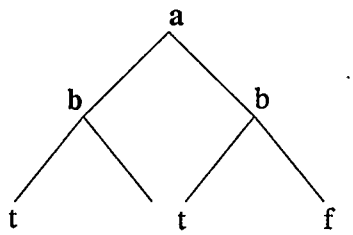

Figure 4.1. Decision tree representing the data from Example 4.1.

does not have a class label, since the tree was constructed from three distinct training events from a space of size four. This tree is irreducible, since it is idempotence reduced (the idempotence operator cannot be applied anywhere in the tree), and the same is true of its only transpose, obtained by transposing decisions a and b. Yet the simpler term $b\langle t|f\rangle$ is also a valid description of the examples.

The goal of this section is to introduce a new operator that further enlarges the weak reasonable preorder to allow the simplification of trees with unlabeled leaves.

The formal treatment of decision trees with unlabeled leaves is based on the theoretical framework developed by Herrera (1988).

4.1.1 Functional identities

The discussion in previous chapters dealt exclusively with decision theories of the form $D = (Q, \{\})$, that is, without any identities beyond those preserved by all terms of any decision theory.

Decision theories may contain additional identities of the form $t_1 = t_2$. A special type of identity, called *functional identity*, is useful in characterizing trees formed from incomplete training data.

Definition 4.1 Let a term t in a decision theory $D = (Q, E)$, and a decision q that does not occur in t; let $arity(q) = n$, and let the distinct variables $\{x_1, \ldots, x_n\}$ be the possible outcomes of q. The identity

$$t = q\langle x_1 | \ldots | x_n \rangle$$

is called a *function on q*.

Unlabeled leaves in a decision tree such as that of Figure 4.1 are called *unreachable leaves*. It is convenient to label every such leaf in a tree with a unique variable. Then a decision problem defined by a set of examples with class attribute $\delta \in \{\delta_1, \ldots, \delta_n\}$ can be expressed by the functional identity

$$t = \delta\langle \delta_1 | \ldots | \delta_n \rangle,$$

where t is the decision tree formed to describe the examples.

Example 4.2 Labeling the unreachable leaf of the tree in Figure 4.1 with the unique variable x yields the term

$$a\langle b\langle t|x\rangle | b\langle t|f\rangle \rangle.$$

The decision problem can then be expressed by the functional identity

$$a\langle b\langle t|x\rangle|b\langle t|f\rangle\rangle = class\langle t|f\rangle,$$

which can in turn be rewritten as

$$class ::> a\langle b\langle t|?\rangle|b\langle t|f\rangle\rangle.$$

The latter representation is obtained by replacing every variable that is not among the outcomes of *class* with the symbol "?". This symbol indicates that the leaf is unreachable, and therefore irrelevant to the decision problem.

Functional identities provide a useful representation for induction problems in general, and particularly for induction from incomplete training data.

4.1.2 Migration

In order to take advantage of the functional identity representation, it is necessary to present a rule that operates on terms with unreachable leaves, also introduced by Herrera (1988).

Definition 4.2 A *deduction node* in a term t is an identity at a leaf of t of the form $x \Rightarrow t'$, read "x entails t'." A deduction node can be treated as a leaf in the sense that its antecedent is a valid leaf label. The deduction node $x \Rightarrow x$ is equivalent to a leaf x.

The migration rule illustrates how deduction nodes might be formed.

D.6: Migration.

$$(q\langle x \Rightarrow t_1|\ldots|x \Rightarrow t_{i-1}|?|x \Rightarrow t_{i+1}|\ldots|x \Rightarrow t_n\rangle)$$
$$\rightarrow (x \Rightarrow q\langle t_1|\ldots|t_{i-1}|?|t_{i+1}|\ldots|t_n\rangle).$$

Migration is very similar to idempotence reduction. However, whereas idempotence is defined as an identity (of which only the simplifying direction is used in the reduction algorithms), the direction in which migration is applied is enforced. The major difference between them is that the condition for the application of migration is weaker. Migration does not require that all leaves (or deduction node antecedents) be equal; it is sufficient that all reachable leaves be equal. From the viewpoint of induction, this makes migration a more powerful simplification operator, for it makes "convenient" assumptions about unseen events. The assumption is that if an unseen configuration of feature values that appears in the tree as an unreachable leaf were in fact a possible event, its value on the class attribute would be such that it would enable idempotence reduction. Since the tree has been built without evidence to the contrary, application of migration yields a consistent — and hence valid — induced description. A second difference between idempotence reduction and migration is that instead of producing a leaf, as the former does, the latter operator produces a deduction node whose conclusion is the tree from which the antecedent was reduced.

Substituting migration for idempotence reduction in the generating conditions for the reasonable preorder produces a larger preorder that includes pairs of trees with unreachable leaves. The following example shows the effect of applying Algorithm 2.1, with migration substituted for idempotence reduction.

Example 4.3 Replacing idempotence reduction with migration in Algorithm 2.1 (`reduce`), and applying the modified algorithm to the tree of Figure 4.1 yields the decision tree of Figure 4.2. The entailed tree is the one that "hangs" from variable t.

Substituting migration for idempotence reduction in the generating conditions for the weak reasonable preorder produces a further enlargement. The algorithmic

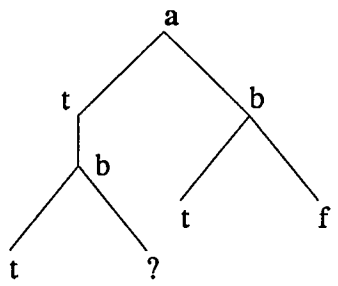

Figure 4.2. Irreducible form of the tree in Figure 4.1, using the migration operator.

interpretation of the interaction between migration and weak transposition is obtained by modifying Algorithm 3.3 (`whisker-branch-reduce`): if the target tree is a deduction node, then instead of returning the antecedent, or leaf value, apply the function to the conclusion tree. The following example illustrates the sort of reduction afforded by this modification.

Example 4.4 Starting from the partially reduced tree of Figure 4.2, b is pulled

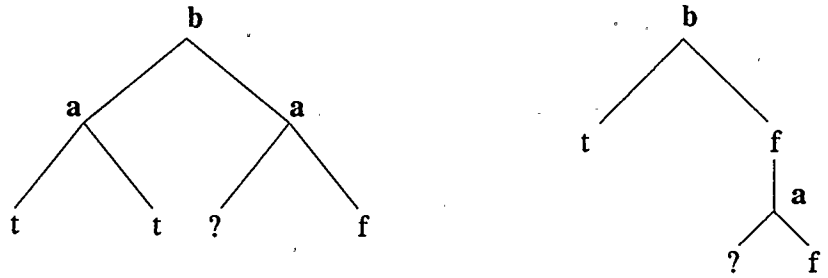

Figure 4.3. Use of the modified branch reduction on the tree of Figure 4.2.

up using the modified version of `whisker-branch-reduce`. Figure 4.3(i) shows the result of the first step, prior to migration. Figure 4.3(ii) shows the final result after migration. After the entire tree has been reduced, the deduction nodes may be replaced by their antecedents; in this case, the result is $b\langle t|f\rangle$.

In the rest of the discussion, it may be assumed that the reduction and whisker reduction algorithms are used with the migration operator wherever applicable.

Use of the migration operator in whisker reduction enables the induction of compact and accurate representations from incomplete data.

Example 4.5 In Example 3.7, the decision tree of Figure 3.12 was induced from the complete space of 24 sample contact lens prescriptions. Gaines (1991) identified 14 of those cases [1] as being critical to induce the correct rules for the problem. Using the methods described in this section, these critical cases yield the same decision tree as the complete training set.

4.1.3 Generating trees from incomplete data

If the training examples cover only a small portion of the space defined by the set of attributes, it is wasteful to generate a full tree, as this increases the running time of reduction. The alternative is to expand the tree, as required, with each new example that is inserted. This method can be used with two different strategies. The first is to split leaves only when the new event would create an inconsistency (*lazy* strategy), and the second is to create a full branch corresponding to each event (*full branch* strategy).

It is clear that the lazy strategy creates smaller trees, while the other creates more unreachable leaves. These unreachable leaves can be used to advantage by the whisker reduction algorithm to accomplish greater simplification. The reader may assume, for the rest of the discussion, that all decisions trees are built by generating a full branch for every event, unless the proper branch already exists.

[1] The cases are numbers 1–4, 6, 8, 10, 12, 13, 16, 18, 20, 22 and 24 in Table 3.2

4.1.4 An experiment on a large data set

The whisker reduction algorithm was compared to standard reduction, and to C4.5 on a large incomplete set. The "551" chess endgame data due to Quinlan (1987) consists of 551 chess board positions described by 39 binary attributes — all of whose values are correct — and classified as "safe" or "unsafe" positions for one of the players. This set is interesting because it is sizable, yet covers a very small portion of the space defined by the attributes. The set was randomly partitioned 15 times into learning and testing portions of roughly equal sizes. On each occasion the entire learning portion was used to induce a decision tree using each of the three methods, and each of the trees was used to classify the test data. The results of these tests are reported in Table 4.1, which summarize average tree sizes (in nodes) and successful classification rates on the unseen events. The trees produced by whisker

Method	Nodes	Hit rate
C4.5	93.9	86.6%
Whisker reduction	102.9	85.9%
Standard reduction	573.7	71.1%

Table 4.1. Comparative results on 551 chess endgame data.

reduction were on average about nine percent larger than those produced by C4.5. The predictive accuracy of C4.5 trees on the unseen data was somewhat better. These results also show the magnitude of the advantage of whisker reduction over standard reduction, in both size and predictive accuracy. It must be noted that approximately 45% of the errors produced by the trees that had been reduced by the standard method were due to "unknown" classifications, that is, by events that were classified by unreachable leaves. The whisker reduced trees had only a negligible number of unreachable leaves; on average, less than one half of one percent of the

test data could not be classified.

4.2 Noise

The presence of incorrect descriptions in the data set is likely to give rise to trees that classify unseen events with high error rates. Furthermore, noisy descriptions force tree building algorithms to discriminate among events when, were the noise not present, no discrimination would be required. This results in trees that are unduly large and complex (Quinlan, 1987; Quinlan, 1986a). Events that are not defined on all of their attributes are of questionable value for the induction process, yet discarding them could result in the loss of valuable information, and might indeed prevent any induction if this defect is highly frequent in a given data set.

This section examines the methods used in ID3 and C4.5 to deal with these problems, and incorporates some of them to principled induction by whisker reduction.

4.2.1 Incorrect descriptions

The reader will recall that ID3 and C4.5 build decision trees top-down, and the decision to be made at a given node is determined by a statistical criterion. The main stopping criterion is that the set of events at the node be entirely homogeneous. An additional stopping criterion, used to prevent tree complexity that is due to noise, is to collapse subtrees into terminal nodes when this does not lead to an increase in the absolute number of errors with respect to the training events from which the subtree was constructed (Quinlan, 1990b). In fact, this particular criterion constitutes a form of pruning, since the subtrees must be generated so that their absolute error can be

computed to evaluate the convenience of discarding them [2]. Lastly, when the current node is the last on a branch that already includes the entire set of attributes, there are obviously no decisions left on which to branch. This occurs when the attribute set is insufficient to express differences among classes, or when some of the event descriptions have been corrupted.

Use of the last stopping criterion gives rise to nodes that must necessarily be leaves, yet have no unique class label. The classification errors can be reduced by labeling such leaves with the most frequent class in the subset of the training events being processed at the node (Breiman et al., 1984; Quinlan, 1986a). It is important to note that the value of this labeling criterion rests heavily on the assumption that the class distribution of the training events is highly representative of the population from which they are drawn.

This labeling criterion was adopted to resolve inconsistencies at the leaves of the trees built before the reduction process, and experiments were performed to compare the results, in terms of tree size and predictive accuracy, to those of C4.5. As these experiments were aimed at comparing principled induction by whisker reduction to the basic top-down method, C4.5 was forced to report the fully grown trees, unpruned with the absolute error criterion described above. These experiments were performed on the two following noisy data sets:

Digits This domain was presented by Breiman et al (1984). It consists of 3000 events described by seven binary attributes, each of which represents the status of an LED, as used to represent the digits 0–9. Every feature value of each event has been inverted with probability 0.1.

[2]This stopping criterion, used in C4.5, replaces the χ^2 significance testing used by ID3 (Quinlan, 1986a; Quinlan, 1986b) to verify the selected branch attribute's relevance to the classification.

Disjunction The 600 events are described by nine binary attributes. Events that satisfy the propositional formula

$$(a_0 \wedge a_1 \wedge a_2) \vee (a_3 \wedge a_4 \wedge a_5) \vee (a_6 \wedge a_7 \wedge a_8)$$

are classified as members of class Y with probability 0.9 and as members of N with probability 0.1. Other events are classified as members of Y or N with the respective probabilities 0.1 and 0.9.

Method	Nodes	Hit rate
C4.5	190.2	72.7%
Whisker reduction	116.3	72.3%

Table 4.2. Comparative results on the Digits data.

Each of the sets was partitioned randomly 15 times to produce learning and test files of approximately equal sizes. With each partition, the entire learning set was used by C4.5 and whisker reduction to produce a decision tree. The average tree

Method	Nodes	Hit rate
C4.5	168.1	77.1%
Whisker reduction	173.8	75.0%

Table 4.3. Comparative results on the Disjunction data.

sizes and successful classification rates on the corresponding test sets are reported in Tables 4.2 and 4.3.

C4.5 trees obtained better predictive accuracy than whisker reduction in both cases; on the Disjunction data the difference was 2.1%, and on the Digits data, 0.4%. On the Digits data whisker reduction produced trees that were considerably

smaller than C4.5's, whereas on the Disjunction data, C4.5 induced trees than were on average approximately three percent smaller than those produced by whisker reduction.

These results indicate that the principled induction method using whisker reduction produces trees that are comparable in compactness and predictive accuracy to those built top-down using statistical criteria.

4.2.2 Reduction of complexity due to noise

Quinlan (1987, 1992) devised *pessimistic pruning* methods to further reduce tree complexity due to noisy examples. These methods are based on making a pessimistic estimate of the error rate to be expected when the tree is used to classify unseen events. Then the tree is replaced with a leaf labeled with the most frequent class in the tree. If the error caused by this replacement with respect to the events used to form the original tree is less than the pessimistic estimate, then the replacement is made permanent.

It will be shown that trees built top-down by C4.5, using the average entropy of the subsets resulting from a partition as the criterion for selecting a test attribute, are particularly well suited for pessimistic pruning. The trees resulting from pruning C4.5 trees are much more compact and have better predictive accuracy than trees pruned from whisker reduced trees. It will also be shown that in spite of this improved *prunability*, applying a combination of pessimistic pruning and the structural manipulation technique used in tree reduction can result in further improvement in the compactness of pruned C4.5 trees.

Pessimistic pruning

What follows is a description of the latest pessimistic pruning method due to Quinlan (1992), a version of which is used by C4.5.

Consider a terminal node t formed from N events, J of which are misclassified. The number of errors can be treated as a random variable with binomial distribution[3].

The probability p that a random unseen event will be misclassified by t can be bounded from above by p_0 with certainty level $1 - \alpha$, where p_0 is found by solving

$$\alpha = \sum_{i=0}^{J} C_i^N \cdot p_0^i \cdot (1 - p_0)^{N-i}.$$

The notation used will be $p_0 = \bar{B}_\alpha(J, N)$.

For a general tree t', the upper bound p_0 can be determined by averaging the upper bounds of it subtrees, weighted by the number of events N_i from which each subtree t_i was generated. This p_0 can be treated as a pessimistic estimate of the probability of error of when t' is used to classify unseen events.

To estimate the quality of the leaf obtained by pruning t' to a leaf labeled $BestClass(t')$, the upper bound p_1 of the probability of error is computed with N, the total number of events from which t' was generated, and J, the number of those events that do not belong to $BestClass(t')$. The tree is pruned to a leaf if $p_1 < p_0$.

This procedure is applied to a tree from the bottom up, as shown in the following example.

Example 4.6 In order to allow application of the pessimistic pruning algorithm, every leaf of the tree must have, in addition to a class label, a count of the events

[3]Quinlan cautioned that the statistical underpinnings of this method "should be taken with a large grain of salt," and indicated that the method's merit lies in the quality of its results.

of each class covered by that leaf. Such a tree is shown in Figure 4.4(i), where the notation $(A.B)$ indicates that the leaf covers A events of class B.

To apply pessimistic pruning to this tree, we start with the left subtree, with test b at its root. The confidence limit used throughout the rest of this discussion will be $\alpha = 0.25$. The error probabilities for the subtree's three terminal nodes, from left to right, are $\bar{B}_{0.25}(0, 80) = 0.0172$, $\bar{B}_{0.25}(0, 50) = 0.0273$, and $\bar{B}_{0.25}(0, 1) = 0.7500$. In

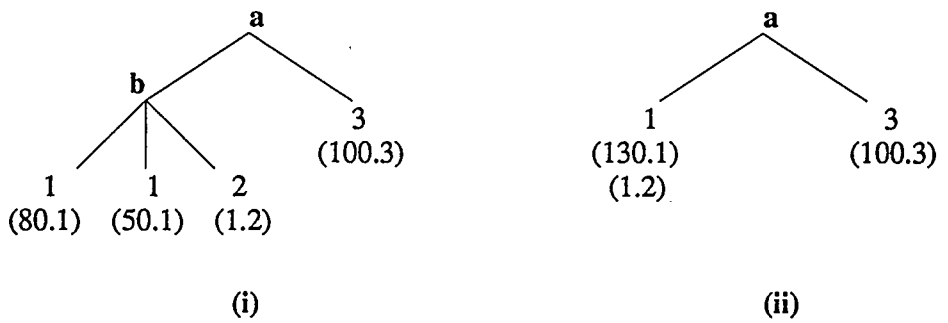

Figure 4.4. An example of pessimistic pruning (Example 4.6).

all cases $J = 0$, since none of the leaves misclassify any of the training events. The error probability for the subtree with b at the root is the weighted sum of these three values:

$$\frac{80}{131} \cdot 0.0172 + \frac{50}{131} \cdot 0.0104 + \frac{1}{131} \cdot 0.7500 = 0.0266.$$

If the subtree were to be replaced by a leaf labeled "1", the leaf would misclassify one out of 131 training events, so the error probability would be given by $\bar{B}_{0.25}(1, 131) = 0.0204$, which is lower than the average probability of the subtrees. As a result, the tree is pruned to that of Figure 4.4(ii).

The probability of error in the new tree is

$$\frac{131}{231} \cdot 0.0204 + \frac{100}{231} \cdot \bar{B}_{0.25}(0, 100) = 0.0176.$$

No further changes are made because pruning the tree to a leaf would give a higher error probability ($\bar{B}_{0.25}(100, 231) = 0.4616$).

The results of applying pessimistic pruning to the trees generated from the Digits and Disjunction and shown in Tables 4.4 and 4.5, which also incorporate the summaries in Tables 4.2 and 4.3. The prefix "P-" in the method names indicates that the trees were pessimistically pruned. On both data sets the pruned C4.5 trees were

Method	Nodes	Hit rate
C4.5	190.2	72.7%
P-C4.5	63.0	72.6%
Whisker reduction	116.3	72.3%
P-Whisker reduction	96.6	72.4%

Table 4.4. Comparative results on the Digits data after pessimistic pruning.

considerably smaller and had better classification rates on the unseen data. This seems puzzling, considering that both methods were applied to trees of similar size and predictive accuracy. To explain this phenomenon it is necessary to reexamine

Method	Nodes	Hit rate
C4.5	168.1	77.1%
P-C4.5	45.5	81.8%
Whisker reduction	173.8	75.0%
P-Whisker reduction	70.0	76.5%

Table 4.5. Comparative results on the Disjunction data after pessimistic pruning.

the ways in which the respective methods build the trees prior to pruning. The method presented in this work builds trees by random selection of test attributes and then reduces them by structural manipulation; C4.5 carefully selects every at-

tribute from the root down to minimize the heterogeneity of the resulting partitions of the training set.

Here lies the advantage of the statistical method which results in improved prunability. Sets with low impurity values are those that require little information to discriminate among their elements. These are typically sets in which some class is strongly represented, and others appear with low frequency. This characterization, albeit approximate, is also true of sets whose corresponding statistically built decision trees are configured propitiously for pessimistic pruning. This is illustrated in the following example.

Example 4.7 Figure 4.5 shows two transpose equivalent trees that correspond to a training sets consisting of 100 events of class 1, 100 events of class 3, one event

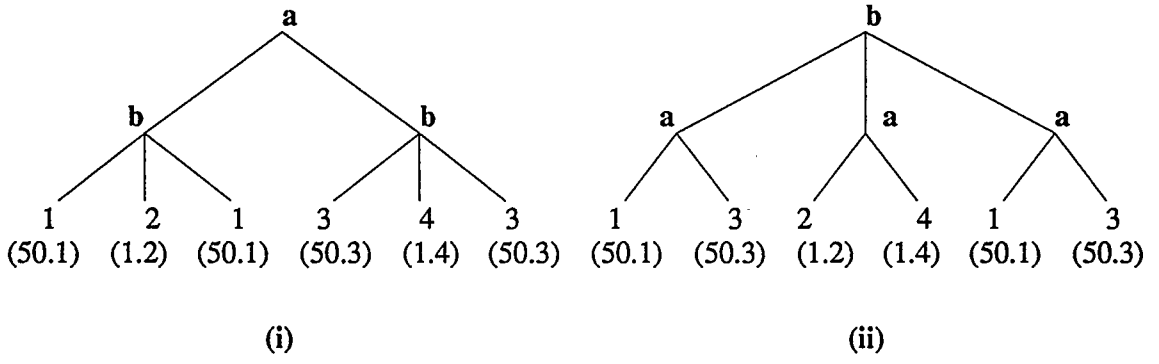

Figure 4.5: Two transpose equivalent trees: **(i)** Prunable tree built to minimize entropy; **(ii)** The only transpose, which cannot be pruned.

of class 2, and one event of class 4. To calculate the average entropy of the subsets obtained by partitioning on attribute a, it is sufficient to calculate the entropy of the subset in the left subtree of Figure 4.5(i) (the entropy of the subset in the right subtree is the same), given by

$$-[\frac{100}{101} \cdot log_2 \frac{100}{101} + \frac{1}{101} \cdot log_2 \frac{1}{101}] = 0.0803.$$

Likewise, to calculate the average entropy of the subsets obtained by partitioning on b, it is sufficient to calculate the entropy of the subset in the leftmost subtree of Figure 4.5(ii):

$$-2 \cdot \frac{50}{100} \cdot log_2 \frac{50}{100} = 1.$$

Since partitioning on a gives a lower average entropy on the resulting subsets, C4.5 would select this test to be at the root of a tree induced from the 202 events. Let us now analyze the prunability of each of these trees. The error probability of the left subtree of Figure 4.5(i) is

$$\frac{50}{101} \cdot \bar{B}_{0.25}(0, 50) + \frac{1}{101} \cdot \bar{B}_{0.25}(0, 1) + \frac{50}{101} \cdot \bar{B}_{0.25}(0, 50) = 0.0344.$$

Replacing this subtree with a leaf labeled "1" would produce one error out of 101 events, so the leaf's error probability is given by $\bar{B}_{0.25}(1, 101) = 0.0264$. Since this error probability is lower, the left subtree would be pruned to a leaf. The same calculations are valid for the right subtree; hence the tree of Figure 4.5(i) would be pruned to $a\langle 1|3\rangle$. No further pruning of this tree is possible, as there is no dominant class. The same is true of the tree in 4.5(ii); clearly, no pessimistic pruning is possible, either at the subtrees or at the root.

The greater pruning achieved on C4.5 trees is explained by the fact that these trees are suited for pruning — which is not necessarily the case with the trees produced by whisker reduction. The superior predictive accuracy of pruned C4.5 trees is explained by the fact that pessimistic pruning works on the tree from the bottom up, thus increasing the likelihood of eliminating decisions in the lower portion of the tree — the decisions that were delayed because of their relatively low discriminating power.

The fact that C4.5 trees are well suited for later pruning does not mean, however, that their configuration enables every possible application of pessimistic pruning. Substituting a pessimistic pruning routine for migration in whisker reduction makes it possible to utilize the restructuring power of the tree reduction algorithm to search for further applications of pessimistic pruning.

Algorithm 4.1 Algorithm `dynamic-prune` is obtained by substituting a pessimistic pruning routine for migration in Algorithm 3.7 (`whisker-reduce`).

A new experiment of 15 trials, as before, was performed on the Disjunction data to test Algorithm 4.1. Each time, the full tree induced by C4.5 was pruned directly (P-C4.5), and the pruned version was further pruned dynamically (DynamicPrune-P-C4.5), using the modified reduction algorithm. This algorithm was also applied directly to the unpruned tree induced by C4.5 (DynamicPrune-C4.5). The average results are reported in Table 4.6. Comparison of the results from P-C4.5 and

Method	Nodes	Hit rate
P-C4.5	46.6	82.8%
DynamicPrune-C4.5	50.3	82.7%
DynamicPrune-P-C4.5	21.8	81.3%

Table 4.6: Pessimistic pruning and dynamic pruning results on the Disjunction data

DynamicPrune-C4.5 confirm that the C4.5 trees are better suited for pruning. Although DynamicPrune-C4.5 evaluated a greater number of pruning opportunities, these opportunities were evidently not the most propitious, since it produced larger trees, with slightly lower classification accuracy. On the other hand, DynamicPrune-P-C4.5 attempted dynamic pruning only after all of the convenient pruning opportunities present in the original tree had been processed, and then reshaped the tree in

an attempt to make further improvements. This resulted in a more than 50% reduction in size, with a loss of one and a half percentage points in classification accuracy. The optimal tree built from a complete set of noise-free Disjunction examples would have 19 nodes, and its best classification accuracy on unseen examples, corrupted with the probabilities discussed before, would be 90.0%

It is important to note that Algorithm 4.1 lacks the theoretical foundation of the original reduction algorithm. The basic algorithm guarantees that after a tree's branches have been reduced, any further improvement can only be produced by factoring the root decision to the roots and (perhaps) eliminating it. Pessimistic pruning is a much deeper reduction operator that idempotence reduction, and there is no guarantee that the best possible pruning can be achieved when the root decision reaches the level above the leaves. Optimal pruning might in fact occur with the original root decision somewhere between the root and the leaves. This indicates that dynamic application of pessimistic pruning is a more complex problem than dynamic application of migration. While evidence was given here that dynamic pessimistic pruning can be expected to improve decision trees, work remains to be done to determine how a good order of pruning is to be selected during the tree restructuring process.

4.2.3 Undefined values

Training examples with undefined values on some of their attributes pose a special problem, since it is not immediately evident where they fit in a decision tree, or how the information they contain can be used to facilitate the induction task. While it is tempting to discard such examples, the loss of the information they contain can be expected to hinder the induction task. This was found to be the case through

empirical studies reported by Quinlan (1989).

In the same studies it was found that good performance could be achieved by assigning a fraction of each case with an undefined value on test a to each of the subsets created by partitioning on a. This method, used in C4.5, assigns the fractional values on the basis of the relative frequencies of the known values of a in the set.

A similar approach was taken to insert events with unknown attribute values into trees to be used as initial valid descriptions. Prior to insertion, each event is assigned a weight $w = 1$. When inserting an event whose value on a is undefined into a subtree $a\langle t_1 | \ldots | t_n \rangle$, the event is redirected into each t_i with weight $\frac{w}{k}$. This insertion method generates a great deal of unreliable information at the leaves, particularly since no consideration is given to the relative frequencies of values on a. However, this was solved effectively by treating any leaf as unreachable (or "unknown") until the most frequent class it covers appears in a proportion that is at least as large as its frequency in the entire training set. This allows the "?" decision at the leaf to be migrated until a clear mode emerges at the leaf. Once such a mode appears, the leaf label is bound to the most frequent class, and the leaf becomes reachable.

Method	Nodes	Hit rate
C4.5	120.5	71.7%
Whisker reduction	117.5	72.6%

Table 4.7: Results on the Digits data, with each attribute value deleted with probability 0.5.

Tables 4.7 and 4.8 show the average results obtained on 15 trials involving the partition of the Digits and Disjunction data into roughly equal parts, for learning and testing, where every attribute value of each event selected for the learning set was deleted with probability 0.5. The results reported for C4.5 corresponded to trees

Method	Nodes	Hit rate
C4.5	68.5	73.9%
Whisker reduction	36.7	74.7%

Table 4.8: Results on the Disjunction data, with each attribute value deleted with probability 0.5.

after the absolute error pruning mentioned in Section 4.2.1 — not to be confused with pessimistic pruning, which was not used at all. Whisker reduction performed better on both data sets, particularly in terms of tree size.

4.3 Summary

In this chapter it was shown how the whisker reduction algorithm can be used effectively to induce decision trees from incomplete and noisy data. Empirical results obtained using large, noisy data sets indicate that principled induction by whisker reduction produces trees that are comparable in compactness and predictive accuracy to those built by C4.5 using statistical criteria. These results validate the claim that principled induction is an effective representation of the problem of learning concept descriptions from examples.

An important observation arising from the experimental use of pessimistic pruning jointly with tree reduction is that statistically built trees are better suited for pruning than those simplified by whisker reduction.

This suggests a topic for further research: improving the dynamic pruning algorithm. The structural manipulation technique can be used to advantage as a way to find good pruning opportunities. Although the cost of applying such an algorithm to unreduced trees (e.g., before whisker reduction) would probably be prohibitive,

the cost of dynamically pruning C4.5 trees that have already been pessimistically pruned is likely to be absorbed by the cost of all previous processing.

Chapter 5

Principled induction of decision rules

Decision trees have often been rated as inadequate for use in expert systems because they lack the explanatory value often required of concept definitions and are ill-suited for performing backward chaining (Cendrowska, 1987; Quinlan, 1987).

This has motivated methods for transforming decision trees into sets of decision rules (Quinlan, 1987), and for inducing decision rules directly from data using statistical methods much like those underlying ID3 and CART, as proposed by Cendrowska (1987).

The difference between tree representations and rule sets is that whereas the former have a built in control strategy, decision rules must be complemented by some external control strategy (e.g., order of application) (Cockett & Herrera, 1986).

This suggests that decision trees and rule sets are not directly comparable. A decision tree may in fact be abstracted as a pair (P, D), where D is an intensional description of the space from which the tree was generated, and P is a body of procedural knowledge pertaining to the use of D.

Under this definition the problem of transforming decision trees into sets of decision rules is that of extracting the descriptive knowledge from the tree. A good solution to this problem can also be expected to provide a method for inducing decision rules via a tree representation created from a training set (e.g., using Algorithms 3.1 and 3.2). This extraction process is the topic of the present chapter. The first section characterizes good decision rules as rules that are *prime*. The second presents a principled induction algorithm for extracting prime decision rules from

$\delta = 1$

ID3	PRISM
$(d = 2) \wedge (c = 2) \wedge (b = 1)$	$(c = 2) \wedge (d = 2) \wedge (b = 1)$
$(d = 2) \wedge (c = 2) \wedge (b = 2) \wedge (a = 1)$	$(a = 1) \wedge (c = 2) \wedge (d = 2)$

$\delta = 2$

ID3	PRISM
$(d = 2) \wedge (c = 1) \wedge (b = 1) \wedge (a = 1)$	$(c = 1) \wedge (d = 2) \wedge (a = 1)$
$(d = 2) \wedge (c = 1) \wedge (b = 1) \wedge (a = 2)$	$(c = 1) \wedge (d = 2) \wedge (a = 2)$
$(d = 2) \wedge (c = 1) \wedge (b = 2)$	$(c = 1) \wedge (d = 2) \wedge (b = 2)$

$\delta = 3$

ID3	PRISM
$(d = 1)$	$(d = 1)$
$(d = 2) \wedge (c = 1) \wedge (b = 1) \wedge (a = 3)$	$(a = 3) \wedge (b = 1) \wedge (c = 1)$
$(d = 2) \wedge (c = 2) \wedge (b = 2) \wedge (a = 2)$	$(b = 2) \wedge (c = 2) \wedge (a = 2)$
$(d = 2) \wedge (c = 2) \wedge (b = 2) \wedge (a = 3)$	$(b = 2) \wedge (c = 2) \wedge (a = 3)$

Table 5.1: Rules generated by ID3 and PRISM for each class δ in the data of Table 3.2.

decision trees.

5.1 Good decision rules

Quinlan (1987) approached the problem of extracting rules from decision trees by treating every path from the root to a leaf as a distinct decision rule, which could later be pruned using statistical criteria.

Table 5.1 shows the decision rules extracted trivially from the tree of Figure 3.12, and those induced directly by PRISM (Cendrowska, 1987) from the data of Table 3.2 (Example 3.7).

Cendrowska pointed out that the PRISM rule set is better because several of its elements contain fewer antecedents than there are tests on the paths of the decision tree. In fact, her algorithm is based on the observation that the goals of rule

generation differ from those of decision tree generation. Tests that are required to provide adequate control of the decision process in a tree may not be relevant to the classification of members of a particular class. A "correct" rule is "one which references all the relevant attributes and no irrelevant ones" (Cendrowska, 1987).

This statement points toward the desirability of rule sets that contain only elements that are *prime.*

Definition 5.1 A decision rule r_1 *subsumes* [1] r_2 if

1. Both rules have the same conclusion, and
2. The decision-outcome pairs in r_1 constitute a subset of the decision-outcome pairs in r_2.

Definition 5.2 A decision rule r in a rule set R that is complete and consistent with respect to instances S is said to be prime if there is no other rule r' that properly subsumes r, and can replace r in R such that the rule set remains consistent.

In order for a rule set to be optimal, it is also necessary that it contain no redundant rules. The rule extraction method presented in the next section fails to meet the second requirement. It is this shortcoming that makes it computationally feasible.

5.2 Extraction of prime decision rules from trees

A very general description of an algorithm for extracting prime rules from decision trees is given in (Herrera, 1988). This algorithm examines every path in the tree separately, attempting to identify test-outcome pairs that can be pruned away without

[1] This definition was adapted from one by Genesereth and Nilsson (1987).

losing consistency. This section presents in detail a principled induction algorithm that performs the task by sweeping the tree from the bottom up.

5.2.1 The extraction algorithm

The algorithm for extracting prime rules is based on the observation that if at a given tree t, some of the rules generated from the paths of one of t's subtrees are subsumed by some rule generated from each of the remaining subtrees, then it is unnecessary to augment those subsumed rules (set SR in Algorithm 5.1) with the test at the root of t.

Example 5.1 In the tree of Figure 5.1, the right subtree $b\langle 1|2\rangle$ has trivial rules

$$\{(b = 1) \Rightarrow 1; (b = 2) \Rightarrow 2\}$$

and the left subtree, 1, has the singleton rule set $\Rightarrow 1$. The trivial rule set for the entire tree is

$$\{(a = 1) \Rightarrow 1; (a = 2) \wedge (b = 1) \Rightarrow 1; (a = 2) \wedge (b = 2) \Rightarrow 2\}.$$

The rule $(a = 1) \wedge (b = 1) \Rightarrow 1$ is not prime, because replacing it with $(b =$

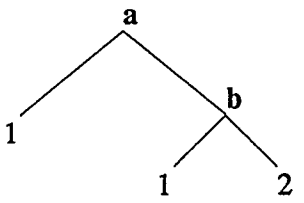

Figure 5.1. A shallow whisker.

$1) \Rightarrow 1$ leaves the set consistent. To see why this is the case, observe that although $(b = 1) \Rightarrow 1$, a rule generated for the right subtree, covers events in both subtrees,

it is still consistent because the events in the left subtree all belong to class 1. In general, the corresponding test-outcome pair corresponding to decision a may be omitted from $(a = 1) \wedge (b = 1) \Rightarrow 1$ because this rule is subsumed by some rule in the rule set of each sibling tree, namely, $\Rightarrow 1$.

The algorithm traverses the tree bottom up, searching for paths that can be reduced to primeness by "pruning" specific tests, as discussed in the previous example. This is the simplification step used in the algorithm. All other rules — those that are not included in SR — are treated as they normally would be during trivial extraction of rules from a tree. If the test at the root of t is a, then rules generated from the i^{th} subtree of t — from left to right — are augmented with the antecedent conjunct $(a = i)$.

Formal presentation of the algorithm

Algorithm 5.1 Given as input a repeat reduced decision tree t that is complete and consistent with respect to a bag of training examples S, this algorithm returns a set of prime rules R that is also complete and consistent with respect to S.

`extract-prime-rules`(t)

if t is a leaf labeled with outcome x **then**

return$(\{\Rightarrow x\})$

else %t is a tree with test a at its root and subtrees $t_1, \ldots, t_k$%

$(R_1, \ldots, R_k) \leftarrow$ map `extract-prime-rules` over $(t_1, \ldots, t_k)$

$SR \leftarrow \{\}$

for $i = 1, \ldots, k$

for every $r \in R_i$

if every R_j, $j \neq i$, contains a rule r' that subsumes r

then $SR \leftarrow SR \cup \{r\}$

for $i = 1, \ldots, k$

$R_i \leftarrow R_i - SR$

$R \leftarrow \bigcup_{i=1}^{k} \bigcup_{r \in R_i} (a = i) \wedge r$

return$(R \cup SR)$

Example 5.2 Figure 5.2 shows the rules generated by Algorithm 5.1 for a portion of the tree in Figure 3.12. Terminal nodes return singleton sets; the only rule is one without antecedents and with the conclusion associated with the leaf. What is noteworthy in this example is that the rule set for node $n1$ contains the rule $(a = 1) \Rightarrow 3$; the antecedent $(b = 2)$ has been omitted because the rising rule $(a = 1) \Rightarrow 3$ is subsumed by the rule from $n2$, $\Rightarrow 1$.

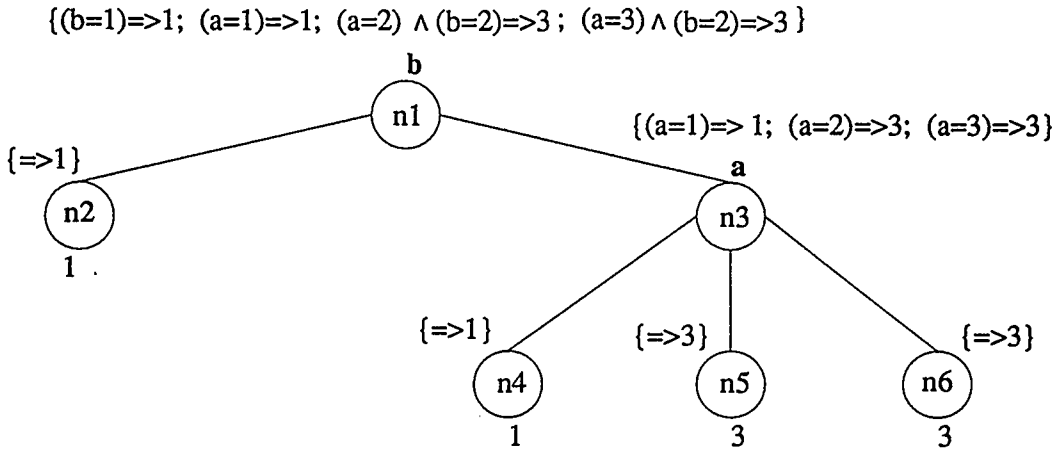

Figure 5.2: The rules generated at each node for a portion of the tree in Figure 3.12.

If this procedure is applied to the entire tree of Figure 3.12, the resulting rules are the same as those generated by PRISM and shown in Table 5.1.

Proposition 5.1 Given a decision tree t that is complete and consistent with respect to training examples S, algorithm 2.1 generates a set of rules R such that R is *complete* and *consistent* with respect to S, and every rule in R is *prime* with respect to S.

Proof Consider the base step. If t is a leaf labeled x, the algorithm returns the rule set $R = \{\Rightarrow x\}$. R is *complete* because a rule without antecedents can classify any instance. It is *consistent*, because the generating tree was constructed exclusively from examples in class x — and that is the only class that R is able to classify. The rule is *prime* because there cannot exist any rule that properly subsumes a rule with no antecedents.

If the tree is not a leaf, then let a be the test at its root, and let $t_1, \ldots, t_k$ be its subtrees, which return rule sets $R_1, \ldots, R_k$. The inductive hypothesis is that these

rule sets are complete and consistent, and that their elements are prime, with respect to the examples covered by $t_1, \ldots, t_k$. The inductive step is shown by contradiction, separately for each of the three properties. As a notational convention, S_t will denote the portion of S used to build a tree t that is complete and consistent with respect to S_t.

Completeness Suppose that R is inconsistent with respect to S_t; then there exists an example $e \in S_t$ that does not fire any of the rules in R. Let e be in S_{t_i}, and let $r \in R_i$ be one of the rules fired by e — by the inductive hypothesis there must be such a rule. If r was in SR, then it also appears unchanged in R, and that contradicts the assumption. Otherwise r appears in R with an additional antecedent, $(a = i)$. But since $e \in S_{t_i}$, this example must also fire the rule $(a = i) \wedge r$, which also contradicts the assumption.

Consistency If R is inconsistent, then then must be some example $e \in S_t$ that is misclassified by some rule $r \in R$. If r contains antecedent $(a = i)$, then e is covered by t_i and fires the rule in R_i from which r was generated. That rule must also misclassify e, in contradiction with the inductive hypothesis. If r does not contain antecedent $(a = i)$, it must have been in SR. Therefore every subtree returned a rule that subsumes it, and every such rule must also be fired by S, again in contradiction with the inductive hypothesis.

Primeness Suppose that there exists a rule $r_0 \in R$ that is not prime with respect to S_t. Then there exists some rule r_1 that properly subsumes r_0, such that r_1 is prime, and if r_0 is replaced by r_1 in R, the rule set remains consistent with respect to S_t.

First suppose that r_0 was not in SR when R was constructed. Then r_0 contains an antecedent $(a = i)$, and R_i contains a rule r_0' that is equal to r_0 but for

the lack of antecedent $(a = i)$. In other words, $r_0 = (a = i) \wedge r'_0$, and by the inductive hypothesis, r'_0 is a prime rule in set R_i.

Consider the relationship between r_1 and r'_0. If r_1 contains antecedent $(a = i)$, then $r_1 = (a = i) \wedge r'_1$, and r'_1 must properly subsume r'_0. The assumption that r_1 can replace r_0 in R implies that r'_1 can replace r'_0 in R_i, but this contradicts the inductive hypothesis that all rules in R_i are prime.

This argument is valid even if r_1 does not include an antecedent $(a = i)$, as long as r_1 properly subsumes r'_0.

Consider the case $r_1 = r'_0$. For r_1 to be prime in R — that is, consistent with respect to S_t — it must either contain an antecedent $(a = i)$, where a is the test at the root of t, or it must be subsumed by some rule in every R_j. We assumed that the first condition does not hold; therefore the second condition must hold, and r_1 is subsumed by some rule in every R_i. But $r_1 = r'_0$, implying that $r_0 = r'_0$, and this contradicts our assumption that r_0 was not in SR and must therefore have an antecedent $(a = i)$.

Suppose now that r_0 was in SR because it is subsumed by some rule in every R_j. Then r_0 is present in some R_i. However, the fact that r_1 subsumes r_0 and can replace it in R implies that the replacement is also valid in R_i, and this contradicts the inductive hypothesis that r_0 is prime in R_i.

Rule extraction as principled induction

Before showing that this is a principled induction algorithm, it is necessary to point out that the intermediate valid descriptions are hybrid in their structure. The initial description is a decision tree, and the result is a set of decision rules. During the extraction process, the valid transformations are decision trees with rule sets at the leaves. If one were to use an instance of this hybrid structure for classification, the

#	a	b	c	d	δ	#	a	b	c	d	δ
1	1	1	1	1	x_0	9	2	1	1	1	x_3
2	1	1	1	2	x_0	10	2	1	1	2	x_3
3	1	1	2	1	x_0	11	2	1	2	1	x_4
4	1	1	2	2	x_0	12	2	1	2	2	x_4
5	1	2	1	1	x_1	13	2	2	1	1	x_3
6	1	2	1	2	x_1	14	2	2	1	2	x_3
7	1	2	2	1	x_2	15	2	2	2	1	x_4
8	1	2	2	2	x_0	16	2	2	2	2	x_0

Table 5.2. Training examples used to build the tree of Figure 5.3.

tree classification procedure would be followed until a leaf is reached. Then the classification would be completed using the decision rules at that leaf.

That the algorithm preserves generality and consistency follows directly from the proof of Proposition 5.1. The algorithm determines a preorder on valid descriptions by detecting and eliminating irrelevant conditions from the descriptions of particular concepts or classes.

As anticipated above, the algorithm does not guarantee that the resulting output will not contain redundant rules. The following example illustrates how Algorithm 5.1 can return a rule set that contains redundant elements.

Example 5.3 The tree of Figure 5.3 is whisker reduced for the training data of Table 5.2.1. The rules for class x_0 are the following: The left subtree yields $(a = 1) \wedge (b = 1) \Rightarrow x_0$ and $(a = 1) \wedge (c = 2) \wedge (d = 2) \Rightarrow x_0$. Decision b is omitted from the latter rule because $(c = 2) \wedge (d = 2) \Rightarrow x_0$ is subsumed by $\Rightarrow x_0$. The right subtree gives rule $(b = 2) \wedge (c = 2) \wedge (d = 2) \Rightarrow x_0$, from which a has been omitted because the rule is subsumed by $(c = 2) \wedge (d = 2) \Rightarrow x_0$. Table 5.3 summarizes the events covered by each of these three rules. Rule 2 covers only cases that are also covered by the other rules, and is therefore redundant in the

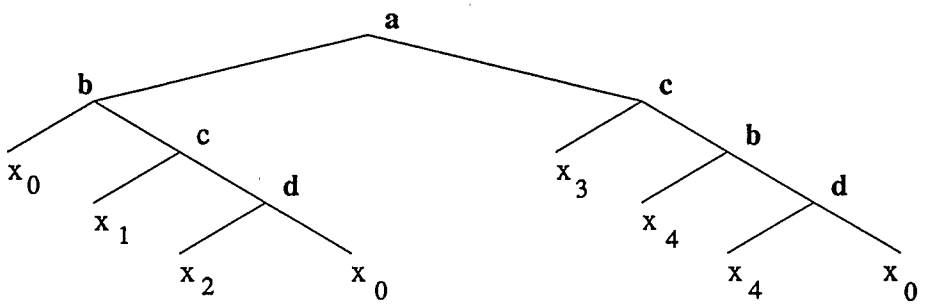

Figure 5.3. A whisker reduced tree that yields one redundant rule.

#	Rule	Examples
1	$(a = 1) \wedge (b = 1) \Rightarrow x_0$	$\{1,2,3,4\}$
2	$(a = 1) \wedge (c = 2) \wedge (d = 2) \Rightarrow x_0$	$\{4,8\}$
3	$(b = 2) \wedge (c = 2) \wedge (d = 2) \Rightarrow x_0$	$\{8,16\}$

Table 5.3. The rules for x_0 and the examples covered by each.

rule set.

The example also provides insight into the form of a decision tree that yields redundant rule sets by Algorithm 5.1. Both arguments of the root decision contain the path $(b = 2) \wedge (c = 2) \wedge (d = 2) \Rightarrow x_0$. If the pruning were delayed to the root decision, then it would be clear that one of the versions could be discarded. But the version of the path from the left argument is pruned because at an earlier stage it is subsumed by some other rule; hence equality is lost and neither rule may be discarded.

To obtain irredundant rule sets using Algorithm 5.1, every rule in the output must be checked for redundancy.

Time complexity of rule extraction

The following proposition analyzes the complexity of the rule extraction algorithm.

Proposition 5.2 Given a decision tree of height h and a maximum branch factor of B, the time complexity of Algorithm 5.1 is $O(h \cdot B^{2 \cdot h+1})$.

Proof Consider a full tree of height h and homogeneous branch factor B. The work at a given node is dominated by the need to check each rule in every subset of rules rising from one of B subtrees, for subsumption by the rules in the remaining subsets. The cost of checking two rules with n and m antecedents for subsumption — assuming that the antecedents are sorted by some ordering on the tests — is $2 \cdot max(m, n)$. If the node is the root of a subtree of height l, this cost is bounded by $2 \cdot l$. The cost of processing the node contains the following factors: B, the number of rule subsets rising from the subtrees; B^{l-1}, the bound on the number of rules in every subset that have to be checked for subsumption by other rules; $B-1$, the remaining subsets against which every rule must be checked; B^{l-1}, the number of rules in every other subset; and $2 \cdot l$, the cost of a single subsumption check.

This product is equal to $2 \cdot l \cdot B^{2 \cdot l}$. The number of nodes in the tree that are roots of subtrees of height l is B^{h-l}. Hence the cost of processing the entire tree is bounded by

$$2 \cdot B^h \cdot \sum_{1 \leq l \leq h} l \cdot B^l.$$

Using the identity for linear exponential sums,

$$\sum_{0 \leq i < n} i \cdot x^i = \frac{(n-1) \cdot x^{n+1} - n \cdot x^n + x}{(x-1)^2}$$

and substituting $h = n - 1$, $B = x$, and $l = i$, we obtain the bound

$$2 \cdot B^h \cdot \frac{h \cdot B^{h+2} - (h+1) \cdot B^{h+1} + B}{(B-1)^2} \leq 2 \cdot B^h \cdot \frac{h \cdot B^{h+2}}{B^2 - 2 \cdot B} \leq 2 \cdot h \cdot B^{2h+1},$$

which is $O(h \cdot B^{2 \cdot h+1})$.

Method	Rules	Tests	Relative time
PRISM	15	48	1.0
EXTRACT-1	15	46	0.6
EXTRACT-2	16	48	1.0
EXTRACT-3	15	46	1.7

Table 5.4. Results on the chess data described in Example 5.4.

If a tree is in full — every $q \in Q$ appears once — this bound translates to $O(|Q| \cdot B^{2 \cdot |Q|+1})$. The complexity of PRISM was given in Section 1.2.4 as $O(|S|^2 \cdot |A|^2)$, where S are the examples and A are the attributes on which they are defined. This translates to $O(B^{2|Q|} \cdot |Q|^2)$, which is somewhat larger if $|Q| > B$. The actual running times of the two algorithms on various examples were found not to differ significantly. Comparative running times on a sample training set are given in Example 5.4.

Example 5.4 The King-Knight-King-Rook chess endgame problem was formulated by Quinlan (1979). It describes chess board positions using four ternary and three binary attributes; the possible outcomes are *lost* and *safe* for one of the players. The set is of size 647, and hence almost complete; there is one combination of the seven attributes that is not legal. PRISM and the prime rule extraction algorithm were applied to the data, with the results reported in Table 5.4. The extraction algorithm was applied in three different ways. EXTRACT-1 denotes the extraction of prime rules from the tree built trivially from the examples. EXTRACT-2 corresponds to rule extraction from the whisker reduced tree for the data, and EXTRACT-3 is the result of eliminating redundant rules from the output of EXTRACT-2. The **Rules** column indicates the number of rules produced by each method, the **Tests** column indicates the total number of tests in the rule set, and the last column indicates the

relative running times required to produce the results, with the time required by PRISM taken as a base [2]. Extraction of prime rules from an unreduced tree gave the same number of rules as PRISM; in particular, both gave the same nine rules with outcome *safe*. As for the rules with outcome *lost*, they, too, were equal, except that two of the PRISM rules had one additional test each — the PRISM rule set was not prime. EXTRACT-2 produced a rule set that contained that of EXTRACT-1, and also included an additional rule with two tests rule and outcome *safe*. This is significant because it gives further evidence that the whisker reduced description can be mapped by Algorithm 5.1 onto rule sets with redundant elements. EXTRACT-3 (EXTRACT-2 plus the removal of redundant rules if any) returned the same rule set as EXTRACT-1. All three rule sets achieved perfect classification accuracy on the data from which they were generated.

5.2.2 Extraction of prime rules from noisy data

Since the rule extraction algorithm is purely qualitative, its output is subject to the same error rates the tree from it is generated. But as pointed out by Gaines (1991), the statistical criterion used by PRISM to grow rules is not sufficient in itself to cope with noise. INDUCT, Gaines' extension of PRISM, uses an additional quantitative criterion to filter out tests selected by the basic PRISM algorithm which are likely to have arisen by chance.

Another approach to inducing rules from noisy data is analogous to the testwise pruning of Herrera's algorithm, mentioned earlier in this chapter. C4.5 turns every path of a tree into a decision rule and rates the tests of each rule by their significance to the identification of the class that is predicted by the rule. If any of the tests are

[2]For the purpose of this comparison, PRISM was implemented in the same language as the reduction and rule extraction algorithms, *Chez Scheme*.

deemed to be individually insignificant, the least significant is pruned from the rule, and the process is repeated until no test of a rule can be judged to be irrelevant (Quinlan, 1987). This approach is very similar to the one used in INDUCT, since the basic PRISM algorithm also selects rule antecedents in order of significance. The major difference is that Quinlan's method works by gradual simplification.

Although the prime rule extraction algorithm could also be modified to use a quantitative criterion for pruning, the method would be at a disadvantage with respect to INDUCT and C4.5. Since the algorithm sweeps the tree from the bottom up, attempting pruning at each node, there is no possibility to sort the tests by some criterion prior to pruning.

5.3 Summary

The chapter presented an efficient principled induction algorithm that transforms an initial valid description represented as an arbitrary decision tree into a set of prime rules. Examples were shown which indicate that the degree to which the initial tree has been simplified has no bearing on the likelihood of obtaining redundant rules.

Chapter 6

Conclusions

This chapter summarizes the contribution of this thesis and points to some research topics arising from this work.

6.1 Summary and contribution

Principled induction from examples described by their values on a set of attributes was defined as the process of defining a set of valid transformations that generate — and therefore preserve — a reducing preorder on the set of valid descriptions, and then applying these transformations to some arbitrary initial valid description until a minimal element in the preorder is obtained. The goal of this thesis was to show that principled induction affords a clear and effective representation of the problem of learning concept descriptions from examples.

6.1.1 Clarity

To validate principled induction it was examined in the area of decision tree induction using the algebraic theory of decision trees by Cockett (1987a, 1987b, 1988), and the reasonable preorder generated by the manipulations provided by this theory. The operations used to generate this partial preorder were composed to form an efficient algorithm for finding minimal elements in the preorder (Cockett & Herrera, 1990).

It was shown that when used for principled induction this algorithm produced trees that were considerably more complex than those produced by statistical top-down tree building methods.

In order to improve the compactness of the trees obtained by principled induction, the reasonable preorder was enlarged by adding a new generating identity, weak transposition. The whisker reduction algorithm was formulated to obtain minimal elements in this weak reasonable preorder. This algorithm was shown to yield results comparable to those produced by statistical top-down tree induction methods.

This success shows the value of pure principled induction when tackling the computationally intractable problem of optimizing inductive conclusions by restricting the optimization to sparse preorders. Every operation that is added to the generators of the preorder brings with it an additional simplification capability, and an additional computational cost, that are well defined.

6.1.2 Effectiveness

A number of sample data sets were presented, in Chapters 3 and 4, on which the principled induction of decision trees produced results that were comparable in compactness and predictive accuracy to those built top-down using statistical criteria. Since both approaches give solutions that are not provably optimal, the comparative advantages of each are problem specific. In Chapter 5 a principled induction methods that produces prime decision rules from a decision tree representation of the training examples was shown to be somewhat more efficient and effective than an algorithm that grows rules using a statistical criterion.

6.2 Future work

The most immediate extension of the work presented in this thesis is the definition of additional preorders for the principled induction of decision trees. This can be done by identifying irrelevant tests which whisker reduction is not capable of elimination

from trees, and then defining operations that are capable of mapping the source tree to the decision equivalent tree with the irrelevant test removed.

In this thesis, principled induction was defined and illustrated in the context of a very restricted subset of inductive learning problems, in which examples are defined by their values on discrete attributes. Work remains to be done to define principled induction methods in more complex domains.

One such domain is that in which the feature values are continuous and are not amenable to a single categorization; different subsets of the training examples may be better described by different categorizations of the continuous domain. Another type of inductive learning is one in which the target concept is a relation, to be described in terms of other relations (Quinlan, 1990a). It is possible that these problems can be be treated as instances of the discrete attributes problem, once adequate categorizations of continuous values, or instantiations of logical predicates have been defined. The problems could then be solved using the principled induction methods described in this thesis.

In Chapter 4 it was shown that a variation of the principled induction method can be used to further reduce the complexity of trees that had been grown and pruned statistically. As indicated in the summary of that chapter, work remains to be done to produce an efficient algorithm that takes full advantage of the structural manipulations supplied by the algebraic decision theory of Chapter 2 and the extensions introduced in this thesis to find good applications of statistical pruning operators.

References

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees.* Monterey, CA: Wadsworth & Brooks.

Cendrowska, J. (1987). PRISM: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies, 27,* 349–370.

Chen, K. & Ras, Z. (1985). Homogeneous information trees. *Fundamenta Informaticae, VIII*(1), 123–149.

Cockett, J. R. B. (1987a). Decision expression optimization. *Fundamenta Informaticae, X,* 93–114.

Cockett, J. R. B. (1987b). Discrete decision theory: manipulations. *Theoretical Computer Science, 54,* 215–236.

Cockett, J. R. B. (1988). Coalgebraic decision theory. *Mathematical Problems in Computation Theory, 21,* 185–196.

Cockett, J. R. B. & Herrera, J. A. (1986). Prime rule-based methodologies give indadequate control. In *Proceedings of the ACM SIGART ISMIS,* (pp. 441–449)., Knoxville, TN.

Cockett, J. R. B. & Herrera, J. A. (1990). Decision tree reduction. *Journal of the Association for Computing Machinery, 37*(4), 815–842.

Gaines, B. R. (1977). System identification, approximation and complexity. *International Journal of General Systems, 3,* 145–174.

Gaines, B. R. (1991). The trade-off between knowledge and data in knowledge acquisition. In Piatetsky-Shapiro, G. & Frawley, W. J. (Eds.), *Knowledge Discovery in Databases,* (pp. 491–505)., Menlo Park, CA. AAAI Press.

Genesereth, M. R. & Nilsson, N. J. (1987). *Logical foundations of Artificial Intelligence,* (pp. 161–162). Los Altos, CA: Morgan Kaufmann.

Herrera, J. A. (1988). *Theoretical foundations and algorithms for the generation of optimal decision trees.* PhD dissertation, University of Tennessee, Knoxville.

Hyafil, L. & Rivest, R. L. (1976). Constructing optimal binary decision trees is NP-Complete. *Information Processing Letters, 5*(1), 15–17.

McCluskey, E. (1956). Minimization of boolean functions. *Bell Systems Technical Journal, 35*, 1417–1444.

Michalski, R. S. (1991). Toward a unified theory of learning: an outline of basic ideas. *Invited paper for the First World Conference on the Fundamentals of Artificial Intelligence, Paris, July 1–5.*

Michalski, R. S. & Chilausky, R. L. (1980). Knowledge acquisition by encoding expert rules versus computer induction from examples: a case study involving soybean pathology. *International Journal of Man-Machine Studies, 12*, 247–271.

Michalski, R. S., Mozetic, I., Hong, J., & Lavrac, N. (1986). The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, volume 2, (pp. 1041–1045)., Los Altos, CA. Morgan Kaufmann.

Mitchell, T. M. (1978). *Version spaces: an approach to concept learning.* PhD dissertation, Stanford University.

Niblett, T. (1988). A study of generalisation in logic programming. In *Proceedings of EWSL 88*, (pp. 131–138)., London. Pitman.

Post, H. R. (1960). Simplicity in scientific theories. *The British Journal for the Philosophy of Science, 11*, 32–41.

Quine, W. V. (1952). The problem of simplifying truth functions. *American Mathematical Monthly, 59*, 521–531.

Quine, W. V. (1955). A way to simplify truth functions. *American Mathematical Monthly, 62*, 627–630.

Quinlan, J. R. (1979). Discovering rules from large collections of examples: a case study. In Michie, D. (Ed.), *Expert systems in the micro-electronic age*, (pp. 168–201)., Edinburgh. Edinburgh University Press.

Quinlan, J. R. (1986a). The effect of noise on concept learning. In Kodratoff, Y. & Michalski, R. (Eds.), *Machine Learning: An artificial intelligence approach*, volume 2, (pp. 148–166)., Los Altos, CA. Morgan Kaufmann.

Quinlan, J. R. (1986b). Induction of decision trees. *Machine Learning, 1*, 81–106.

Quinlan, J. R. (1987). Simplifying decision trees. *International Journal of Man-Machine Studies, 27*, 221–234.

Quinlan, J. R. (1989). Unknown attribute values in induction. In *Proceedings of the Sixth International Workshop on Machine Learning*, (pp. 164–168)., Palo Alto, CA. Morgan Kaufmann.

Quinlan, J. R. (1990a). Learning logical definitions from relations. *Machine Learning*, *5*, 239–266.

Quinlan, J. R. (1990b). Probabilistic decision trees. In Kodratoff, Y. & Michalski, R. (Eds.), *Machine Learning: An artificial intelligence approach*, volume 3, (pp. 140–152)., San Mateo, CA. Morgan Kaufmann.

Quinlan, J. R. (1992). *C4.5: programs for empirical learning,* (In press), (pp. 35–39). Morgan Kaufmann.

Shannon, C. E. & Weaver, W. (1948). *The Mathematical Theory of Communications.* Urbana: University of Illinois Press (Book published 1964).

Sober, E. (1975). *Simplicity*, chapter 1. Oxford: Clarendon Press.

Webb, G. I. (1991). Einstein — an interactive inductive knowledge acquisition tool. In *Proceedings of the Sixth Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, volume 2, (pp. 36.1–36.16).

Wegener, I. (1987). *The complexity of Boolean functions*, chapter 2. Great Britain: Wiley & Sons.