To the Graduate Council:

I am submitting herewith a thesis written by Lloyd Fredrick Arrowood III entitled "Oriented consensus methods for internally disjunctive production rules." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

J. R. B. Cockett

J. R. B. Cockett, Major Professor

We have read this thesis
and recommend its acceptance:

Jean R. S. Blair

William J. McCalla

Accepted for the Council:

Vice Provost
and Dean of The Graduate School

ORIENTED CONSENSUS METHODS FOR INTERNALLY DISJUNCTIVE PRODUCTION RULES

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Lloyd Fredrick Arrowood III

August 1989

DEDICATION

To my parents, Lloyd F. and Wanda L. Arrowood

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor Dr. J. Robin B. Cockett for his guidance of this thesis. His patience and assistance throughout my research has been greatly appreciated, and his ideas and comments have contributed materially. I would also like to thank Dr. Jean R. S. Blair and Dr. William J. McClain for serving on my committee. Their constructive criticism greatly improved the quality of this thesis. Whatever errors remain, however, are solely my responsiblity.

Several other individuals have assisted me during the courrse of this research. Dr. Michael Marchioni of East Tennessee State University made several helpful suggestions on the organization on this material. William Capshaw produced the masterfully drawn figures throughout this thesis. Lastly, I would like to thank my wife Sandra and daughter Leslie for their understanding and support.

ABSTRACT

Several consensus methods have been studied and modified to process internally disjunctive production rules. Oriented consensus methods have been developed which retain information concerning the intended use of production rules. The behavior of oriented consensus is studied, and an antecedent-to-antecedent consensus is defined to be a widening. Widenings can be distributed throughout a proof such that no widening occurs before any other consensus. In fact, it is shown that the widenings themselves can be distributed when the consensus is at distinct attributes. We examine the effects of preventing all widenings from occurring during the generation of prime rules.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Production rules are the most widely recognized form of knowledge representation. However, present techniques for codifying knowledge as production rules often fail to produce a concise and complete description of the problem domain. Inaccurate information can cause a knowledge-based system to arrive at erroneous conclusions with potentially devastating results. Even when the set of rules is correct, poorly represented knowledge can complicate inference processes or handicap explanatory capabilities. Production rules should be concise yet expressive. Nonsystematic methods of validating and refining expert knowledge may prove insufficient for a large set of production rules.

We address such problems in this research by examining the analogous problem of minimizing Boolean expressions. In the traditional approach to Boolean minimization, the initial set of expressions for a function are used to generate the prime implicants of that function and a minimal cover of those prime implicants is then found. Similar methods can be devised to manipulate internally disjunctive production rules, which are based upon a concise knowledge representation formulated by Michalski [32,33,34]. Existing methods of generating prime implicants are studied and then modified to accept internally disjunctive production rules and their dual representation, subproducts. After the prime rules have been generated, a covering algorithm finds a minimal cover for the set of primes.

1.1 Approach

We can transform production rules into a set of prime rules which can then be used to generate a minimal set of rules. Constructing knowledge bases, however, is an iterative process which encodes knowledge into production rules, tests the rules for accuracy and consistency, and refines them to capture additional information. Therefore, the set of prime rules must be constructed incrementally as each new rule is added. This permits domain experts to examine the ramifications of each new rule and is one method of identifying invalid production rules before a system is implemented. A domain expert can represent his knowledge as production rules, using an interactive system to display the consequences of a particular rule as it is introduced into the knowledge base. This makes it easier to identify conflicting rules and eliminate redundant information.

An interactive system has been developed in PROLOG[1] to generate incrementally prime rules and to produce a minimal representation of these primes. During a session, the user sees the effects of new rules upon an existing knowledge base. In addition, some primes may express logical relationships more succinctly than the domain expert. The consensus method for production rules produces a new rule. If the new rule is not subsumed, it is prime and should added to the set of prime rules. Conversely, if the new prime subsumes any existing rules, the subsumed rules are no longer prime and can be removed from the set of prime rules.

Others have studied various forms of this problem. The consequence-finding problem formulated by Lee [24] is as follows: Given axioms $A_1, A_2 \ldots A_n$, find an axiom A_m such that it follows from $A_1, A_2 \ldots A_n$. Under this formulation, if A_m is the empty clause, consequence-finding reduces to refutation proof-finding. Minicozzi and Reiter [36] studied the completeness of various linear resolution strategies for consequence finding. In particular, they showed that linear resolution with merging and subsumption is complete for

[1] A programming language developed by the Groupe Intelligence Artificielle of the Faculté des Sciences de LUMINY, Marseille, France.

consequence-finding. The difference between this research and previous efforts is that this investigation concentrates upon internally disjunctive production rules. In doing so, several issues arise which have not been considered previously.

1.2 Organization

Attribute systems are defined in Chapter 2. These are the models for internally disjunctive production rules. After formalizing many of the notions behind internally disjunctive rules and their prime representations, we extend the basic model by allowing half-open intervals to serve as codomains for an attribute. The latter portion of the chapter describes the actual algebraic operations which are performed for discrete or continuous domains.

Chapter 3 discusses *consensus* methods for generating the prime implicants (implicates) for a set of conjunctive (disjunctive) Boolean expressions. Using results obtained in Chapter 2, these methods can be generalized so that they generate the *maximal subproducts* of an attribute system. Literals in the Boolean case can be represented in an attribute system which can take one of two values, 0 or 1. The maximal subproducts of an attribute system are analogous to the prime implicants of a set of Boolean expressions.

Also in Chapter 3, those approaches to consensus are discussed which are particularly well suited for handling internally disjunctive rules: The Quine-McCluskey algorithm, iterative consensus, and generalized consensus. The Quine-McCluskey algorithm manipulates Boolean expressions in canonical form, i.e., all the variables in the function are present in every expression. Iterative consensus is shown to perform consensus upon expressions which are not in canonical form and to utilize explicit subsumption when eliminating those implicants which are not prime. Generalized consensus (or Tison's method) presents a more efficient form of consensus as the number of iterations is based upon the number of complementary variables, i.e., both the positive and negative form of a variable are in a function. Muroga [37] has published empirical results which support claims that generalized consensus is more efficient for large minimization functions. Kean and Tsiknis [22] give an

incremental version of this algorithm which generates the prime implicants for a set of expressions and recalculates the set of prime implicants when a new expression is added. The incremental algorithm is then bounded by the number of complementary variables in the new expression. This last method appears most promising in generating the set of prime implicants for a constantly changing, yet monotonic, set of expressions.

Chapter 4 concentrates upon production rules and presents a consensus technique which retains the original form of the rules. Classical resolution [47] must convert production rules to their clausal form, losing valuable information concerning the original intended use of the rule. Other techniques [1,7,8,38,53] have been developed to manipulate expressions which are not in clausal form. Unlike these methods, the new consensus technique divides rules into two parts: the logical rule and the form that the rule takes. Also introduced is a convention to make the new rule more concise than its parents.

In Chapter 5, the covering algorithm is shown to be a sophisticated depth-first search which locates essential rules at each level of the search. The user can direct the search by indicating whether to remove a rule from consideration or to include it in a minimal cover. When a rule is added which violates the minimality restrictions, the search fails, backtracks to a preceding level and continues the search. Users can elect to produce minimal covers interactively until all covers have been found, whereupon the system announces that no additional covers can be obtained. Searching for covers can be interactive, allowing the user to force or remove rules from consideration in a cover, provided that minimality and covering constraints are not violated. We do not consider the effects of orientation upon set covering for production rules.

The thesis concludes by assessing the primary contributions of this research, suggests possible applications of oriented consensus and proposes additional research topics. Appendices A and B are program traces showing unoriented consensus methods and their effects upon a set of production rules. Appendix C shows covering techniques as they are applied

to a set of prime rules. While we have defined oriented consensus and studied its behavior, we have not implemented these ideas.

Chapter 2

Attribute systems

Marek and Pawlak [28] and Pawlak [41] provide detailed descriptions of attribute systems. The following definitions have been simplified for purposes of this discussion.

2.1 Objects in attribute systems

Definition 2.1 *An* **attribute system**, $\mathcal{A}$, *over the attribute sets* $A_1,\ldots,A_n$ *is a pair,* $A = (X,(a_1 \ldots a_n))$, *where* X *is the* **set of objects** *and each* a_i *is an attribute map such that*

$$a_i : X \longrightarrow A_i.$$

For each $x \in X$, *we say that* $x \longmapsto a_i(x)$.

Elements of the object set are referred to as objects of an attribute system. If x is an object, the a_i attribute of x is $a_i(x) \in A_i$. Attribute sets are not necessarily distinct and can be repeated throughout an attribute system.

One way of representing an attribute system is as a table with rows corresponding to the objects and with columns corresponding to the n-tuple of attributes associated with each object.

Example

Let the object set X be the set of chemical compounds $\{Mg, MgO, C_2, CO_2, O_2, H_2O\}$ and the attribute sets be respectively:

A_1 =*metallic?*= $\{yes, no\}$

A_2 =*oxide?*= $\{yes, no\}$

A_3 =*physical state?*= $\{solid, liquid, gas\}$

To each of the six chemical compounds one can associate several descriptors. These associations may be regarded as the attribute maps, *metallic?*, *oxide?*, and *physical state?*, respectively. The attribute system which results can then be specified by Table 2.1.

Table 2.1: An attribute system in tabular form.

chemical	metallic?	oxide?	physical state
Mg	yes	no	solid
MgO	yes	yes	solid
C_2	no	no	solid
O_2	no	yes	gas
H_2O	no	yes	liquid
CO_2	no	yes	gas

Note that the pair of objects $\{O_2, CO_2\}$ have identical attributes in this system. Each attribute map a_i induces an equivalence relation denoted by $E(a_i)$. The intersection of all such equivalence relations is denoted as $E(A)$.

Table 2.2: The equivalence classes E(*metallic?*), E(*oxide?*) and E(*physical state?*).

Mg	MgO
C_2	O_2
H_2O	CO_2

Mg	MgO
C_2	O_2
H_2O	CO_2

MgO	Mg
C_2	O_2
H_2O	CO_2

Table 2.3: The intersection of all equivalence classes $E(\mathbf{A})$.

Mg	MgO
C_2	O_2
H_2O	CO_2

Definition 2.2 *Two objects are* **indistinguishable** *if and only if they are in the same equivalence class of* $E(A)$.

Indistinguishability can also be described by a map of attribute systems.

Definition 2.3 *Let* $A_1 = (X_1, (a_{11} \ldots a_{1n}))$ *and* $A_2 = (X_2, (a_{21} \ldots a_{2n}))$ *be attribute systems over* $(A_1 \ldots A_n)$ *and* $f : X_1 \to X_2$ *a map, if the following equality holds for each* $i = 1 \ldots n$*:*

$a_i(x_i) = a_i(f(x_i)), \forall x_i \in X_i$ *then* f *is a* **map of attribute systems.**

For any collection of attribute sets, $(A_1, \ldots, A_n)$ there is a canonical form of an attribute system, called the *final attribute system.*

Definition 2.4 *The* **final attribute system** *has the object set,* $A_1 \times \ldots \times A_n$*, and the attribute maps,* $p_1, \ldots, p_n$*, where the map* p_i *is the ith projection from the Cartesian product* $A_1 \times \cdots \times A_n$*.*

There is a unique map from any attribute system over $\{A_1, \ldots, A_n\}$ to the final attribute system. If $\mathcal{A} = (X, (a_1 \ldots a_n))$ is an attribute system over $A_1 \times \cdots \times A_n$ then by definition of a product there is a unique map $\langle a_i, \ldots, a_n \rangle : X \to A_1 \times \cdots \times A_n \; ; x \mapsto \langle a_i(x), \ldots, a_n(x) \rangle$ such that $a_i = p_i(\langle a_1, \ldots, a_n \rangle)$ which makes it a map of attribute systems.

Lemma 2.1 *Every attribute system has a unique map to the final attribute system. Furthermore two objects in an attribute system are indistinguishable if and only if they have the same object type.*

The object set of the final attribute system is referred to as the set of object types; consequently, every object in an attribute system has a unique object type associated with it through the unique map of attribute systems.

An attribute system can be represented as a density function on the Cartesian product of attributes. The attribute signature is a subset of this product where the density is non-zero. Within this signature, relationships or dependencies among attributes can be identified.

2.2 Attribute signatures and subproducts

Definition 2.5

1. *A subset S of $A_1 \times \cdots \times A_n$ is called an* **attribute signature** *for attribute systems over $(A_1, \ldots A_n)$.*

2. *An attribute system $\mathcal{A} = \{X, (a, \ldots a_n)\}$ over $(A_1, \ldots, A_n)$* **satisfies** *the signature S if the unique map of attribute systems:*

$$\langle a_1, \ldots, a_n \rangle : X \longrightarrow A_1 \times \cdots \times A_n; x \longmapsto \langle a_1(x), \ldots, a_n(x) \rangle$$

factors through the subset S.

Thus, for every object x of A, its object type $\langle a_1(x), \ldots, a_n(x) \rangle$ is a member of S. In the Cartesian product of attributes a signature represents the volume in which the object types of any attribute systems which satisfy the signature must lie.

Given any attribute system $\{X, (a_1 \ldots a_n)\}$, there is a signature which it induces, namely the image of the object type map,

$$\Im(\langle a_1, \ldots, a_n \rangle) = \{\langle a_1(x), \ldots, a_n(x) \rangle \mid x \in X\}.$$

An attribute system satisfies a signature S if and only if $\Im(\langle a_1, \ldots, a_n \rangle)$ is contained in S. This induced signature provides the basis for inferring those properties satisfied by the attributes.

If $A_1' \subseteq A_1$ then $A_1' \times \cdots \times A_n \subseteq A_1 \times \cdots \times A_n$ is a *subproduct* and defines a signature. In fact, every signature can be expressed as a union of subproducts. By expanding subproducts until any further expansion would cover space not included in the signature, a *maximal subproduct* can be obtained.

Definition 2.6

1. *A* **subproduct** *of $A_1 \times \cdots \times A_n$ is a product $A_1' \times \cdots \times A_n'$ where each $A_i' \subseteq A_i$.*

2. *Let S be an attribute signature over $A_1 \times \cdots \times A_n$. A* **maximal subproduct** *satisfying the signature is a subproduct P of $A_1 \times \cdots \times A_n$ such that P is contained in S and there is no strictly larger subproduct contained in S.*

Maximal subproducts can be used for describing signatures.

Proposition 2.1 *Every subset S of a product $A_1 \times \cdots \times A_n$ can be written as the union of its maximal subproducts.*

The proof of this can be obtained using the following result:

Lemma 2.2 *Every subproduct contained in S is contained in a maximal subproduct.*

Using Lemma 2.2, Proposition 2.1 can be proven:

Proof of Proposition 2.1

Every element of S is a trivial subproduct

$$\{a_1\} \times \cdots \times \{a_n\} = \{\langle a_1 \ldots a_n \rangle\}.$$

Thus, every element is contained in a maximal subproduct, and the union of maximal subproducts must be S. □

Proof of Lemma 2.2

Let $A'_1 \times \cdots \times A'_n$ be a subproduct of finite sets. If there is a larger subproduct contained in S then there is an A''_i, $1 \leq i \leq n$, such that $A'_i \subset A''_i$ and $A'_1 \times \cdots \times A''_i \times \cdots \times A'_n$ is contained in S. The *ith* component of the subproduct can be enlarged until any further expansion would cause the containment of the subproduct in S to fail. If each component is successively enlarged then the result will be a maximal subproduct, since enlarging any component would cause the containment to fail.

For infinite sets, the following observation is made. Since the union of an ascending chain of subproducts contained in S is also a subproduct contained in S, Zorn's lemma (See [15]) may be applied. Therefore, A_i has at least one maximal element. □

The process of enlarging a subproduct is not unique; as evidenced by Figure 2.1, there may be many maximal subproducts which contain a given subproduct. However, at least one maximal subproduct will contain any given subproduct.

Figure 2.1: A subproduct contained in several maximal subproducts

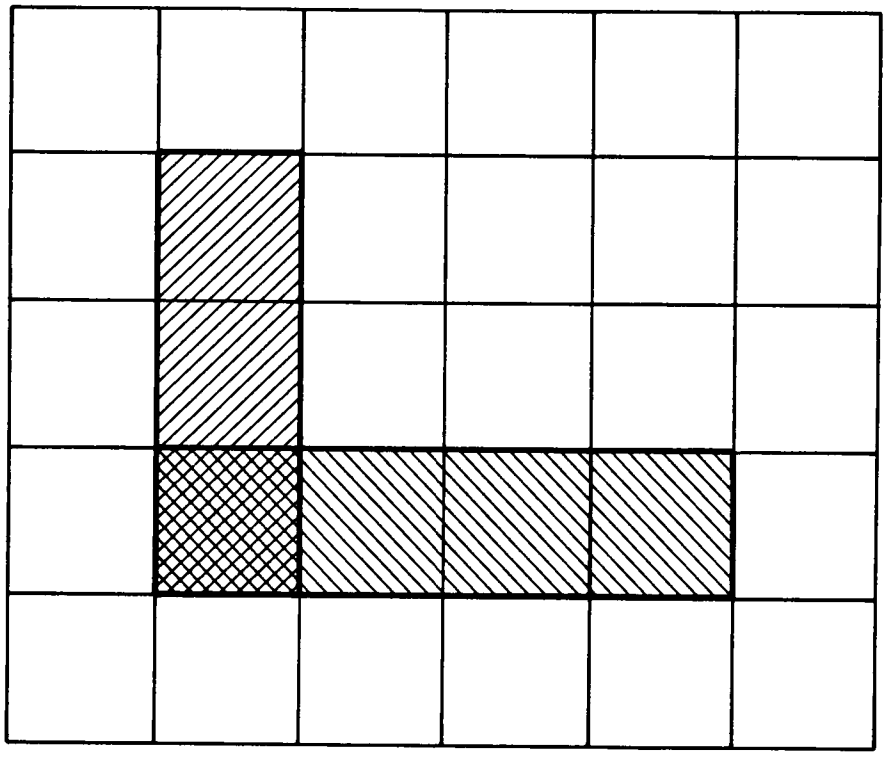

Figure 2.2: A subproduct contained in two maximal subproducts.

It is useful to introduce a notation for subproducts before the algorithms to manipulate subproducts are discussed. An attribute system $\mathcal{A} = (X, (a_1, \ldots, a_n))$ satisfies the *attribute restriction* $a_i \diamond A_i'$ where $A_i' \subseteq A_i$ if $a_i(x) \in A_i'$ for all $x \in X$. Attribute restrictions can be combined both conjunctively and disjunctively. Notice that a conjunct of attribute restrictions is equivalent to a subproduct in the set of object types. By indicating the attribute map in the expression for a subproduct, $A_1' \times \cdots \times A_n'$ is represented as $a_1 \diamond A_1' \wedge \cdots \wedge a_n \diamond A_n'$ where each term in the disjunction $a_i \diamond A_i'$ denotes

$\{x \in A_1 \times \cdots \times A_n \mid p_1(x) = y_1 \vee \cdots \vee p_i(x) = y_m\}$ where $y_j \in A_j$ and $p_i(x) \in A_i$.

The set A_i' is called an *internal disjunction*; the attribute a_i may map into any of the values in the attribute set.

By including the attribute map in the representation of a subproduct, the ordering of attribute restrictions in an expression can be arbitrary. Also any term for which $A_i = A_i'$ (i.e., all elements in the attribute set for that coordinate are possible) can be omitted from the expression; these are *unrestricted* attributes. The aim is to use maximal subproducts to derive succinct expressions for attribute signatures.

2.3 Set operations on intervals and ordered lists

By ordering the codomains of attributes and designing set operators which exploit ordering, significant savings can be realized. Intervals lend themselves to ordered operations, and symbolic values can also be manipulated by alphabetic ordering. The three operators and recursive procedures for implementing them are described below:

Operations on ordered lists

Intersect(L_1,L_2,L_3)

This procedure intersects the ordered lists L_1 and L_2, creating L_3. It successively compares elements of L_1 and L_2. For any comparison between two elements, $x \in L_1$ and $y \in L_2$, three outcomes are possible:

if $x > y$ then return Intersect($[x|L_1], L_2, L_3$).

if $x = y$ then return Intersect($L_1, L_2, [x|L_3]$).

if $x < y$ then return Intersect($L_1, [y|L_2], L_3$).

If there is no intersection between L_1 and L_2, the entire procedure fails, provoking PROLOG's backtracking mechanism.

Union(L_1,L_2,L_3)

This procedure merges L_1 and L_2, creating L_3. It also compares elements of L_1 and L_2. For any comparison between two elements, $x \in L_1$ and $y \in L_2$, three outcomes are possible:

if $x > y$ then return Union($L_1, [y|L_2], [x|L_3]$).

if $x = y$ then return Union($L_1, L_2, [x = y|L_3]$).

if $x < y$ then return Union($[x|L_1], L_2, [y|L_3]$).

When all the elements of one list have been exhausted, the procedure halts after appending the other list to L_3.

Minus(L_1,L_2,L_3)

This procedure subtracts all elements of L_1 from L_2, giving L_3. For any comparison between $x \in L_1$ and $y \in L_2$, three outcomes are possible:

if $x > y$ then return Minus($L_1, L_2, [x|L_3]$).

if $x = y$ then return Minus(L_1, L_2, L_3).

if $x < y$ then return Minus($L_1, L_2, [y|L_3]$).

When all the elements of L_1 or L_2 have been exhausted, the procedure halts. If L_1 is not empty, the procedure appends L_1 to L_3 and returns L_3.

Operations on intervals

Half-open intervals are allowed as codomains and may consist of only an upper or lower bound. For instance, $[\inf, b]$ indicates that $a \leq b$; similarly, $[a, \inf]$ indicates that any $b \geq a$. Set operations for intervals differ from their counterparts for ordered lists in that intervals often overlap. Consequently, two sets of values must be examined by each operation—both the beginning and ending values.

Units of measure are associated with each set of intervals. The set operations do not act upon attributes with differing units of measure. For example, kilos and grams would never be intersected or merged.

Intersect intervals(I_1,I_2,I_3)

This procedure intersects two sets of intervals, creating a third. Each interval $[a_1, b_1] \in I_1$ is checked against successive intervals in I_2 until it is intersected or all intervals in I_2 have been be examined. First, the intervals are checked to see if they are disjoint:

Given $[a_1, b_1] \in I_1$ and $[a_2, b_2] \in I_2$

if $b_1 < a_2$ then return Intersect intervals$(I_1, [[a_2, b_2]|I_2, I_3)$.

if $b_2 < a_1$ then return Intersect intervals$([[a_1, b_1]|I_1, I_2, I_3)$.

If the two intervals overlap, they are merged to form a third interval $[a, b]$. In this case, the following comparisons are required:

if $a_2 < b_2$ then $a = a_2$

if $a_1 < b_1$ then $b = b_1$

otherwise $b = a_1$.

otherwise $a_1 \geq a_2, a = b_2$

if $a_1 < b_1$ then $b = b_1$

otherwise $b = a_1$.

if $b_1 < b_2$ then $b = b_1$; otherwise $b_1 \geq b_2, b = b_2$.

Then the procedure recurs:

return Intersect intervals$(I_1, I_2, [[a, b]|I_3])$.

When all the elements of one list have been exhausted, the procedure halts, returning I_3.

Union intervals(I_1, I_2, I_3)

This procedure merges I_1 and I_2, creating I_3. Each $[a_1, b_1] \in I_1$ is successively compared against intervals in I_2 until it is merged or appended to the end of I_2. This operation first determines whether the intervals overlap.

Given $[a_1, b_1] \in I_1$ and $[a_2, b_2] \in I_2$

if $a_2 < b_1$ then return Union intervals$(I_1, I_2, [[a_1, b_1], [a_2, b_2]|I_3])$.

if $b_2 < a_1$ then return Union intervals$(I_1, I_2, [[a_2, b_2], [a_1, b_1]|I_3])$.

Again, if the intervals overlap, they are merged to form a third interval $[a, b]$. In this case, the following comparisons are required:

if $a_1 < a_2$ then $a = a_1$; otherwise $a_1 \geq a_2, a = a_2$.

if $b_1 < b_2$ then $b = b_1$; otherwise $b_1 \geq b_2, b = b_2$.

Then the procedure is recurs:

return Union intervals$(I_1, I_2, [a, b]|I_3])$.

When all the elements of one list have been exhausted, the procedure halts after appending the other list to I_3.

Minus intervals(I_1,I_2,I_3)

This procedure takes a set of intervals I_1 and subtracts them from another set I_2 to form I_3. Each interval $[a_1, b_1] \in I_1$ is first negated:

$[\text{inf}, b_1] \Rightarrow [b_1, \text{inf}]$,

$[a_1, b_1] \Rightarrow [inf, a_1], [b_1, \text{inf}]$, and

$[a_1, \text{inf}] \Rightarrow [\text{inf}, a_1]$.

The negation of $[a_1, b_1] \in I_1$ is then intersected with I_2. This procedure is repeated for all intervals in I_1.

$x > y$ call Minus intervals$(I_1, I_2, [i|I_3])$ where i is a set of intervals.

When all the elements of I_1 have been exhausted, the procedure halts, returning I_3; when all the elements of I_2 have been exhausted, the procedure appends the remaining elements of I_1 to I_3.

Chapter 3

Generalized prime implicants

In Chapter 2, the necessary definitions of attribute systems and their notation were introduced. This chapter describes three techniques for minimizing Boolean expressions and discusses modifications required to process maximal subproducts. Among the earliest algorithms to minimize switching functions were those of Quine [43,44] and McCluskey [29]. Other techniques have been described in [9,18,21,39,40,46,48,52,55,56]. As digital circuitry has become more complex, newer techniques have dealt with both exact minimization and heuristics for approximate solutions [20,37]. In addition, increased attention has afforded multivalued logics and their application in the design of programmable logic arrays (PLAs) [4,10,16,49]. This investigation has not considered these newer techniques as a basis for more sophisticated consensus mthods. Further research may consider modifying these approaches to develop additional consensus methods for internally disjunctive production rules.

In Section 3.1, the problem of finding prime implicants is discussed, and the analogous case for maximal subproducts is addressed. Section 3.2 describes the Quine-McCluskey algorithm and introduces an example problem which is used extensively throughout this chapter. Section 3.3 describes iterative consensus and explains its utility. In Section 3.4, another consensus method, Tison's method, which is not iterative in the same sense as the two earlier algorithms is discussed and an incremental version devised by Kean and Tsiknis is described. In this chapter, we show that these methods can be extended to generate maximal subproducts.

3.1 Prime implicant generation

A well-studied problem related to the generation of maximal subproducts is the minimization of switching functions. Both are disjunctions of conjunctive expressions. A switching function is a map $f : 2^n \longrightarrow 2$ where $2 = \{0,1\}$ determined by the set $S = \{(\alpha_1,\ldots,\alpha_n)|f(\alpha_1,\ldots,\alpha_n) = 1\}$. Thus to determine f it suffices to describe S or $\neg S$. A convenient description of S is as a union of d-cells

$$D_{(\delta,r)} = \{(\alpha_1,\ldots,\alpha_n)|\alpha_{r_1} = \delta_1 \wedge \cdots \wedge \alpha_{r_{n-d}} = \delta_{n-d}\}$$

where $r : \mathbf{n-d} \longrightarrow n$. These d-cells are often called *implicants*. We need only consider the d-cells contained in S which are not contained in another larger d-cell contained in S. The maximal d-cells contained in S are called the *prime implicants* of f. Using de Morgan's laws, S can be described alternately as an intersection of the negation of d-cells. The negations of the d-cells containing S are often called *implicates* and the maximal d-cells contained in $\neg S$ are the *prime implicates*. Both forms are important in representing functions and minimizing circuits.

A similar technique can be used to represent functions

$$f : A_1 \times \cdots \times A_n \longrightarrow 2$$

where $A_1,\ldots,A_n$ are sets of attribute values. Again f can be described in terms of

$$S = \{(a_1,\ldots,a_n)|f(a_1,\ldots,a_n) = 1\}$$

or $\neg S$. Clearly S is a signature. Instead of using maximal d-cells to describe S we shall use *maximal subproducts*. A d-subproduct $\tilde{P} = (P_1,\ldots,P_{n-d})$

$$D_{\tilde{P},r} = \{(\alpha_1,\ldots,\alpha_n)|\alpha_{r_1} \in P_1 \wedge \cdots \wedge \alpha_{r_{n-d}} \in P_{n-d}\}$$

where $P_i \subseteq A_{r_1}$. A maximal subproduct of S is one which is contained in S but not contained in a larger subproduct which is contained in S.

These subproducts are useful for giving succinct conjunctive descriptions of S and thus of f. Two expressions *consense* when they are used to create a new expression. The original expressions are then referred to as *parent* clauses. When a more succinct expression implies another, it is said to *subsume* the other expression. To obtain the union of maximal subproducts from a description of S it suffices to consense subproducts and remove subsumed expressions until no new subproduct is obtained. The problem of identifying the set of maximal subproducts for S is the general case of determining the prime implicants of a switching function.

The computational requirements of generating the prime implicants increase exponentially with the number of variables and clauses in the expressions. McMullen and Shearer [31] show that any Boolean function f which can be expressed by a sum of conjunctive expressions (*sum of product form* in Boolean terminology) using m product terms can contain possibly $2^m - 1$ implicants. It has also been shown that some functions generate many more prime implicants than are needed to produce the minimal representations [11,35]. In fact, the number of prime implicants for one class of n-variable functions is proportional to $\frac{3^n}{n}$ [13].

3.2 The Quine-McCluskey algorithm

The method originally developed by Quine [43] becomes computationally expensive as the number of variables or clauses in the given canonical form increases. McCluskey [29] modified this procedure by aggregating the clauses comprising the initial canonical form, thereby reducing the number of comparisons. This technique arranges all trivial subproducts (called *minterms* in Boolean terminology) into groups, such that all subproducts in the same group have the same number of 1's in their binary representation. It begins with the least number of 1's and continues with groups of increasing numbers of 1's. The number of 1's in a term is called the index of that term. The following description of the Quine-McCluskey algorithm is given by Kohavi [23]:

Algorithm 3.1 (The Quine-McCluskey algorithm)

Arrange all minterms in groups, such that all terms have the same number of 1's in their binary representations. Start with the least number of 1's and continue with groups of increasing numbers of 1's. The number of 1's in a term is called an index.

Compare every term of the lowest group with each term in the successive group; whenever possible, combine the two by means of the combining theorem $Aa + A\overline{a} = A$. *Repeat this by comparing each term in a group of index* i *with every term in the index* $i + 1$, *until all possible applications of the combining theorem have been exhausted.*

The terms generated in Step 2 are now compared in the same fashion: a new term is generated by combining two terms which differ by only a single 1 and whose dashes are in the same position. This process continues until no further combinations are possible. The remaining terms constitute the set of prime implicants.

Two terms from adjacent groups are combinable if their binary representation differ by a single digit in the same position; the combined term consists of the original fixed representation with the different digit replaced by a dash (—). A check mark ($\surd$) is placed next to every term which has been combined with at least one term (Note that each term may be combined with several terms, but only a single $\surd$ is required. This technique for restricting the number of consensus operations cannot be generalized for the set-theoretic case, as we intend to do here. To see how Algorithm 3.1 can be used on subproducts, we define a *trivial subproduct* to be subproduct with all product terms in which all product terms are singleton sets.

A trivial subproduct is analogous to a minterm in the canonical form of a Boolean expression. Any nontrivial subproduct must be reformulated into the appropriate number of trivial subproducts. This is necessary to generate the maximal subproducts by the Quine-McCluskey algorithm. The following example illustrates why trivial subproducts are

required by this variant of the Quine-McCluskey algorithm and how inefficient this approach can be.

Example 3.1 *We are given two subproducts*

$$\{1,2\} \times \{1,3\} \times \{2\}$$

$$\{1,3\} \times \{1,2\} \times \{2\},$$

and would like to generate the set of maximal subproducts. The Quine-McCluskey algorithm cannot consense these subproducts because they differ at two product terms. Reformulating these subproducts as a set of trivial subproducts yields

$$\{1\} \times \{1\} \times \{2\}$$

$$\{1\} \times \{3\} \times \{2\}$$

$$\{2\} \times \{1\} \times \{2\}$$

$$\{2\} \times \{3\} \times \{2\}$$

$$\{1\} \times \{2\} \times \{2\}$$

$$\{3\} \times \{1\} \times \{2\}$$

$$\{3\} \times \{2\} \times \{2\}$$

After several iterations, the maximal subproducts are generated.

$$\{1,2\} \times \{1,3\} \times \{2\},$$

$$\{1,3\} \times \{1,2\} \times \{2\},$$

$$\{1,2,3\} \times \{1\} \times \{2\},$$

$$\{1\} \times \{1,2,3\} \times \{2\}.$$

In Section 3.3 an algorithm for generating the set of maximal subproducts from subproducts in nontrivial form will be presented. Now we present the modified Quine-McCluskey algorithm which accepts trivial subproducts as the initial set of expressions.

Algorithm 3.2 (The Quine-McCluskey algorithm for subproducts) *Compute the maximal Boolean subproducts of S as a signature over $A_1, \ldots, A_n$ where each A_i is a finite set*

from the elements of S. Let S_0 be the initial set of trivial subproducts and $U = \{\}$ be the initial set of unmarked subproducts. The algorithm returns the set of prime implicants PI, the sum of all unmarked subproducts at the final iteration.

```
PI ⟵ GENERATE SUBPRODUCTS(S, U)
n = |S|; i = 0; j = 0; k = 0;
if S = {} or S = {s} then return S + U
      else
         do i = 1, n
            do j = i + 1, n
               X:=S[i];
               Y:=S[j];
               (v, Z) =CONSENSE(X, Y);
               if v = TRUE
                  then
                     {X:=MARK(X);
                     Y:=MARK(Y);
                     S'[k] = Z;
                     k = k + 1}
         U' =COLLECT UNMARKED SUBPRODUCTS(S', U)
         return (GENERATE SUBPRODUCTS(S', U'))
```

CONSENSE combines two terms if their binary representations differ by a single digit. The resulting term consists of those digits which are identical and the internal disjunction $\{0,1\}$ at the complementary digit. If additional digits are complementary, no consensus is performed.

```
CONSENSE(X, Y)
c = |X|; i = 0;
do while (i ≤ c)
   x:=X[i];
   y:=Y[i];
   if x = ȳ then do while (i ≤ c)
      x:=X[i];
      y:=Y[i];
      if x = ȳ
         then return (F, {});
      return (T, {0,1} ∧ X − x);
      i = i + 1;
   then return (F, {});
```

After all consensus operations have been attempted, the subproducts are checked to see which have not been consensed. Consensed terms are subsumed by their products. All

other terms are used at the next level.

```
COLLECT UNMARKED SUBPRODUCTS(S)
U = {}; n = |S|; i = 0; j = 0;
do while i ≤ n
    X:=S[i];
    if ¬MARKED(X) then { U[j] = X;
        j = j + 1 }
    i = i + 1;
return U
```

The algorithm successively enlarges the set of subproducts until they are maximal. Compare pairs of these subproducts down the list. If they differ in exactly one coordinate, and in that coordinate each subproduct contains exactly one value not present in the other, introduce the subproduct with the union set at that coordinate. Mark all the trivial subproducts which can be enlarged in this manner. Now continue to the next level using all unmarked or new subproducts.

The result of this consensus is a subproduct in which the consensed coordinate is the union of the original subsets. For each level the process of attaining the next level consists of performing all consensus operations and marking the enlarged subproducts in the current level. The procedure continues until no consensus is performed at a level. The set of all unmarked subproducts throughout all levels are then the maximal subproducts. The following example illustrates how the modified algorithm works; it will be used repeatedly throughout this chapter so that the reader can compare how other consensus methods perform against the same set of data.

Example 3.2 *Given the following elements of the Cartesian product*

$$\{\langle 1,1,1\rangle, \langle 1,2,1\rangle, \langle 1,3,2\rangle, \langle 1,1,3\rangle, \langle 1,2,3\rangle, \langle 1,3,3\rangle, \langle 2,2,1\rangle\}$$

find the maximal subproducts of this attribute signature. Table 3.1 shows the form that the minimization table takes.

$$\{1,2\} \times \{2\} \times \{1\},$$

$$\{1\} \times \{3\} \times \{2,3\},$$

Table 3.1: Maximal subproducts generated by the Quine-McCluskey algorithm.

List 1		List 2		List 3	
1	$\{1\} \times \{1\} \times \{1\}$	*1,2*	$\{1\} \times \{1,2\} \times \{1\}$	*1,2,4,5*	$\{1\} \times \{1,2\} \times \{1,3\}$
2	$\{1\} \times \{2\} \times \{1\}$	*1,4*	$\{1\} \times \{1\} \times \{1,3\}$	*4,5,6*	$\{1\} \times \{1,2,3\} \times \{3\}$
3	$\{1\} \times \{3\} \times \{2\}$	*2,5*	$\{1\} \times \{2\} \times \{1,3\}$		
4	$\{1\} \times \{1\} \times \{3\}$	*2,7*	$\{1,2\} \times \{2\} \times \{1\}$		
5	$\{1\} \times \{2\} \times \{3\}$	*3,6*	$\{1\} \times \{3\} \times \{2,3\}$		
6	$\{1\} \times \{3\} \times \{3\}$	*4,5*	$\{1\} \times \{1,2\} \times \{3\}$		
7	$\{2\} \times \{2\} \times \{1\}$	*4,6*	$\{1\} \times \{1,3\} \times \{3\}$		
		5,6	$\{1\} \times \{2,3\} \times \{3\}$		

$$\{1\} \times \{1,2\} \times \{1,3\},$$

$$\{1\} \times \{1,2,3\} \times \{3\}.$$

The algorithm generates the set of prime implicants, which overspecify the function. Chapter 5 discusses methods for obtaining minimal covers for a set of primes.

Note that consensi may not be unique if restrictions are not placed upon the consensus process. For example, the prime implicant

$$\{1\} \times \{1,2\} \times \{1,3\}$$

is produced by consensing either 1,4 and 2,5 or 1,2 and 4,5 in List 2 of Table 3.1.

Each level may be distinguished by the number of internal disjunctions. To increase the number of internal disjunctions, one must find a consensus. Suppose that a maximal subproduct is given; then it is easy to see how it may be broken down into subproducts with fewer and fewer internal disjunctions. These subproducts can then be consensed to form the original maximal subproduct. Clearly, every maximal subproduct is generated and these are the only subproducts which cannot be further enlarged by consensus.

Although this algorithm arrives at the set of maximal subproducts, it does so by incrementally building successively higher levels. In the process, numerous intermediate results may be generated and pruned in later stages of the operation. The next algorithm is designed to obtain the maximal subproducts at an earlier stage of the process.

3.3 Iterative consensus

Iterative consensus [44] is a generalization of the Quine-McCluskey algorithm which does not require that terms be expressed in canonical expressions. It use a technique called consensus.

Definition 3.1 *The conjunction $\phi\psi$ is called the* **consensus** *of $\alpha\phi$ and $\overline{\alpha}\psi$ provided that it does not contain a variable which is both positive and negative.*

We can now give the algorithm for iterative consensus.

Algorithm 3.3 (Iterative consensus) *Perform 1 and 2 until no further results are obtained:*

1. *Drop tautologies. If one of the clauses subsumes another, drop the subsumed clause.*
2. *Add, as an additional clause, the* **consensus** *of two clauses.*

Consensus does not apply if the result is subsumed by an existing clause; otherwise the procedure could oscillate between 1 and 2.

In Section 3.2 we defined trivial subproducts and mentioned an algorithm which does not require that the initial set of subproducts be in trivial form. Iterative consensus can be generalized to subproducts by merging the codomains of that attribute on which consensus is performed. All other attributes are intersected, with an empty intersection representing a nonvalid result. Since a rule without a restriction for an attribute allows that attribute to take any value, consensus at an attribute present in only one of the rules results in that attribute restriction being placed in the new rule.

We now present an algorithm based on iterative consensus which is guaranteed to produce the set of maximal subproducts in fewer consensus operations than the Quine-McCluskey algorithm. However, subsumption is inherent in the Quine-McCluskey algorithm while subsumption checking is necessary for iterative consensus. Therefore, iterative

consensus is not guaranteed to be more efficient than Quine-McCluskey, but is guaranteed to obtain the maximal subproducts at an earlier stage.

Algorithm 3.4 (iterative consensus for subproducts) *Compute the maximal subproducts given S_0 as the initial set of subproducts. The algorithm returns PI, the set of maximal subproducts. S is a vector which contains clauses. Each clause S_i is itself a vector whose entries correspond to pairs of attributes and their codomains for that clause. Since the expressions are not in expanded form, the consensus algorithm must determine whether two clauses share an attribute and the codomains for those attributes can be consensed.*

```
PI ⟵ GENERATE SUBPRODUCTS(S)
n = |S|; k = 0;
if S = {} or S = {s} then return S;
      else { S' = {};
         do i = 1, n
            do j = i + 1, n
               X:=S[i];
               Y:=S[j];
               Z =CONSENSE(X, Y);
               if ¬SUBSUMED(Z,S)
                  then { S:=SUBSUMES(Z, S);
                     S'[k] = Z;
                     k = k + 1 }
            i = i + 1;
            j = j + 1;
         return (GENERATE SUBPRODUCTS(S ∪ S'))}
```

This version of CONSENSE differs from earlier ones in that those variables not merged are intersected. In addition, the pair (attribute,value) is accessed by means of a dot operator. If the structure being manipulated is x then the attribute portion of the pair can be obtained by $x.att$. Similarly, the codomain of that attribute can be obtained by $x.val$.

```
CONSENSE(X, Y)
c = |X|; i = 0;
do while i ≤ c
   x:=X[i];
   j = 0;
   do while j ≤ c
      y:=Y[i];
      if x.att = y.att then {
```

if $x.val \not\subseteq y.val$, $y.val \not\subseteq x.val$ { **then** $X' = X - x$, $Y' = Y - y$
return (T, UNION(x, y)$\wedge$ INTERSECT(X', Y')) } }
else $i = i + 1$;

In this algorithm, the routines SUBSUMED and SUBSUMES replace COLLECT UNMARKED SUBPRODUCTS in the Quine-McCluskey algorithm. SUBSUMED takes X and determines whether any existing expressions subsume it. If X is not subsumed, it can be added to the set of subproducts.

```
SUBSUMED(X, S)
n = |S|; i = 0;
do while i ≤ n
    Let Y:=S[i];
    c = |X|; j = 0;
    do while j ≤ c
        x:=X[i];
        y:=Y[i];
        if x ⊆ y
            then i = i + 1;
            else return FALSE;
        return TRUE;
```

SUBSUMES determines whether X subsumes any existing expressions. If it does, those expressions are deleted from the set of subsumed. SUBSUMES must be performed after SUBSUMED to ensure completeness.

```
SUBSUMES(X, S)
n = |S|; i = 0;
do while i ≤ n
    Y:=S[i];
    c = |Y|; j = 0;
    do while j ≤ c
        x:=X[j];
        y:=Y[j];
        if y.val ⊆ x.val then j = j + 1;
            else return FALSE;
        S:=DELETE(Y);
    i = i + 1;
return S;
```

For each pair of subproducts

$$A_{11} \times \cdots \times A_{1n} \text{ and } A_{21} \times \cdots \times A_{2n}$$

and $1 \leq i \leq n$ define the following consensus operation to form the new subproduct

$$(A_{11} \wedge A_{21}) \times \cdots \times (A_{1i} \vee A_{2i}) \times \cdots \times (A_{1n} \wedge A_{2n}),$$

where $\wedge$ and $\vee$ are set intersection and union respectively. If non-canonical expressions are used, unspecified coordinates are logically equivalent to that attribute's codomain (i.e., an internal disjunction comprised of all possible values for that coordinate). Set intersection between this set and a restricted set of values yields the restricted set.

Note that if any coordinate in a new subproduct which is not the result of a UNION operation is empty, the entire subproduct is empty. If the result is contained in an existing subproduct, the new subproduct is subsumed. Conversely, if the new subproduct contains existing subproducts, the existing subproducts are subsumed. We need not consider consensus between subproducts which have proper restrictions on the same attribute and no disjoint restrictions on any other attribute.

Let there be a maximal subproduct P which is contained in S, the signature determined by the union of subproducts. If P is in the initial set of subproducts, each consensus results in a subproduct which is still contained in S so the subproduct is never subsumed. Otherwise, suppose we had started with trivial subproducts or at least subproducts which could not be further divided. At each iteration we would obtain the non-subsumed subproducts of the lower levels and the subproducts at the current one. The algorithm terminates when the current level produces no new subproducts. Starting with an arbitrary set of subproducts one observes that after each iteration there is a set of subproducts which subsumes those of the previous algorithm. Thus it must terminate with the maximal subproducts.

Table 3.2 shows the minimization table formed by iterative consensus using the set of trivial subproducts introduced in Example 3.2. The number of consensus operations is

dramatically reduced. For example, the consensi $2,5$ and $2,7$ are not generated because 2 is subsumed by consensus $1,2$.

Table 3.2: Maximal subproducts generated by iterative consensus (with trivial subproducts as input).

List 1		List 2		List 3	
1	$\{1\} \times \{1\} \times \{1\}$	1,2	$\{1\} \times \{1,2\} \times \{1\}$	1,2,4,5	$\{1\} \times \{1,2\} \times \{1,3\}$
2	$\{1\} \times \{2\} \times \{1\}$	3,6	$\{1\} \times \{3\} \times \{2,3\}$	3,4,5,6	$\{1\} \times \{1,2,3\} \times \{3\}$
3	$\{1\} \times \{3\} \times \{2\}$	4,5	$\{1\} \times \{1,2\} \times \{3\}$	1,2,7	$\{1,2\} \times \{2\} \times \{1\}$
4	$\{1\} \times \{1\} \times \{3\}$				
5	$\{1\} \times \{2\} \times \{3\}$				
6	$\{1\} \times \{3\} \times \{3\}$				
7	$\{2\} \times \{2\} \times \{1\}$				

If this example is reformulated so that the initial set of subproducts is expressed more succinctly, the advantage of the second algorithm becomes even more apparent. This is illustrated by Table 3.3.

Table 3.3: Maximal subproducts generated by iterative consensus (with nontrivial subproducts as input).

List 1		List 2	
1	$\{1\} \times \{1,2\} \times \{1\}$	1,3	$\{1\} \times \{1,2\} \times \{1,3\}$
2	$\{1\} \times \{3\} \times \{2\}$	1,4	$\{1,2\} \times \{2\} \times \{1\}$
3	$\{1\} \times \{1,2,3\} \times \{3\}$	2,3	$\{1\} \times \{3\} \times \{2,3\}$
4	$\{2\} \times \{2\} \times \{1\}$		

In this example all subproducts differing in a single coordinate have been expressed as a single subproduct. The complete set of maximal subproducts has been generated after a single iteration.

An incremental version of this method can be devised by generating the prime implicants P of a set of rules, and adding a new rule R. The new rule is then consensed with all primes in P. Forward and backward subsumption are applied. If any new primes are produced, they are consensed with all primes in P. This process proceeds until no consensus produces

a new rule. It is obvious that the proof for the general algorithm holds for the incremental approach.

3.4 Generalized consensus

Tison [56] describes a generalization of consensus in which all the consensus operations on one attribute are preformed together. After a brief discussion of Tison's original algorithm, slight modifications will be discussed which permit the processing of subproducts.

3.4.1 Tison's method

Tison [56] gives the following definition of generalized consensus:

Definition 3.2 *Given a set of expressions* Σ,

1. *A variable* x *is* **monoform** *if* $x \in \Sigma$ *and* $\overline{x} \notin \Sigma$.

2. *A variable* y *is* **biform** *if* $y, \overline{y} \in \Sigma$.

Let $M = \wedge_{i=1}^{n} M_i$ *and* M *be the n-order generalized consensus of* Σ *if and only if* $B_1 \vee \cdots \vee B_n = TRUE$ *holds irredundantly.*

Obviously a variable must be either monoform or biform. Tison states that the primary contibution of this generalization of consensus is that the consensus operations can be ordered so that the variables are eliminated one after the other. He then gives the following algorithm:

Algorithm 3.5 (Tison's method) *Let* L *be the set* $(A_1, A_2, \ldots, A_n)$ *which is called the* implicant list. *At the end of the procedure,* L *consists of the prime implicants of* Σ.

For each biform variable x *in* L,

1. *For every pair of clauses* A_i, $A_j \in L$, *add to* L *the consensus* $\mathrm{CS}(A_i, A_j, x)$ *if such a consensus exists.*

2. Delete from L all clauses Q such that there is a $Q' \in L$ which subsumes Q.

This method generates all and only the prime implicants of Σ. The proof can be found in [13] and [26].

3.4.2 Generalizing Tison's Method to subproducts

The key property which ensures the correctness of Tison's method is the following proof-theoretic identity which allows the consensus operations to be reordered.

Lemma 3.1 *Let R_1, R_2 and R_3 be subproducts and q, q' be attributes in those subproducts.*

$$\mathrm{CS}(\mathrm{CS}(R_1, R_2, q), R_3, q') = \mathrm{CS}(\mathrm{CS}(R_1, R_3, q'), \mathrm{CS}(R_2, R_3, q'), q).$$

Proof of Lemma 3.1 by cases

The following notation is used: R_i denotes the *ith* rule, R^x the value of attribute x in R, and R_i^x the value of x in R_i. P_1 is the consensus proof before distributing R_3 while P_2 is the consensus proof which results from the distribution. Only x and y need be considered as the remaining attributes are all intersected. The proof is obvious for $x = y$. For $x \neq y$

P_1		P_2	
x	y	x	y
$(R_1^x \cup R_2^x) \cap R_3^x$	$(R_1^y \cap R_2^y) \cup R_3^y$	$(R_1^x \cap R_3^x) \cup (R_2^x \cap R_3^x)$	$(R_1^y \cup R_3^y) \cap (R_2^y \cup R_3^y)$

Corollary 3.1 *Techniques which order the consensus operations (e.g., Tison's method) can be used to generate maximal subproducts.*

Tison's method must be modified before it can process subproducts. Note that in Tison's method consensus is performed once for each variable because further consensi at that variable cannot alter the final result. This is due to the fact that variables in the Boolean case can have only two values. In subproducts, variables become attributes and the values become attribute restrictions. This allows intermediate forms of consensus to be created which may need to be consensed further at the same attribute to obtain all possible

subproducts. Therefore, an iterative step must be added to Tison's method to ensure that all maximal subproducts are generated (See Table 3.4 for an illustration of this iterative step and its necessity). This consists of iterating the consensus operation on all new consensi at each attribute until no new subproducts are produced.

Tison's method for generating subproducts can be altered so that consensus is performed on each attribute in turn:

Algorithm 3.6 (Tison's method for subproducts) *Let L be the set $(A_1, A_2, \ldots, A_n)$ which is called the implicant list. At the end of the procedure, L consists of the prime implicants of Σ.*

For each attribute a in L,

1. *Initialize $C = \{\}$*
2. *For every pair of clauses A_i, $A_j \in L$*
   (a) *Add to C the consensus $\mathrm{CS}(A_i, A_j, a)$ if such a consensus exists*
   (b) *Delete from C all clauses Q such that there is a $Q' \in C$ which subsumes Q.*
3. *For every pair of clauses C_i, $C_j \in C$*
   (a) *Add to C the consensus $\mathrm{CS}(C_i, C_j, a)$ if such a consensus exists.*
   (b) *Delete from C all clauses Q such that there is a $Q' \in C$ which subsumes Q.*
4. *Delete from C all clauses Q such that there is a $Q' \in L$ which subsumes Q.*
5. *Add C to L*

Using the earlier example, we find the maximal subproducts of this attribute signature by Tison's method. This is shown in Table 3.4:

If the initial set of subproducts is expressed more succinctly, Table 3.5 is produced:

Table 3.4: Maximal subproducts generated by Tison's method (with trivial subproducts as input).

Consensing at the first coordinate					
1	$\{1\} \times \{1\} \times \{1\}$	2,7	$\{1,2\} \times \{2\} \times \{1\}$		
2	$\{1\} \times \{2\} \times \{1\}$				
3	$\{1\} \times \{3\} \times \{2\}$				
4	$\{1\} \times \{1\} \times \{3\}$				
5	$\{1\} \times \{2\} \times \{3\}$				
6	$\{1\} \times \{3\} \times \{3\}$				
7	$\{2\} \times \{2\} \times \{1\}$				
Consensing at the second coordinate					
1	$\{1\} \times \{1\} \times \{1\}$	1,2,7	$\{1\} \times \{1,2\} \times \{1\}$	4,5,6	$\{1\} \times \{1,2,3\} \times \{3\}$
3	$\{1\} \times \{3\} \times \{2\}$	4,5	$\{1\} \times \{1,2\} \times \{3\}$		
4	$\{1\} \times \{1\} \times \{3\}$	4,6	$\{1\} \times \{1,3\} \times \{3\}$		
5	$\{1\} \times \{2\} \times \{3\}$	5,6	$\{1\} \times \{2,3\} \times \{3\}$		
6	$\{1\} \times \{3\} \times \{3\}$				
2,7	$\{1,2\} \times \{2\} \times \{1\}$				
Consensing at the third, and final, coordinate					
1,2,7	$\{1\} \times \{1,2\} \times \{1\}$	1,2,4,5,6,7	$\{1\} \times \{1,2\} \times \{1,3\}$		
2,7	$\{1,2\} \times \{2\} \times \{1\}$	3,4,5,6	$\{1\} \times \{3\} \times \{2,3\}$		
3	$\{1\} \times \{3\} \times \{2\}$				
4,5,6	$\{1\} \times \{1,2,3\} \times \{3\}$				

In this example all subproducts which differ by a single coordinate have been expressed as a single subproduct. The complete set of maximal subproducts has been generated after a single iteration.

3.4.3 Incremental prime implicant generation

The IPIA (Incremental Prime Implicant Algorithm) of Kean and Tsiknis [35] takes a set of prime implicants, Π, and a clause, C. The algorithm returns $\Sigma \cup \Pi$, the set of prime implicants for $\{C\} \cup \Pi$. As this algorithm is an incremental version of Tison's method, it too must be modified to accept subproducts.

Algorithm 3.7 (IPIA)

Initialize $\Sigma = \{C\}$. *If* C *is subsumed by some clause in* Π *delete* C *from* Σ *and halt the procedure.*

Table 3.5: Maximal subproducts generated by Tison's method (with nontrivial subproducts as input).

Consensing at the first coordinate			
1,2 3 4,5,6 7	$\{1\} \times \{1,2\} \times \{1\}$ $\{1\} \times \{3\} \times \{2\}$ $\{1\} \times \{1,2,3\} \times \{3\}$ $\{2\} \times \{2\} \times \{1\}$	1,2,7	$\{1,2\} \times \{2\} \times \{1\}$
Consensing at the second coordinate			
1,2 1,2,7 3 4,5,6	$\{1\} \times \{1,2\} \times \{1\}$ $\{1,2\} \times \{2\} \times \{1\}$ $\{1\} \times \{3\} \times \{2\}$ $\{1\} \times \{1,2,3\} \times \{3\}$		
Consensing at the third coordinate			
1,2 1,2,7 3 4,5,6	$\{1\} \times \{1,2\} \times \{1\}$ $\{1,2\} \times \{2\} \times \{1\}$ $\{1\} \times \{3\} \times \{2\}$ $\{1\} \times \{1,2,3\} \times \{3\}$	1,2,4,5,6 3,4,5,6	$\{1\} \times \{1,2\} \times \{1,3\}$ $\{1\} \times \{3\} \times \{2,3\}$

For each biform variable x occurring in C do

(a) For each $S \in \Sigma$ and $P \in \Pi$ such that P,S have consensus on x do

i. $T = \mathrm{CS}(S, P, x)$

ii. $\Sigma = \Sigma \cup T$

(b) Delete any $D \in \Sigma \cup \Pi$ such that there is a $D' \in \Sigma \cup \Pi$ that subsumes D

The IPIA is modified in a fashion similar to Tison's method:

Algorithm 3.8 (IPIA for maximal subproducts)

Initialize $\Sigma = \{C\}$. If C is subsumed by some clause in Π delete C and halt the procedure.

For each attribute x occurring in C do

(a) Initialize $\Sigma = \{\}$

(b) Iterate until $\Sigma \cup \Pi$ is not modified

i. $T = \mathrm{CS}(S, P, x)$ *such that* $S \in \Sigma, P \in \Pi$

ii. $\Sigma = \Sigma \cup \{T\}$

iii. Delete any $D \in \Sigma \cup \Pi$ *such that there is a* D' *in* $\Sigma \cup \Pi$ *that subsumes* D

Again, using the earlier example:

Example

Find the maximal subproducts of this attribute signature. Table 3.6 shows the prime implicant generation.

Table 3.6: Maximal subproducts generated by IPIA (with trivial subproducts as input).

Π_1	1 $\{C\}$	$\{1\} \times \{1\} \times \{1\}$ $\{1\} \times \{2\} \times \{1\}$	1,$\{C\}$	$\{1\} \times \{1,2\} \times \{1\}$
Π_2	1 $\{C\}$	$\{1\} \times \{1,2\} \times \{1\}$ $\{1\} \times \{3\} \times \{2\}$		
Π_3	1 2 $\{C\}$	$\{1\} \times \{1,2\} \times \{1\}$ $\{1\} \times \{3\} \times \{2\}$ $\{1\} \times \{1\} \times \{3\}$	1,$\{C\}$	$\{1\} \times \{1\} \times \{1,3\}$
Π_4	1 2 3 $\{C\}$	$\{1\} \times \{1,2\} \times \{1\}$ $\{1\} \times \{3\} \times \{2\}$ $\{1\} \times \{1\} \times \{1,3\}$ $\{1\} \times \{2\} \times \{3\}$	1,$\{C\}$	$\{1\} \times \{1,2\} \times \{1,3\}$
Π_5	1 2 $\{C\}$	$\{1\} \times \{3\} \times \{2\}$ $\{1\} \times \{1,2\} \times \{1,3\}$ $\{1\} \times \{3\} \times \{3\}$	1,$\{C\}$ 2,$\{C\}$	$\{1\} \times \{3\} \times \{2,3\}$ $\{1\} \times \{1,2,3\} \times \{3\}$
Π_6	1 2 3 $\{C\}$	$\{1\} \times \{1,2\} \times \{1,3\}$ $\{1\} \times \{3\} \times \{2,3\}$ $\{1\} \times \{1,2,3\} \times \{3\}$ $\{2\} \times \{2\} \times \{1\}$	1,$\{C\}$	$\{1,2\} \times \{2\} \times \{1\}$

If this example is reformulated so that the initial set of subproducts is expressed more succinctly, Table 3.7 is produced:

Kean and Tsiknis [22] optimize their incremental algorithm by recognizing relationships among consensi during subsumption checking. Although this optimization decreases the average time complexity of the algorithm, it cannot affect the worst case complexity. As we are concerned with the actual consensus methods, we close this section by noting that the optimized version of IPIA can be generalized to generate the maximal subproducts of

Table 3.7: Maximal subproducts generated by IPIA (with nontrivial subproducts as input).

Π_1	1,2 $\{C\}$	$\{1\} \times \{1,2\} \times \{1\}$ $\{1\} \times \{3\} \times \{2\}$		
Π_2	1 2 $\{C\}$	$\{1\} \times \{1,2\} \times \{1\}$ $\{1\} \times \{3\} \times \{2\}$ $\{1\} \times \{1,2,3\} \times \{3\}$	1,$\{C\}$ 2,$\{C\}$	$\{1\} \times \{1,2\} \times \{1,3\}$ $\{1\} \times \{3\} \times \{2,3\}$
Π_3	1 2 3 $\{C\}$	$\{1\} \times \{1,2,3\} \times \{3\}$ $\{1\} \times \{1,2\} \times \{1,3\}$ $\{1\} \times \{3\} \times \{2,3\}$ $\{2\} \times \{2\} \times \{1\}$	2,$\{C\}$	$\{1,2\} \times \{2\} \times \{1\}$

an attribute signature.

3.5 Discussion

In general, iterative consensus performs fewer consensus operations at each level than the Quine-McCluskey algorithm, but must perform subsumption checking at each level. Intuitively, iterative consensus should generate the maximal subproducts at an earlier stage of the algorithm. The motivation for this algorithm is to take subproducts at the *ith* level and generate subproducts in the $i + jth$ level where $j \geq 1$. Since the maximal subproducts will lie in the outermost regions, an algorithm which reaches those frontiers quickly is more appealing. However, it is possible that these subproducts would not subsume the lesser subproducts at that particular level. It could be that the smaller subproducts were not subsumed until one or more additional iterations. Tison's method also consenses fewer times than the Quine-McCluskey algorithm and requires subsumption checking, but this method should be more efficient than either the Quine-McCluskey algorithm or iterative consensus.

For the sample problems, iterative consensus performs reasonably well. In fact, since the initial set of subproducts are trivial all consensi at the first level will be subsumed by the resulting subproducts. In this case, iterative consensus enjoyed a decided advantage against the Quine-McCluskey algorithm. In practice, however, the advantages of iterative consensus

are not so clear. It seems reasonable to assume that fewer intermediate subproducts will be generated by iterative consensus, but subsumption checking could negate this benefit. In this instance, the success of iterative consensus lies in the fact that most of the subproducts in the lower levels were subsumed after a single iteration. Thus the similar figures in subsumption checking could be attributed to the initial use of trivial subproducts and the large numbers of intermediate subproducts generated in the second level.

Let us begin to analyze these algorithms' performance by examining a special case of iterative consensus. Iterative consensus is equivalent to Quine-McCluskey when it can subsume all parent clauses at the lower levels and any clause which is not a parent is a prime implicant. Then the number of subsumption checks is equal for both methods and iterative consensus will perform no more consensi than Quine-McCluskey and possibly far fewer. Otherwise, for each subproduct generated by iterative consensus, the number of additional subsumption checks at level S_i can be calculated as

$$2\sum_{j=0}^{i-1}|S_i||S_j|.$$

Since the number of consensus operations in Tison's method and IPIA is bounded by the number of attributes, both methods should be superior to the Quine-McCluskey algorithm and iterative consensus in the general case. Kean and Tsiknis [22] prove that the worst case complexity of IPIA is $O((\frac{n}{k})^{2k})$ where n is $|\Pi|$, the number of prime implicants, and k is the number of biform variables in $\{C\}$, the clause being added to Π.

We have generalized consensus techniques to maximal subproducts. As production rules are merely the disjunctive case, we can apply these same methods to minimize internally disjunctive production rules by using the dual algorithm. The dual algorithm for disjunctive expressions intersects at the consensus attribute and merges at the nonconsensus attributes. In converting the production rules to disjunctive expressions, however, we lose the original forms of the rules. Chapter 4 discusses *oriented* consensus methods designed to retain

the original forms of production rules and to use those forms to dictate the outcome of consensus.

Chapter 4

Oriented consensus and widenings

The attributes in a production rule must occur either in the antecedent or in the consequent of that rule. This gives the rule an *orientation* which is an indication of how it is intended to be used. Converting production rules to their clausal form loses the orientation information and thus some valuable information concerning the original intended use of the rule.

In an *oriented rule* we distinguish the orientation information from the logical information and regard the rule as two separate components: the logical rule and the orientation. *Oriented consensus* takes two oriented rules and produces another oriented rule which is implied by the parents. In order to obtain an oriented rule from this consensus, it is necessary to follow some convention which determines the orientation of the result. A convention which produces the strongest possible attribute restrictions is introduced in Section 4.1. Attribute restrictions are *strengthened* when the disjunctive set is reduced or *weakened* when it is enlarged.

Consensing oriented rules can produce an oriented rule whose conclusion has been weakened when compared to either parent. Such a consensus is called a *widening* and the result is said to be *wider* than either parent. Widenings are undesirable because the conclusion is strictly weaker. Consequently, the parents of a widening provide a more concise representation than their widened offspring. In generating prime rules, these widened rules will also consense with other rules to produce further undesirable rules with weakened conclusions.

Widened rules will not subsume their parents, nor will their parents subsume them. It is, however, desirable to remove them. Section 4.2 describes how the results of widenings can be pruned from the set of primes.

In Section 4.3 it is shown that widenings can be distributed throughout a proof such that no widening occurs before any other consensus. In fact, it is shown that the widenings themselves can be distributed when the consensus is at distinct attributes. When the attributes are the same, it is possible for a widening to be completely eliminated by distribution. Section 4.4 examines the effects of preventing widenings from occuring during the generation of primes, and discusses methods for eliminating any resulting rules which are not prime. This chapter concludes by noting that oriented consensus is compatible with the consensus techniques described in Chapter 3.

4.1 Oriented consensus

Consensus produces a new rule which is implied by its parents. When the rules are oriented, there must be a convention for orienting the results of the consensus. In the remainder of this section, the entailment operator $\vdash$ is used to separate the antecedents of a rule from the consequents.

Definition 4.1

1. *The* **surface form** *of a rule is given as*

$$A \vdash C$$

where A is the set of antecedents and C is the set of consequents.

2. *An* **oriented** *rule R is a pair (S, θ) where S is the surface form of R and θ is the* **orientation** *of R. θ is a set of attributes such that no attribute in the conseqeuent of S is contained in θ and every attribute in the antecedent of S is in θ.*

3. *An oriented rule has* **minimal orientation** *if all antecedent attributes of the surface form occur in the orientation.*

The orientations of the parents will determine the orientation of the result of an oriented consensus. Two rules $R_1 = (S_1, \theta_1)$ and $R_2 = (S_2, \theta_2)$ have the same orientation if and only if $\theta_1 = \theta_2$.

Definition 4.2 *Given two oriented rules R_1 and R_2, there is an* **orientation clash** *at attribute q in R_1 and R_2 if q occurs in the orientation of one rule and the consequent of the surface form of the other rule.*

If there is an orientation clash at an attribute then some uniform convention must be followed for determining the placement of that attribute in the surface form of the new rule. The orientation is important as an attribute restriction can be eliminated in consensus and can still be in the orientation even though it is missing from the surface form. It is this maintenance of orientation which ensures that the proof manipulations of Sections 4.3 and 4.4 will apply.

Definition 4.3 *The* **oriented consensus** *of $R_1 = (S_1, \theta_1)$ and $R_2 = (S_2, \theta_2)$ at attribute q is denoted as*

$$\mathrm{CS}^*(R_1, R_2, q).$$

The orientation of the new rule is defined as

$$cs(\theta_1, \theta_2, q) = \begin{cases} \theta_1 \cup \theta_2 & \text{if } q \in \theta_1 \text{ and } q \in \theta_2 \\ \theta_1 \cup \theta_2 \setminus \{q\} & \text{otherwise} \end{cases}$$

The following convention orients the result of consensus in the surface forms. and orientation of the parents.

1. *If θ_1 and θ_2 have an orientation clash at the consensus attribute q, then the residual of q is placed in the consequent of the surface form.*

$$\frac{\begin{array}{llll} q \diamond c_0 & \vdash & & \theta_1 \\ & \vdash & q \diamond c_1 & \theta_2 \end{array}}{\vdash \quad q \diamond c_1 - c_0 \quad \theta = cs(\theta_1, \theta_2, q)}$$

2. *If θ_1 and θ_2 have an orientation clash at a non-consensus attribute q', then the residual of q' is in the antecedent of the result.*

$$\begin{array}{lll} q' \diamond c_0 & \vdash & \theta_1 \\ & \vdash \; q' \diamond c_1 & \theta_2 \\ \hline q' \diamond c_0 - c_1 & \vdash & \theta = cs(\theta_1, \theta_2, q) \end{array}$$

3. *If there is no orientation clash between R_1 and R_2 at q, q is brought into the new rule with the same orientation and the new orientation is $cs(\theta_1, \theta_2, q)$.*

Example 4.1 presents several consensus operations. In Example 4.1(3) the restriction $q_1 \diamond [b]$ is treated as an exception.

Example 4.1

1. *At a consensus attribute q_1*

$$\begin{array}{lll} q_0 \diamond [a,c] \wedge \underline{q_1 \diamond [a,b,c]} & \vdash \; q_2 \diamond [a] & \theta_1 = \{q_0, q_1\} \\ q_0 \diamond [b,c] & \vdash \; \underline{q_1 \diamond [a,c,e]} \vee q_2 \diamond [b] & \theta_2 = \{q_0\} \\ \hline q_0 \diamond [c] & \vdash \; \underline{q_1 \diamond [e]} \vee q_2 \diamond [a,b] & \theta = \{q_0\} \end{array}$$

2. *At a non-consensus attribute q_1*

$$\begin{array}{lll} q_0 \diamond [a,c] \wedge \underline{q_1 \diamond [b,c,d]} & \vdash \; q_2 \diamond [a] & \theta_1 = \{q_0, q_1\} \\ q_0 \diamond [b,c] & \vdash \; \underline{q_1 \diamond [a,c]} \vee q_2 \diamond [b] & \theta_2 = \{q_0\} \\ \hline q_0 \diamond [a,b,c] \wedge \underline{q_1 \diamond [b,d]} & \vdash \; q_2 \diamond [a,b] & \theta = \{q_0, q_1\} \end{array}$$

3. *Switching q_1 to the consequent*

$$\begin{array}{lll} q_0 \diamond [a] & \vdash \; q_2 \diamond [a] & \theta_1 = \{q_0\} \\ q_0 \diamond [b] \wedge \underline{q_1 \diamond [b]} & \vdash \; q_2 \diamond [b] & \theta_2 = \{q_0, q_1\} \\ \hline q_0 \diamond [a,b] & \vdash \; q_2 \diamond [a,b] \vee \underline{q_1 \diamond \neg [b]} & \theta = \{q_0\} \end{array}$$

With some slight modifications, subsumption can accomodate oriented rules with internal disjunctions. Most notably, we address subsumption at both a logical and orientation level since we have split production rules into two parts.

Definition 4.4 (Oriented subsumption) *An oriented rule $R' = (A' \vdash C', \theta')$ subsumes another oriented rule $R = (A \vdash C, \theta)$ if and only if*

1. $A \subseteq A'$,

2. $C' \subseteq C$, *and*

3. $\theta' \subseteq \theta$.

This is denoted as $R' \succ R$.

Intuitively, this means that orientation subsumption is given by the least number of attributes in the antecedent, i.e., the less that must be assumed, the better. It is immediate that

Lemma 4.1 *If* $R_1' \succ R_1$ *and* $R_2' \succ R_2$ *then*

$$\mathrm{CS}^*(R_1', R_2', q) \succ \mathrm{CS}^*(R_1, R_2, q).$$

We must now prove that oriented consensus operations can be distributed:

Lemma 4.2

$$\mathrm{CS}^*(\mathrm{CS}^*(R_1, R_2, q), R_3, q') = \mathrm{CS}^*(\mathrm{CS}^*(R_1, R_3, q'), \mathrm{CS}^*(R_2, R_3, q'), q).$$

In order to prove this lemma, we introduce the predicate A_q. The predicate $A_q(\theta)$ is true if and only if attribute q is in the orientation. It is clear that

$$A_q(\mathrm{CS}^*(\theta_1, \theta_2, q)) = A_q(\theta_1) \wedge A_q(\theta_2), \text{ and } A_q(\mathrm{CS}^*(\theta_1, \theta_2, q')) = A_q(\theta_1) \vee A_q(\theta_2).$$

We must prove that

$$\mathrm{CS}^*(\mathrm{CS}^*(\theta_1, \theta_2, q_1), \theta_3, q_2) = \mathrm{CS}^*(\mathrm{CS}^*(\theta_1, \theta_3, q_2), \mathrm{CS}^*(\theta_2, \theta_3, q_2), q_1).$$

Proof of Lemma 4.2 (by cases)

1. $q \neq q_1 \neq q_2$.

$$(A_q(\theta_1) \vee A_q(\theta_2)) \vee A_q(\theta_3) = (A_q(\theta_1) \vee A_q(\theta_3)) \vee (A_q(\theta_2) \vee A_q(\theta_3)).$$

This holds because $\vee$ is associative and idempotent.

2. $q = q_1 = q_2$.

$$(A_q(\theta_1) \wedge A_q(\theta_2)) \wedge A_q(\theta_3) = (A_q(\theta_1) \wedge A_q(\theta_3)) \wedge (A_q(\theta_2) \wedge A_q(\theta_3)).$$

This holds because $\wedge$ is associative and idempotent.

3. $q = q_1$, $q \neq q_2$.

$$(A_q(\theta_1) \wedge A_q(\theta_2)) \vee A_q(\theta_3) = (A_q(\theta_1) \vee A_q(\theta_3)) \wedge (A_q(\theta_2) \vee A_q(\theta_3)).$$

This holds as $\vee$ distributes over $\wedge$.

4. $q \neq q_1$, $q = q_2$.

$$(A_q(\theta_1) \vee A_q(\theta_2)) \wedge A_q(\theta_3) = (A_q(\theta_1) \wedge A_q(\theta_3)) \vee (A_q(\theta_2) \wedge A_q(\theta_3)).$$

This holds as $\wedge$ distributes over $\vee$.

An oriented consensus proof requires that the consensed attribute be present in the orientation of both rules.

4.2 Widenings and narrow primes

We can now discuss forms of oriented consensus called *widenings* which weaken conclusions and consequently are undesirable. Those primes which cannot be generated from other prime rules by one or more widenings are called *narrow* and form an important subset of the generated primes.

Definition 4.5

1. *A rule $R' = (A' \vdash C', \theta')$ is* **wider** *than rule $R = (A \vdash C, \theta)$ if and only if $C \subset C'$ and $\theta \subseteq \theta'$.*

2. *Consensus is a* **widening** *if the result is wider than both parents. A widening of R_1 and R_2 at q is denoted as*

$$\mathrm{CS}^*_{\mathrm{W}}(R_1, R_2, q).$$

A rule R_1 is *narrower* than rule R_2 if and only if R_2 is wider than R_1.

Note that widenings always weaken the consequent of the new rule. Such rules are usually undesirable since most knowledge-based systems require rules possessing the strongest consequents. These ideas are illustrated by Example 4.2.

Example 4.2

1.

$$\begin{array}{llll} q_1 \diamond [a] & \vdash & q_2 \diamond [a] & \theta_1 = \{q_1\} \\ q_1 \diamond [b] & \vdash & q_2 \diamond [b] & \theta_2 = \{q_1\} \\ \hline q_1 \diamond [a,b] & \vdash & q_2 \diamond [a,b] & \theta = \{q_1\} \end{array}$$

In this example, all attribute restrictions have been weakened.

2.

$$\begin{array}{llll} q_0 \diamond [a,c] \wedge q_1 \diamond [a] \wedge q_2 \diamond [a] & \vdash & q_3 \diamond [a] & \theta_1 = \{q_0, q_1, q_2\} \\ q_0 \diamond [a,e] \wedge q_4 \diamond [a \wedge q_2 \diamond [b] & \vdash & q_3 \diamond [b] & \theta_2 = \{q_0, q_1, q_4\} \\ \hline q_0 \diamond [a] \wedge q_1 \diamond [a] \wedge q_4 \diamond [a] \wedge q_2 \diamond [a,b] & \vdash & q_3 \diamond [a,b] & \theta = \{q_0, q_1, q_2, q_4\} \end{array}$$

The antecedent is more restrictive, but the consequent is weaker.

3.

$$\begin{array}{llll} q_1 \diamond [a] & \vdash & q_2 \diamond [a] & \theta_1 = \{q_1\} \\ q_1 \diamond \neg [a] & \vdash & q_2 \diamond [b] & \theta_2 = \{q_1\} \\ \hline & \vdash & q_2 \diamond [a,b] & \theta = \{q_1\} \end{array}$$

In this example, q_1 is disappears from the surface form of the new rule, but remains in the orientation.

4.

$$\begin{array}{llll} q_0 \diamond [a,c] \wedge q_1 \diamond [a] & \vdash & q_2 \diamond [a] & \theta_1 = \{q_0, q_1\} \\ q_0 \diamond [a,e] \wedge q_1 \diamond [a] & \vdash & q_2 \diamond [a] & \theta_2 = \{q_0, q_1\} \\ \hline q_0 \diamond [a,c,e] \wedge q_1 \diamond [a] & \vdash & q_2 \diamond [a] & \theta = \{q_0, q_1\} \end{array}$$

This consensus is not a widening because the new rule does not possess a weakened consequent. Instead, the parents are subsumed by the new rule.

Widenings only occur in the presence of orientation and internal disjunctions. Consider the widening

$$\begin{array}{llll} q_1 \diamond [a] & \vdash & q_2 \diamond [a] & \theta_1 = \{q_1\} \\ q_1 \diamond [b] & \vdash & q_2 \diamond [b] & \theta_2 = \{q_1\} \\ \hline q_1 \diamond [a,b] & \vdash & q_2 \diamond [a,b] & \theta = \{q_1\} \end{array}$$

In clausal notation the parents become

$$\{\neg q_1(a), q_2(b)\},\ \{\neg q_1(b), q_2(b)\}.$$

Resolution does not pair these clauses because there is no complementary set of literals; resolution may be viewed as insisting upon orientation clashes while consensus for internal disjunctions does not. Clauses may be implicitly oriented by collecting negated literals as the antecedents and positive literals as the consequents. Generalizing consensus to internal disjunctions makes orientation an imposed structure rather than an implicit structure. However, this does not mean that there is anything productive about an antecedent-to-antecedent consensus in which there is no orientation clash.

With their distinct characteristics, widenings can be identified as they are being formed. Therefore, the most efficient method of eliminating widened rules would be to prevent their formation in the pairing process. The price of such a strategy, however, may be the loss of completeness. The aim of this section is to investigate how primes produced by widenings can be identified for removal and the conditions under which the prevention of widenings at the generation stage does not affect completeness.

Lemma 4.3 *A consensus is a* **widening** *if and only if the consensus attribute is in the orientation of both rules.*

Proof of Lemma 4.3

Consequent-to-consequent consensus causes a restriction of the consequent and thus cannot be a widening. Antecedent-to-consequent consensus is impossible because the orientation of the resulting rule is smaller. Finally, if antecedent-to-antecedent consensus is performed then the consequent is weakened and a widening results. □

Corollary 4.1 *Suppose that* $R_1 = (S_1, \theta_1)$, $R_2 = (S_2, \theta_2)$ *and* $P = \mathrm{CS}^*(R_1, R_2, q)$ *is a widening. Let* $R_1' = (S_1', \theta_1)$ *and* $R_2' = (S_2', \theta_2)$. *If* $R_1' \succ R_1$, $R_2' \succ R_2$ *and* $q \in \theta_1 \cap \theta_2$ *then* $P = \mathrm{CS}^*(R_1', R_2', q)$ *is also a widening.*

Not all primes are needed to retain the knowledge embodied in a set of production rules. In this section, we examine a subclass of the primes generated by oriented consensus and determine the difficulty of detecting membership in of that subclass. Our motivation is to eliminate those primes which can be produced from widenings of other primes.

Definition 4.6 *A* **wide** *prime is one which has a widening proof from other primes. A prime which is not wide is called a* **narrow** *prime.*

It is relatively simple to determine whether a prime is narrow by using the fact that it must not be wide:

Lemma 4.4 *A prime P is wide if and only if* $\exists R_1 = (S_1, \theta_1)$, $R_2 = (S_2, \theta_2) \in P$ *such that*

$$P = \mathrm{CS}^*_{\mathrm{W}}(R_1, R_2, q).$$

Proof of Lemma 4.4

If a widening proof exists, then consider the last step of the proof. Replace the parents with primes. Since q is in the orientation of the result, it must have been in the orientation of its parents. Thus, it must be a widening. □

4.3 Delaying widenings in proofs

We will show that widenings can be distributed through a proof tree so that they follow all other consensi. First, any proof tree can be transformed into one in which all the widenings follow all other consensi. It suffices to show that a proof tree of the form shown in Figure 4.3 can be transformed into the one shown in Figure 4.3.

In this distributed proof, no widening precedes a consensus which is not a widening. This proposition is proven by Lemmas 4.5 and 4.6.

Lemma 4.5

$$\mathrm{CS}^*(\mathrm{CS}^*_{\mathrm{W}}(R_1, R_2, q), R_3, q) = \mathrm{CS}^*(\mathrm{CS}^*(R_1, R_3, q), \mathrm{CS}^*(R_2, R_3, q), q).$$

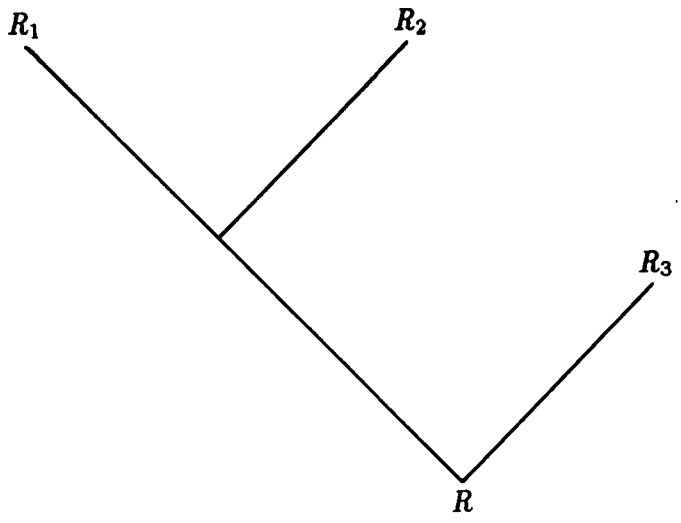

Figure 4.1: The original proof.

Proof of Lemma 4.5 Let $A_q(\theta_3) = \bot$.

$$(A_q(\theta_1) \wedge A_q(\theta_2)) \wedge A_q(\theta_3) = \bot.$$

$$(A_q(\theta_1) \wedge A_q(\theta_3)) \wedge (A_q(\theta_2) \wedge A_q(\theta_3)) = \bot.$$

In $\mathrm{CS}^*(R_1, R_3, q)$ and $\mathrm{CS}^*(R_2, R_3, q)$ q is removed from the orientation of the resulting rules. Thus,

$$\mathrm{CS}^*(\mathrm{CS}^*(R_1, R_3, q), \mathrm{CS}^*(R_2, R_3, q), q)$$

is a consequent-to-consequent consensus which cannot be a widening. □

Lemma 4.6

$$\mathrm{CS}^*(\mathrm{CS}^*_{\mathrm{W}}(R_1, R_2, q), R_3, q') = \mathrm{CS}^*_{\mathrm{W}}(\mathrm{CS}^*(R_1, R_3, q'), \mathrm{CS}^*(R_2, R_3, q'), q).$$

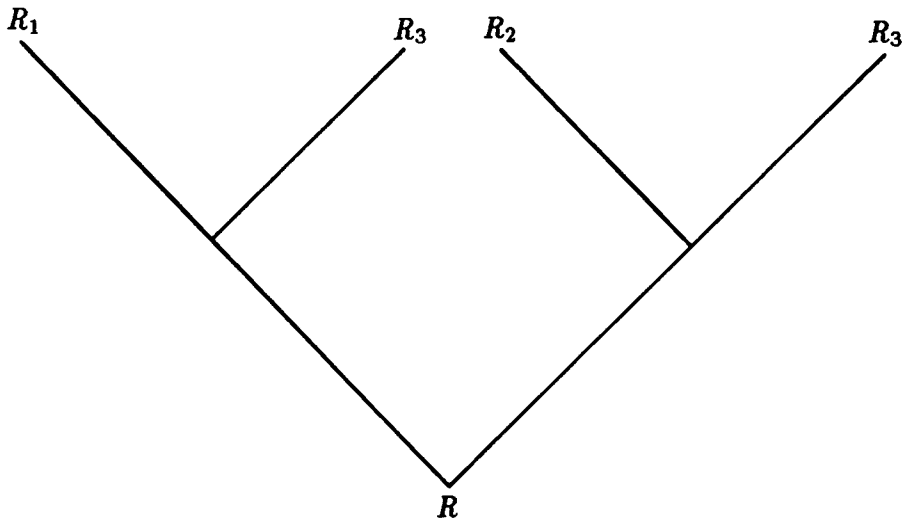

Figure 4.2: The distributed proof.

Proof of Lemma 4.6 Let $A_q(\theta_3) = \bot$.

$$(A_q(\theta_1) \wedge A_q(\theta_2)) \vee A_q(\theta_3) = \top.$$

$$(A_q(\theta_1) \vee A_q(\theta_3)) \wedge (A_q(\theta_2) \vee A_q(\theta_3)) = \top.$$

$\mathrm{CS}^*(R_1, R_3, q')$ and $\mathrm{CS}^*(R_2, R_3, q')$ have q in their respective orientations. Thus,

$$\mathrm{CS}^*(\mathrm{CS}^*(R_1, R_3, q'), \mathrm{CS}^*(R_2, R_3, q'), q)$$

is a widening. Consequently, the widening distributes. □

The next theorem shows that these results can be used to distribute widenings over a proof tree of depth n.

Theorem 4.1 *All widenings in a proof $\mathcal{P}$ can be distributed such that all other consensi in the distributed proof $\mathcal{P}'$ precede them.*

Proof of Theorem 4.1 by induction on the depth of $\mathcal{P}$

Basis. depth($\mathcal{P}$) = 2

Follows from Lemmas 4.5 and 4.6.

Inductive step.

Assume that all proof trees of depth $\leq n$ satisfy the theorem. Given two trees $\mathcal{P}_1$, $\mathcal{P}_2$ where depth($\mathcal{P}_1$) = n and depth($\mathcal{P}_2$) $\leq n$, the proof tree formed by consensing $\mathcal{P}_1$ and $\mathcal{P}_2$ must be of depth $n+1$; it is shown in Figure 4.3.

If R_3 is a widening, then the theorem holds since $\mathcal{P}_1$, $\mathcal{P}_2$ satisfy the theorem and no nonwidening has been introduced. If, however, R_3 is a nonwidening it must be distributed over $\mathcal{P}_1$ and $\mathcal{P}_2$ until no widenings precede it. Use Lemmas 4.5 and 4.6 after splitting $\mathcal{P}_1$ into subtrees $\mathcal{P}_{11}$, $\mathcal{P}_{12}$. The proof tree which results is shown in Figure 4.4.

By Lemmas 4.5 and 4.6, the widening has been distributed or eliminated. If depth($\mathcal{P}_2$) < n then the inductive hypothesis states that consensi α, β can be distributed such that the theorem holds, because α, β terminate at depth n. If depth($\mathcal{P}_2$) = n, one more distribution step is needed; this last step is shown in Figure 4.5.

At this stage, all nonwidenings in $\mathcal{P}'$ occur at the *nth* level of all subtrees and can be distributed given the inductive hypothesis. □

4.4 Pruning the prime rules

Oriented consensus and subsumption produce the complete set of prime rules. By preventing widenings, some primes may not be generated and some generated rules may not be prime. The set of rules which are generated by blocking widenings and eliminating rules using oriented subsumption is called the *modified* set of primes $\mathbf{P}_{\mathrm{M}}$. In this section, we show that all narrow primes will be present in the modified set of primes and that they can be identified relatively easily.

Lemma 4.7 *All narrow primes are present in* $\mathbf{P}_{\mathrm{M}}$.

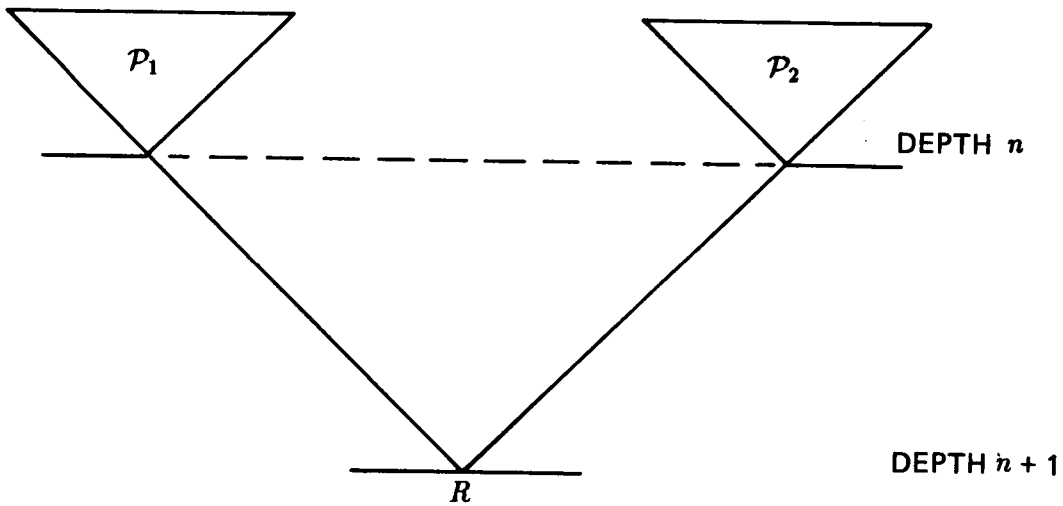

Figure 4.3: The initial proof tree.

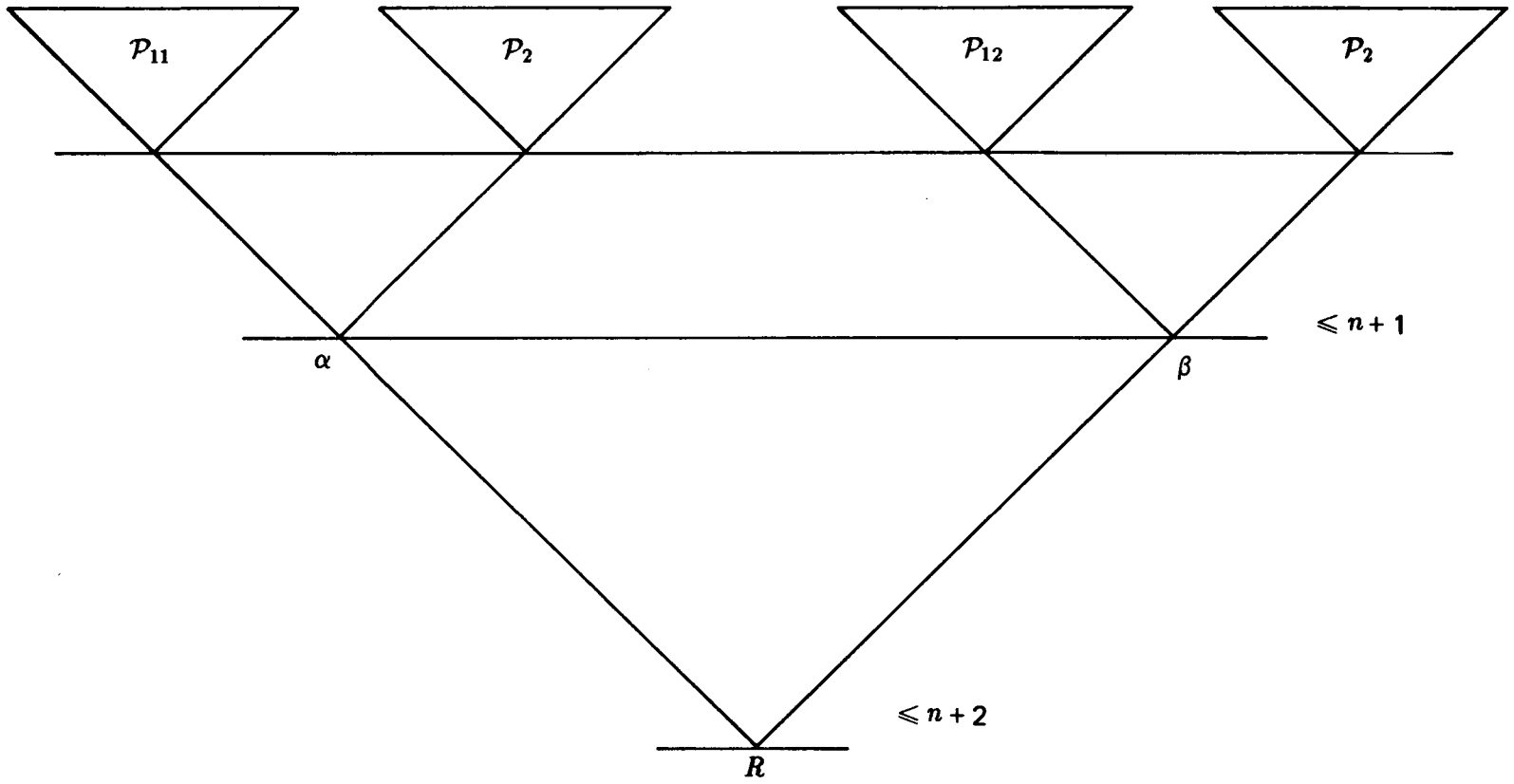

Figure 4.4: The proof tree after the first distribution.

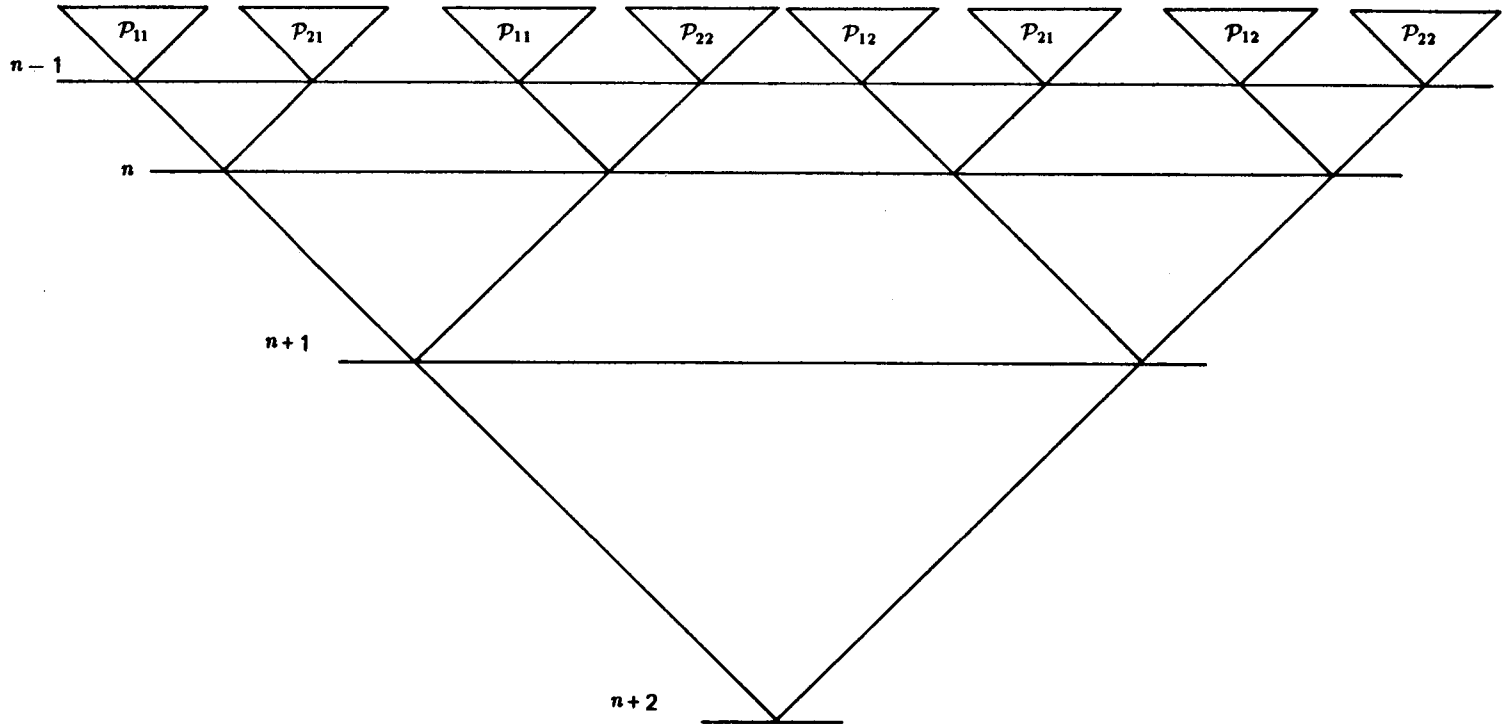

Figure 4.5: The proof after the second distribution.

Proof of Lemma 4.7 If P is narrow but $P \notin \mathbf{P_M}$, then every proof of P contains widenings which cannot be removed by delaying widenings. Consider a proof of P which terminates in a widening. This widening can be strengthened as in Corollary 4.1 to show that P is not narrow. □

Definition 4.7 *A* widening proof *of R is a consensus proof of R' in which only widenings are used, where $R' \succ R$.*

The following observation is useful in identifying the set of narrow primes.

Lemma 4.8 *If there is a widening proof of W from $C_1, \ldots, C_n$ then $C_1, \ldots, C_n$ are narrower than W and $C_1, \ldots, C_n \Longrightarrow W$.*

Unfortunately, while this condition is necessary to find those rules in $\mathbf{P_M}$, it may not be sufficient. Were there no antecedent-to-consequent consensi, this condition would be sufficient; however, such consensi could occur. Further research may provide a more satisfactory result.

4.5 Discussion

Distributing widenings is compatible with the consensus technique devised by Tison [56] and extended by Kean and Tsiknis [22]. Lemmas 4.5 and 4.6 indicate that the distribution of widenings is allowable as long as an attribute is only used once in the proof. However, expressions with internal disjunctions may require more than one consensus operation at an attribute. In Chapter 3, Tison's method was modified to permit iterative consensus on each attribute in turn. This allowed Tison's method and the incremental version of Kean and Tsiknis [22] to generate all primes from a set of expressions with internal disjunctions.

We strengthen the results of [56] and [22] by introducing *oriented consensus* and restricting *widenings*. In doing so, we keep more rules than is necessary because we have

limited the power of subsumption by requiring that the orientation information in the rules be identical.

Chapter 5

Covering prime implicants

When all consensi have been generated, the result is a set of maximal subproducts. As depicted earlier in Chapter 2, not all maximal subproducts are required for a complete description of the attribute signature. Since maximal subproducts are analogous to prime implicants, we can modify covering techniques for prime implicants to obtain a means of producing an irredundant set of maximal subproducts needed to represent an attribute signature.

Covering a set of prime implicants has been accomplished by branching techniques [43,29,58], algebraic methods [5,18,19,27,42], and integer programming [6,12,50]. The covering procedure discussed herein is a branching method which attempts to reduce the size of the covering problem through the identification of unique components and subsumption of covered elements. Branching is performed by either removing a prime implicant from consideration or adding it to the cover. Each branching step creates two new covering problems of lesser size. Although this technique is guaranteed to generate all irredundant covers, it is computationally expensive as the number of irredundant covers grows exponentially on the number of primes.

5.1 The set covering problem

The problem of finding an irredundant set of primes is related to set covering:

Definition 5.1 *Given sets* $V = \{v_1, v_2 \ldots v_n\}$, $P = \{p_1, p_2 \ldots p_n\}$ *where* $p_j \subseteq V$ *and*

$$\bigcup_{j=1}^{n} p_j = V.$$

Associated with each set p_j *may be a non-negative cost* c_j ; $P' \subseteq P$ *such that*

1. $\forall v_i \exists p_j \in P'$ *such that* $v_i \in P$, *and*

2. $\sum c_j$ *is minimal, where the sum is over* j, *such that* $p_j \in P$,

is said to be a **minimal cost** *cover. If the set of all* C_j*'s is identical, then the minimal cost cover is the* **minimal cardinality** *cover.*

The set covering problem is known to be NP-complete if all P_j's have $|P_j| < 2$; if all P_j's have $|P_j| = 2$ the problem can be solved by matching techniques [17]. The covering algorithm described in this chapter is based upon the reduction of the covering problem into smaller subproblems. Branching is performed when other types of reduction (e.g., subsumption) are not applicable. By iterating this procedure, every problem can be reduced to the trivial subproblem with $P_j = \emptyset$. However, the generation of 2^n subproblems from n branches in the worst case is no better than enumerating all possible covers and verifying that they are irredundant. In most cases, however, the average complexity of this problem is much less than 2^n.

Definition 5.2 *Let* S *be a signature over* $(A_1, \ldots, A_n)$.

1. *If* S *is the conjunction of prime rules*

$$S = \bigwedge_{j \in J} R_j$$

such that no rule R_j *can be excluded without the equality failing, then the conjunction is said to be an* **irredundant cover**.

2. *A prime rule which is contained in every irredundant cover is said to be* **essential**.

3. A prime rule which is not contained in any irredundant cover is said to be **inessential.**

Having found all prime rules, a table can be constructed to represent attribute-value pairs as columns and prime rules as rows. An entry in the table indicates that the attribute-value pair for that column is covered by the prime rule for that row. Attribute-value pairs covered by only one rule must be included; other attribute mappings covered by this process are then removed. The reduced table is analyzed to determine which remaining expressions complete the cover. A table in which no mappings are uniquely covered is said to be *cyclic.* Such tables can be analyzed to find all combinations which can cover the signature. This branching technique is employed by the Quine-McCluskey algorithm.

At a particular stage in the algorithm there are two sets, those rules one has decided to definitly include in the cover, C, and those rules which could still be used in the cover, S. The set covering algorithm can be characterized by three basic operations:

1. Rule inclusion: If $r \in S$ is such that $r \not\subset S \bigcup C$ then r is **relatively essential** and must be removed from S and included in C.

2. Rule dominance: If $R \in S$ has $R \subset \bigcup C$ then r can be removed from S.

3. If neither of the above can be performed, we may branch by picking an arbitrary rule and either **remove** it from S or **force** it into C by removing it from S and placing it in C.

If r is removed, this algorithm is similar to other branching techniques; for the forced rules, another set F must be maintained. The set F contains all rules which have been forced. As a rule may be derived from combinations of prime rules, forcing a rule could lead to covers which are not irredundant. When a prime rule is forced, its unique components (i.e., those found nowhere else in S) are determined. If there are none, that rule is not forced; instead, another rule is selected. When relatively essential rules are later added

to C, they are checked again to verify that all rules in F still possess unique components. Otherwise, C will not be an irredundant cover and that branch of the search should be terminated. The algorithm then backtracks, removing the last forced rule and proceeding down a new path.

The following example demonstrates the covering procedure prescribed by McCluskey [29]. As the expressions are in conjunctive form, the search for essentials can be perfomed by inspection. Similarly, the reduction techniques can be applied in a straightforward manner.

Example The prime implicants generated in Chapter 3 will not be covered. The initial prime implicant table is given in Table 5.1:

Table 5.1: Initial prime implicant chart.

	Coordinate 1			Coordinate 2			Coordinate 3		
subproduct	1	2	3	1	2	3	1	2	3
1	X				X		X		
1		X			X		X		
2	X					X		X	
2	X					X			X
3	X						X		
3	X								X
3	X						X		
3	X								X
4	X			X					X
4	X				X				X
4	X					X			X

After the first set of operations, Table 5.2 results:

Table 5.2: The chart after one reduction.

	Coordinate 2
subproduct	2
3	X
4	X

At this stage, either of the remaining rules may be used. Thus, the set of irredundant covers are obtained:

$$\{\{p_1, p_2, p_3\}, \{p_1, p_2, p_4\}\}.$$

5.2 Subtraction of subproducts

In the case of disjunctive expressions, covering by inspection is more difficult. Roth [48] presents the # or *sharp* algorithm which subtracts vertices of one cube from another. The sharp operation takes a set of vertices A and subtracts the set of vertices from B, forming the set of subcubes in A which are not included in B. This operation may produce more than one subcube. Denoted $A\#B$, this operation is algebraically defined by Table 5.3.

Table 5.3: Coordinate #-operation $a_i\#b_i$.

a_i	b_i		
	0	1	x
0	z	y	z
1	y	z	z
x	1	0	z

To describe the #-operation between two subproducts, let $c^r = (a_1, a_2, \ldots, a_n)$ and $c^s = (b_1, b_2, \ldots, b_n)$.

$$c^r \# c^s = \begin{cases} c^r & \text{if } a_i\#b_i = y \text{ for any } i \\ \emptyset & \text{if } a_i\#b_i = z \text{ for all } i \\ \bigcup_i (a_i, \ldots, a_{i-1}, \alpha_i, a_{i+1}, \ldots, a_n) & \text{where } a_i\#b_i = \alpha_i = 0 \text{ or } 1 \end{cases}$$

The #-operation has the following properties:

$c^r\#c^s = c^r$ if $c^r \cap c^s = \emptyset$.
$c^r\#c^s \subseteq c^r$.
$c^r\#c^s \neq c^s\#c^r$ (noncommutative).
$(c^r\#c^s)\#c^t \neq c^r\#(c^s\#c^t)$ (nonassociative).

It also satisfies the distributive laws

$$(c^r \cup c^s)\#c^t = (c^r\#c^t) \cup (c^s\#c^t),$$
$$(c^r \cap c^s)\#c^t = (c^r\#c^t) \cap (c^s\#c^t),$$

and the following commutative law

$$(c^r \# c^s) \# c^t = (c^r \# c^t) \# c^s.$$

A similar operation can be defined for sets. First the coordinate operation must be modified for this algorithm to manipulate sets. Table 5.4 illustrates the coordinate operations for sets.

Table 5.4: Coordinate subtraction $A_i - B_i$.

	B_i		
A_i	$\neg B_i$	B_i	x
$\neg A_i$	$A_i - B_i$	y	z
A_i	y	$A_i - B_i$	z
x	B_i	$\neg B_i$	z

This subtraction operation must be altered to reflect that individual attributes comprising subproducts may not be completely subtracted as in the Boolean case. Let $c^r = (a_1, a_2, \ldots, a_n)$ and $c^s = (b_1, b_2, \ldots, b_n)$.

$$c^r \# c^s = \begin{cases} c^r & \text{if } a_i \# b_i = y \text{ for any } i \\ \emptyset & \text{if } a_i \# b_i = z \text{ for all } i \\ \bigcup_i (a_i, \ldots, a_{i-1}, \alpha_i, a_{i+1}, \ldots, a_n) & \text{where } \alpha_i \neq y, \alpha_i \neq z \end{cases}$$

The sharp algorithm was devised for vertices expressed as conjuncts; however, a dual operation can be defined for disjuncts using the identity

$$\overline{(x_1 \wedge \cdots \wedge x_n)} = \overline{\neg x_1 \vee \cdots \vee \neg x_n}.$$

Consequently, only the set minus operations need be reversed to provide the dual coordinate operation. Table 5.5 reflects this change.

We now present an algorithm which performs this subtraction routine for maximal subproducts.

Algorithm (Subproduct subtraction)

Table 5.5: Dual of subtraction for disjunctive expressions.

	B_i		
A_i	$\neg B_i$	B_i	x
$\neg A_i$	$B_i - A_i$	y	z
A_i	y	$B_i - A_i$	z
x	B_i	$\neg B_i$	z

Procedure: Let A and B be two arrays consisting of a set of vertices. Each vertex is itself an array representing the coordinates. The algorithm returns A', an array which contains the set of subcubes in A which are not included in B.

$A' \longleftarrow$ MINUS(A, B)
$A' = \{\}$; $n = |A|$; $i = 0$; $j = 0$;
do while $i \leq n$
  $\{a$:=$A[i]$;
  b:=$B[i]$;
  if $a \cap b = \emptyset$ **then return** $\{A\}$
    else if $a = \emptyset$ **and** $b \neq \emptyset$ **then** $\{A'[j] := A[i]$:=b
      j:=$j + 1\}$
      else if $a \neq \emptyset$ **and** $b \neq \emptyset$ **then** $\{a'$:=$a - b$
      if $s' \neq \{\}$ **then** $\{A'[j]$:=$A[i]$:=s'
        j:=$j + 1\}\}$
  i:=$i + 1\}$
return A'

5.3 The set covering algorithm

Algorithm (Basic covering algorithm)

Procedure: Find the set of all irredundant covers for a set of prime implicants. Let PI be the initial set of prime implicants, $F = \{\}$ be the set of prime implicants forced into a cover and $E = \{\}$ be the set of essential prime implicants. Backtracking is controlled by a global stack.

$C \longleftarrow$ COVER(PI, F, E)
$i = 0$;
if $PI = \{\}$ or $PI = \{pi\}$ **then return** $PI \cup E$;
  else { E' =FIND ESSENTIALS(PI, E);
    PI':=DELETE(E');
    do while $i \leq |F|$ {
      if $F'[i]$:=MINUS$(E \cup E', F[i]) = \emptyset$ **then backtrack**;

```
        else i:=i + 1 }
    I:=FIND INESSENTIALS(PI', E ∪ E');
    PI':=DELETE(I);
    if PI' = PI then S:=DIRECT SEARCH(PI);
    if S = ∅ then return (COVER(PI', F', E ∪ E' ∪ S));
    else return (COVER(PI', F', E ∪ E'));
}
```

The routines FIND ESSENTIALS and FIND INESSENTIALS identity essential and inessential subproducts. A subproduct is essential if there is some component which is not contained in any other subproduct. Conversely, a subproduct is inessential if it is completely covered by the union of all other subproducts.

```
FIND ESSENTIALS(PI, E)
E' = {}; n = |PI|; i = 0; j = 0;
do while i ≤ n
    {pi:=PI[i]
    if MINUS(pi, PI ∪ E) ≠ {} then {E'[j]:=pi
        j:=j + 1}
    i:=i + 1}
```

```
FIND INESSENTIALS(PI, E)
I = {}; n = |PI|; i = 0; j = 0;
do while i ≤ n
if MINUS(E, PI[i]) = {}
    then {I[j]:=PI[i]
    j := j + 1}
    i:=i + 1
return I
```

The next routine is performed when reductions 1–3 do not affect the set of primes at that stage. The user may request to control the removal and inclusion of subproducts into the cover. The system must then "remember" which operation was performed, so that the other may be attempted upon backtracking.

```
DIRECT SEARCH(PI)
if PI = {pi} then return pi
else {
    if no backtracking then { S:=PI[1]
        if query? = TRUE then action:=QUERY();
        else action:=REMOVE;
```

```
    if action=REMOVE then { STACK:=PUSH(S,FORCE);
    return S }
    else { STACK:=PUSH(S,REMOVE);
    return ∅ }
else { next:=POP(STACK);
if next.action =FORCE then return next.s;
else return ∅
}
```

Description

To generate all minimal covers, start by finding all essential primes at the root of a search tree T. Next, direct the search by either

1. arbitrarily removing a prime P_i, or

2. forcing P_i to be relatively essential.

At each successive level of the search tree, find those primes which are relatively essential at that particular level (given the absence of previously removed rules). Ensure that the addition of any new primes will not cause a forced rule to violate minimality requirements. Eliminate any inessential rules at that level and proceed to the next. Repeat this procedure until there are no remaining primes. To generate additional covers, backtrack to an unexplored branch.

We now illustrate the covering technique for disjunctive expressions, i.e., production rules. In Appendix A, the following set of prime rules is generated:

p_1: *seizure is myclonic, petit mal or pyschomotor.*

p_3: *if seizure is not pyschomotor then drug is depakene or tridione.*

p_6: *if seizure is not myclonic or petit mal then drug is tegretol.*

p_7: *if seizure is not pyschomotor and drug is not tridione then dangerous reaction is hepatic failure.*

p_8: *drug is depakene, tegretol or tridione.*

p_{10}: *if drug is not tegretol or tridione then dangerous reaction is hepatic failure.*

p_{11}: *dangerous reaction is bone marrow depression or drug is depakene or tridione.*

p_{12}: *if drug is not tridione then dangerous reaction is bone marrow depression.*

The prime implicant chart for this set of rules is shown in Table 5.6:

Table 5.6: Initial prime implicant chart for production rules

Seizures			Drugs			Dangerous reactions	
myclonic	petit mal	psychomotor	depakene	tegretol	tridione	bone marrow depression	hepatic failure
X	X	X					
		X	X		X		
X	X			X			
		X			X		X
			X	X	X		
X	X					X	
				X	X		X
			X		X	X	
					X	X	X

The covering algorithm finds $\{p_2\}$ to be the set of essentials. After removing p_2, p_1 is removed as the covering algorithm selects the next prime in order. P_1 is not the best choice that could have been made. By inspection, p_2 and p_3 can produce p_1 and p_4; consequently, neither p_1 nor p_4 should be forced into the intermediate cover. Another expression, p_3 is removed and the first minimal cover is produced.

Other minimal covers can be generated by backtracking, forcing p_3 into the intermediate cover and proceeding to remove other expressions as needed. The complete set of minimal covers for P is given as

$$C = \{\{p_2, p_4, p_8\}, \{p_2, p_3, p_7, p_8\}, \{p_2, p_3, p_6, p_7\}, \{p_2, p_3, p_4, p_6\}\}.$$

Determining whether a set of rules cover another rule is to subtract successively each rule from the one being tested. If a null value is not obtained, then there is no complete covering. As several forms of a prime may be present, the system must also prevent relatively essential

rules from violating minimality constraints. The depth-first search with backtracking used to generate C is shown in Figure 5.1.

5.4 Discussion

As previously mentioned, arbitrarily choosing primes for inclusion or exclusion in a possible cover may not be the most efficient approach. One alternative is to suggest candidate expressions and allow users to make the final determination. Another might employ a *greedy heuristic* to choose those primes thought most likely to be in a minimal cover. Such a heuristic might select those primes which represent the largest portion of the attribute signature. This approach is motivated by the fact that the prime which holds the greatest amount of the signature may be in the cover. At each level of the search, the next largest prime would be included in the cover. This would select the most general rules first and then could proceed to locate more specific rules. Other heuristics could also be employed. Baker [3] presents heuristic algorithms for the weighted set covering algorithm.

Several researchers have studied the behavior of set covering algorithms. Lifschitz and Pittel [25] analyze an entire class of set-covering algorithms which employ branching when no other reduction is feasible. Specifically, they study the computation time of these algorithms as a function of the number of steps needed to find a minimum cover. Stone and Sipala [54] analyze algorithms for depth-first search of binary trees. Their research provides insight into why some instances of NP-complete problems are solvable with a low average complexity.

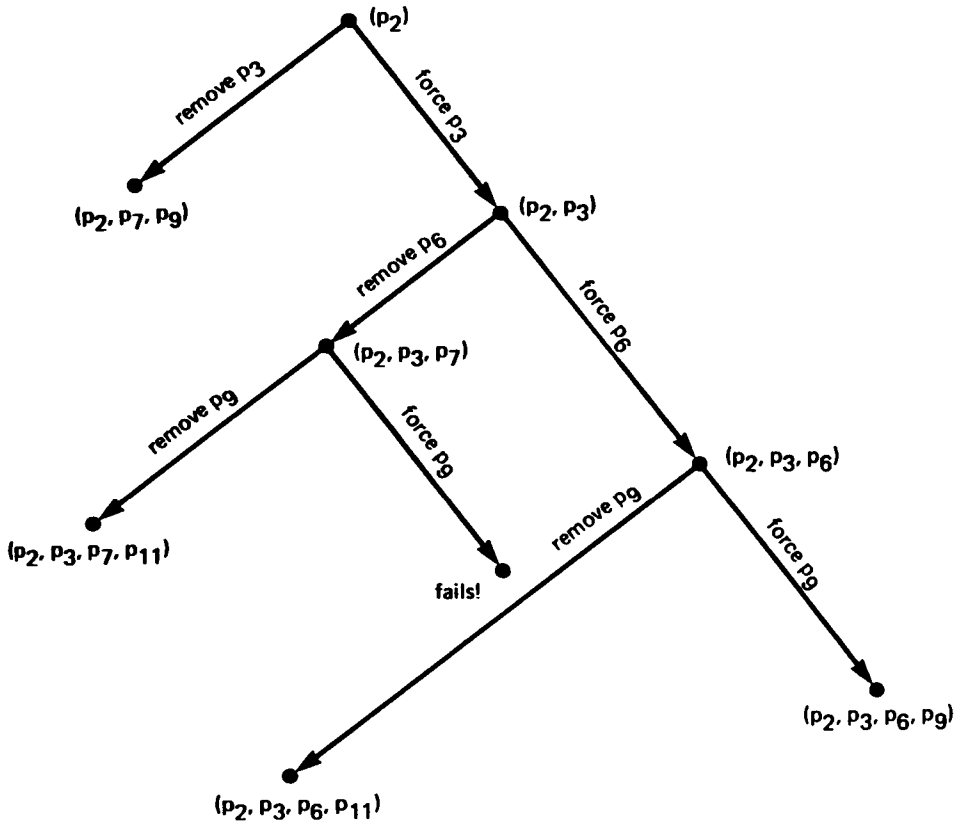

Figure 5.1: Depth-first search with backtracking

Chapter 6

Conclusions

We have generalized consensus methods to internally disjunctive production rules. Using Tison's method and the incremental algorithm, the logical consequences of a set of internally disjunctive production rules can be generated efficiently as new rules are added. Thus, all possible inferences from a set of production rules can be generated during the early stages of knowledge base design, assisting the validation and verification processes. More importantly, the effects of new rules upon the entire knowledge base can be studied in a reasonable amount of time as the computational requirements of the incremental consensus algorithm depends upon the number of attributes in the new rule and the number of rules already present. Consequently, a new rule with few attributes could be tested without too much difficulty.

A production rule may contain vital information in the form of its orientation. Consensus methods should retain this information. We have established a form of oriented consensus for use with internally disjunctive production rules. This orientation information is utilized by consensus to produce the most concise, yet expressive production rules. We have briefly examined methods to generate minimal sets of maximal subproducts. Most importantly, we have studied the behavior of oriented consensus and devised techniques for eliminating those consensi which are not narrow. This represents the primary contribution of this research to Computer Science. We conclude with a few comments about possible applications of

oriented consensus and some suggestions for future research directions.

6.1 Possible applications for oriented consensus

An immediate area for the application of oriented consensus is as a validation procedure for internally disjunctive production rules. Many commercially available expert system shells do not allow internal disjunctions within production rules. Therefore, several rules must be formulated when a single rule containing internal disjunctions would best suit the domain. Such a cumbersome procedure might also introduce errors into a knowledge base as the resultant set of rules might be incomplete. Furthermore, there is no systematic approach to generating the consequences of a set of production rules in most commercial products; presently, verification is a strictly manual procedure.

Michalski [32,33] has employed internally disjunctive expressions in inductive learning experiments. It is conceivable that oriented consensus could augment learning techniques which employ internally disjunctive expresssions to describe problem domains. Oriented consensus could serve to determine whether production rules supplied by experts are compatible with those produced by inductive methods. Oriented consensus methods could also serve as a unifying framework for machine learning systems.

The IPIA algorithm of Kean and Tsiknis [22] was motivated by a research problem in truth maintenance. Reiter and de Kleer [45] defined a *clause management system* (CMS) which used the prime implicates of a datum as the minimal sets of support needed to believe that datum. They raised the problem of efficiently generating the prime implicates of these data as new information entered the system. They suggested that existing consensus methods be studied to determine whether any method could incrementally generate the prime implicates needed by the CMS. Tsiknis and Kean [57] show how the CMS can be employed for diagnostic reasoning.

6.2 Further research

Several extensions to oriented consensus for internally disjunctive rules are possible. First, variables could be allowed in the production rules. Rules with variables might produce an infinite set of prime rules. Steps would need to be taken to avoid this occurrence. Further consideration of orientation could be given in covering the set of prime rules. We have discussed one approach: ignoring orientation altogether, allowing logically equivalent primes to cover the set of all possible primes. Another approach might preserve orientation completely, perhaps permitting several logically equivalent primes with different orientations to reside in the minimal cover. Clearly, the first approach may be inadequate as vital orientation information could be lost, but the second approach appears to be too costly.

One troubling question concerning this research is whether our notion of orientation is useful. We have not implemented the results on oriented consensus; therefore, we have not established its utility. Furthermore, the problem of identifying the narrow primes after blocking needs further study.

BIBLIOGRAPHY

Bibliography

[1] Andrews, P. B., "Theorem proving via general matings," *Journal of the ACM*, Vol. 28, No. 2, 1981, 193–214.

[2] Avis, D. "A note on some computationally difficult set covering problems," *Mathematical Programming*, Vol. 18, pp 138–145, 1980.

[3] Baker, E. K. "Efficient heuristic algorithms for the weighted set covering problem," *Computers & Operations Research*, Vol. 8, No. 4, pp. 301–310, 1981.

[4] Bartlett, K. A. *et al.*, "Multilevel logic minimization using implicit don't cares," *IEEE Transactions on Computer-Aided Design*, Vol. CAD-7, No. 6, 1988.

[5] Batni, R. P., J. D. Russell and C. R. Kime, "An efficient algorithm for finding an irredundant set cover," *Journal of the ACM*, Vol. 21, No. 3, pp. 351–355, 1974.

[6] Bellmore, M. and H. D. Ratliff, "Set covering and involutary bases," *Management Sciences*, Vol. 18, pp. 194–206, 1971.

[7] Bibel, W., "On matrices with connections," *Journal of the ACM*, Vol. 28, No. 4, 1981, 633–645.

[8] Bibel, W., *Automated Theorem Proving*, Friedr. Vieweg & Sohn, Braunschweig, West Germany, 1982.

[9] Biswas, N. N., "Minimization of Boolean functions," *IEEE Transactions on Computers*, Vol. C-20, pp. 925–929, 1971.

[10] Brayton, R. K., "Factoring logic functions," *IBM Journal of Research and Development*, Vol. 31, No. 2, pp. 187–198, 1987.

[11] Breitbart, Y. and K. Vairavan, "The computational complexity of a class of minimization algorithms for switching functions," *IEEE Transactions on Computers*, Vol. C-28, No. 12, 1978.

[12] Breuer, M. A., "Simplification of the covering problem with application to Boolean expressions," *Journal of the ACM*, Vol. 17, No. 1, pp. 166–181, 1970.

[13] Chandra, A. K. and G. Markowsky, "On the number of prime implicants," *Discrete Mathematics*, Vol. 24, pp. 7–11, 1978.

[14] Clocksin, W. F. and C. S. Mellish, *Programming in Prolog*, Springer-Verlag, Berlin, 1981.

[15] Cohen, P. J., *Set theory and the continuum hypothesis*, W. A. Benjamin, New York, 1966.

[16] Dagenais, M. R., V. K. Agarwal, and N. C. Rumin, "McBoole: A new procedure for exact logic minimization," *IEEE Transaction on Computer-Aided Design*, Vol. CAD-5, No. 1, pp. 229–237, 1986.

[17] Garey, Michael R. and David S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.

[18] Ghazala, M. J. "Irredundant disjunctive and conjunctive forms of a Boolean function," IBM Journal of Research and Development Vol. 1, pp. 171–176, April 1957.

[19] Gimpel, J. F., "A reduction technique for prime implicant tables," *IEEE Transactions on ELectronic Computers*, pp. 535–541.

[20] Hong, S. J., R. G. Cain and D. L. Ostapko, "MINI: A heuristic approach for logic minimization," *IBM Journal of Research and Development*, pp. 443–458, 1974.

[21] Hwa, H. R., "A method for generating prime implicants of a Boolean expression," *IEEE Transactions on Computers*, Vol. C-23, pp. 637–644, 1974.

[22] Kean, Alex and George Tsiknis, *An incremental method for generating prime implicants/implicates*, Technical report 88–16, Department of Computer Science, The University of British Columbia, July 1988.

[23] Kohavi, Z., *Switching and Finite Automata Theory*, 2nd ed., McGraw-Hill, New York, 1978.

[24] Lee, R. C. T. "A completeness theorem and a computer program for finding theorems derivable from given axioms," Ph.D. dissertation, University of California, Berkeley, 1967.

[25] Lifschitz, V. and B. Pittel, "The worst and most probable performance of a class of set-covering algoritms," *SIAM Journal of Computing*, Vol. 12, No. 2, pp. 329–346, 1983.

[26] Loui, M. C. and G. Bilardi, "The correctness of Tison's method for generating prime implicants," UILU-ENG 82-2218, Coordinated Science Laboratory, University of Illinois at Champaign–Urbana, February 1982.

[27] Luccio, F., "A method for the selection of prime implicants," *IEEE Transactions on Electronic Computers*, Vol. EC-15, No. 2, pp. 205–212, 1966.

[28] Marek, W. and Z. Pawlak, "Information storage and retrieval systems: mathematical foundations," *Theoretical Computer Science*, Vol. 1, 331–354, 1976.

[29] McCluskey, E. J. Jr., "Minimization of Boolean functions," *Bell System Technical Journal*, Vol. 35, pp. 1417–1444, 1957.

[30] McCluskey, E. J. Jr., "Detection of group invariance or total symmetry of a Boolean function," *Bell System Technical Journal*, Vol. 35, pp. 1445–1453, 1957.

[31] McMullen, C. and J. Shearer, "Prime implicants, minimum covers, and the complexity of logic simplification," *IEEE Transactions on Computers*, Vol. C-35, No. 8, pp. 761–762, 1986.

[32] Michalski, R. S., "Variable-valued logic and its applications to pattern recognition and artificial intelligence," in Rine, D., Ed., *Multiple-Valued Logic and Computer Science*, North-Holland, Amsterdam, 1975.

[33] Michalski, R. S., "Pattern recognition as rule-guided inductive inference," *IEEE Transactions on pattern analysis and machine intelligence*, Vol. PAMI-2, No. 4, 1980, pp. 349–361.

[34] Michalski, R. S., "A Theory and methodology of inductive learning," *Artificial Intelligence*, Vol. 20, No. 2, 1983, pp. 111–161.

[35] Miller, R. E. *Switching theory*, Vol. 1, John Wiley & Sons, New York, 1965.

[36] Minicozzi, E. and R. Reiter, "A note on linear resolution strategies in consequence-finding," *Artificial Intelligence*, Vol. 3, No. 2, pp. 175–180, 1972.

[37] Muroga, S., "Logic design of VLSI electronic circuits—tutorial," in *VLSI: Algorithms and Architectures*, P. Bertolazzi and F. Luccio (eds.), Elsevier Science Publishers, New York, 1985.

[38] Murray, N. V., "Completely nonclausal theorem proving", *Artifical Intelligence*, Vol. 18, No. 1, 1982, 67–85.

[39] Nagle, H. T. Jr., B. D. Carroll and J. D. Irwin, *An Introduction to Computer Logic*, Printice-Hall, Englewood Cliffs, NJ, 1975.

[40] Nelson, R. J. "Simplest normal truth functions," *Journal of Symbolic Logic*, Vol. 20, pp. 105–108, 1955.

[41] Pawlak, Z. "Information systems—theoretical foundations," *Information Systems*, Vol. 6, No. 3, pp. 205–218, 1981.

[42] Petrick, S. R., "A direct determination of the irredundant forms of a Boolean function from the set of prime implicants," Air Force Cambridge Research Center, Bedford, Mass., Tech. Dept. 56-110, April 1956.

[43] Quine, W. V. "The problem of simplifying truth functions," *American Mathematical Monthly*, Vol. 59, pp. 521–531, 1952.

[44] Quine, W. V. "A way to simplify truth functions", *American Mathematical Monthly*, Vol. 62, pp. 627–631, Nov. 1955.

[45] Reiter, R. and J. de Kleer, "Foundations of assumption-based truth maintenance: A preliminary report," *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, Washington, August 1987, 183–188.

[46] Rhyne, V. T., *et al.*, "A new technique for the fast minimization of switching functions," *IEEE Transacations on computers*, Vol. C-26, No. 8, pp. 757–764, 1977.

[47] Robinson, J. A., "A machine-oriented logic based on the resolution principle," *Journal of the ACM*, Vol. 12, No. 1, 1965, 23–41.

[48] Roth, J. P. "Algebraic topological methods for the synthesis of switching systems I," *Transactions of the American Mathematical Society*, Vol. 88, pp. 301–326, 1958.

[49] Rudell, R. L. and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," IEEE Transactions on Computer-Aided Design, Vol. CAD-6, No. 5, 1987.

[50] Salkin, H. M. and R. D. Koncal, "Set covering by an all integer algorithm: computational experience," *Journal of the ACM*, Vol. 20, No. 2, pp. 189–193, 1973.

[51] Slagle, J. R., C. L. Chang and R. C. T. Lee, "Completeness theorems for semantic resolution in consequence finding," *Proceedings of the first International Joint Conference on Artificial Intelligence*, Washington, D.C., pp. 281–285, 1969.

[52] Slagle, J. R., C. L. Chang and R. C. T. Lee, "A new algorithm for generating prime implicants," *IEEE Transactions on Computers*, Vol. C-19, No. 4, April 1970.

[53] Stickel, M.E., "A complete unification algorithm for associative–commutative functions," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Tbilisi, Georgia, U.S.S.R., Septemeber 1975, 71–76.

[54] Stone, H. S. and P. Sipala, "The average complexity of depth-first search with backtracking," *IBM Journal of Research and Development*, Vol. 30, No. 3, pp. 242–258, 1986.

[55] Sureschander, "Minimization of switching functions—a fast technique", *IEEE Transactions on Computers*, Vol. C-24, pp. 753–756, 1975.

[56] Tison, P., "Generalized consensus theory and application to the minimization of Boolean functions," *IEEE Transactions on Electronic Computers*, Vol. E-16, No. 4, 1967, pp. 446–456.

[57] Tsiknis, G. and A. Kean, *Clause management systems*, Technical report 88–21, Department of Computer Science, University of British Columbia, October 1988.

[58] Wells, M. A. *Elements of Combinatorial Computing*, Pergamon Press, New York, 1971.

APPENDICES

Appendix A

An example of iterative consensus on production rules

```
Here is the original set of rules:

seizure is myclonic, petit_mal or pyschomotor.

if seizure is pyschomotor then drug is tegretol.

if seizure is not pyschomotor then drug is depakene or tridione.

if drug is tegretol then dangerous_reaction is bone_marrow_depression.

if drug is depakene then dangerous_reaction is hepatic_failure.


Rule(2) is weaker than Rule(6).  It has been removed.

Rule(2) and Rule(1) have been paired to get Rule(6):
if seizure is not myclonic or petit_mal then drug is tegretol.

Rule(5) and Rule(3) have been paired to get Rule(7):
if drug is not tridione and seizure is not pyschomotor
     then dangerous_reaction is hepatic_failure.

Rule(6) and Rule(3) have been paired to get Rule(8):
drug is depakene, tegretol or tridione.

Rule(6) and Rule(4) have been paired to get Rule(9):
if seizure is not myclonic or petit_mal
     then dangerous_reaction is bone_marrow_depression.

Rule(5) is weaker than Rule(10).  It has been removed.

Rule(7) and Rule(6) have been paired to get Rule(10):
if drug is not tegretol or tridione then dangerous_reaction is hepatic_failure.
```

```
Rule(4) is weaker than Rule(11).  It has been removed.

Rule(8) and Rule(4) have been paired to get Rule(11):
if drug is not depakene or tridione
     then dangerous_reaction is bone_marrow_depression.

Rule(9) and Rule(3) could be paired.  The result is equivalent to Rule(11).
if drug is not depakene or tridione
     then dangerous_reaction is bone_marrow_depression.

Rule(9) and Rule(7) have been paired to get Rule(12):
if drug is not tridione
     then dangerous_reaction is bone_marrow_depression or hepatic_failure.

Rule(10) and Rule(3) could be paired.  The result is equivalent to Rule(7).

Rule(11) and Rule(6) could be paired.  The result is equivalent to Rule(9).

Rule(11) and Rule(10) could be paired. The result is equivalent to Rule(12).

Rule(12) and Rule(6) could be paired.  The result is weaker than Rule(9).


Here is the set of prime rules:

Rule(1):  seizure is myclonic, petit_mal or pyschomotor.

Rule(3):  if seizure is not pyschomotor then drug is depakene or tridione.

Rule(6):  if seizure is not myclonic or petit_mal then drug is tegretol.

Rule(7):  if drug is not tridione and seizure is not pyschomotor
          then dangerous_reaction is hepatic_failure.

Rule(8):  drug is depakene, tegretol or tridione.

Rule(9):  if seizure is not myclonic or petit_mal
          then dangerous_reaction is bone_marrow_depression.

Rule(10):  if drug is not tegretol or tridione
           then dangerous_reaction is hepatic_failure.

Rule(11):  if drug is not depakene or tridione
           then dangerous_reaction is bone_marrow_depression.

Rule(12):  if drug is not tridione
           then dangerous_reaction is bone_marrow_depression or hepatic_failure.
```

Appendix B

An example of Tison's method on production rules

```
Here is the original set of rules:

seizure is myclonic, petit_mal or pyschomotor.

if seizure is pyschomotor then drug is tegretol.

if seizure is not pyschomotor then drug is depakene or tridione.

if drug is tegretol then dangerous_reaction is bone_marrow_depression.

if drug is depakene then dangerous_reaction is hepatic_failure.


Pairing with respect to drug


Rule(4) and Rule(2) have been paired to get Rule(6):
if seizure is pyschomotor then dangerous_reaction is bone_marrow_depression.

Rule(5) and Rule(3) have been paired to get Rule(7):
if drug is not tridione and seizure is not pyschomotor
     then dangerous_reaction is hepatic_failure.


Pairing with respect to seizure


Rule(2) is weaker than Rule(8).  It has been removed.

Rule(2) and Rule(1) have been paired to get Rule(8):
if seizure is not myclonic or petit_mal then drug is tegretol.

Rule(6) is weaker than Rule(9).  It has been removed.
```

```
Rule(6) and Rule(1) have been paired to get Rule(9):
if seizure is not myclonic or petit_mal
     then dangerous_reaction is bone_marrow_depression.

Rule(4) is weaker than Rule(10).  It has been removed.

Rule(6) and Rule(3) have been paired to get Rule(10):
dangerous_reaction is bone_marrow_depression or drug is depakene or tridione.

Rule(8) and Rule(3) have been paired to get Rule(11):
drug is depakene, tegretol or tridione.

Rule(5) is weaker than Rule(12).  It has been removed.

Rule(8) and Rule(7) have been paired to get Rule(12):
if drug is not tegretol or tridione then dangerous_reaction is hepatic_failure.

Rule(9) and Rule(3) could be paired.  The result is equivalent to Rule(10).
dangerous_reaction is bone_marrow_depression or drug is depakene or tridione.

Rule(9) and Rule(7) have been paired to get Rule(13):
if drug is not tridione
     then dangerous_reaction is bone_marrow_depression or hepatic_failure.


Here is the set of prime rules:

Rule(1):  seizure is myclonic, petit_mal or pyschomotor.

Rule(3):  if seizure is not pyschomotor
          then drug is depakene or tridione.

Rule(7):  if drug is not tridione and seizure is not pyschomotor
          then dangerous_reaction is hepatic_failure.

Rule(8):  if seizure is not myclonic or petit_mal then drug is tegretol.

Rule(9):  if seizure is not myclonic or petit_mal
          then dangerous_reaction is bone_marrow_depression.

Rule(10):  dangerous_reaction is bone_marrow_depression
           or drug is depakene or tridione.

Rule(11):  drug is depakene, tegretol or tridione.

Rule(12):  if drug is not tegretol or tridione
           then dangerous_reaction is hepatic_failure.

Rule(13):  if drug is not tridione
           then dangerous_reaction is bone_marrow_depression or hepatic_failure.
```

Appendix C

Covering a set of prime rules

```
Here is the set of prime rules.

Rule(1):  seizure is myclonic, petit_mal or pyschomotor.

Rule(3):  if seizure is not pyschomotor then drug is depakene or tridione.

Rule(7):  if drug is not tridione and seizure is not pyschomotor
          then dangerous_reaction is hepatic_failure.

Rule(8):  if seizure is not myclonic or petit_mal then drug is tegretol.

Rule(9):  if seizure is not myclonic or petit_mal
          then dangerous_reaction is bone_marrow_depression.

Rule(10):  dangerous_reaction is bone_marrow_depression
           or drug is depakene or tridione.

Rule(11):  drug is depakene, tegretol or tridione.

Rule(12):  if drug is not tegretol or tridione
           then dangerous_reaction is hepatic_failure.

Rule(13):  if drug is not tridione
           then dangerous_reaction is bone_marrow_depression or hepatic_failure.


Interactively direct the search? no.


Essentials at this stage:
if seizure is not myclonic or petit_mal then drug is tegretol.


Removing . . . if seizure is not pyschomotor then drug is depakene or tridione.

Essentials at this stage:
if drug is not tridione and seizure is not pyschomotor
```

```
     then dangerous_reaction is hepatic_failure.
dangerous_reaction is bone_marrow_depression or drug is depakene or tridione.


Minimum set of rules 1:
if seizure is not myclonic or petit_mal then drug is tegretol.
if drug is not tridione and seizure is not pyschomotor
     then dangerous_reaction is hepatic_failure.
dangerous_reaction is bone_marrow_depression or drug is depakene or tridione.


*******************************


Find another cover? yes.

Forcing . . . if seizure is not pyschomotor then drug is depakene or tridione.

Removing . . . if drug is not tridione and seizure is not pyschomotor
               then dangerous_reaction is hepatic_failure.

Essentials at this stage:
if drug is not tegretol or tridione then dangerous_reaction is hepatic_failure.


Removing . . . if seizure is not myclonic or petit_mal
               then dangerous_reaction is bone_marrow_depression.

Essentials at this stage:
dangerous_reaction is bone_marrow_depression or drug is depakene or tridione.


Minimum set of rules 2:
if seizure is not pyschomotor then drug is depakene or tridione.
if seizure is not myclonic or petit_mal then drug is tegretol.
if drug is not tegretol or tridione then dangerous_reaction is hepatic_failure.
dangerous_reaction is bone_marrow_depression or drug is depakene or tridione.


*******************************


Find another cover? yes.

Forcing . . . if seizure is not myclonic or petit_mal
              then dangerous_reaction is bone_marrow_depression.

Minimum set of rules 3:
if seizure is not myclonic or petit_mal
     then dangerous_reaction is bone_marrow_depression.
if seizure is not pyschomotor then drug is depakene or tridione.
```

```
if seizure is not myclonic or petit_mal then drug is tegretol.
if drug is not tegretol or tridione then dangerous_reaction is hepatic_failure.


********************************


Find another cover? yes.

Forcing . . . if drug is not tridione and seizure is not pyschomotor
              then dangerous_reaction is hepatic_failure.

Removing . . . if seizure is not myclonic or petit_mal
               then dangerous_reaction is bone_marrow_depression.

Essentials at this stage:
dangerous_reaction is bone_marrow_depression or drug is depakene or tridione.


A forced rule is not relatively essential.
This can't be a minimal cover!

Forcing . . . if seizure is not myclonic or petit_mal
              then dangerous_reaction is bone_marrow_depression.

Minimum set of rules 4:
if seizure is not myclonic or petit_mal
     then dangerous_reaction is bone_marrow_depression.
if drug is not tridione and seizure is not pyschomotor
     then dangerous_reaction is hepatic_failure.
if seizure is not pyschomotor then drug is depakene or tridione.
if seizure is not myclonic or petit_mal then drug is tegretol.


********************************


Find another cover? yes.

No additional minimal covers!
```

VITA

Lloyd Frederick Arrowood III was born in Chattanooga, Tennessee on October 2, 1959. He attended elementary and secondary schools in that city and Johnson City, Tennessee. He graduated from Science Hill High School in June 1976. He was awarded a Bachelor of Arts degree from The University of Tennessee in June 1980. Immediately upon graduation, he began work at Martin Marietta Energy Systems (formerly known as Union Carbide Corporation) in Oak Ridge, Tennessee.

While employed at Martin Marietta, he entered The University of Tennessee as a part-time student to pursue a Master's degree in Computer Science. Upon graduation, he plans to continue his employment at Martin Marietta.