MINIMIZATION OF MULTIPLE-VALUED RELATIONS

by

Yosinori Watanabe

Memorandum No. UCB/ERL M91/48

24 May 1991

MINIMIZATION OF MULTIPLE-VALUED RELATIONS

by

Yosinori Watanabe

Memorandum No. UCB/ERL M91/48

24 May 1991

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Minimization of Multiple-Valued Relations

Yosinori Watanabe

University of California
Berkeley, CA 94720

Department of Electrical Engineering
and Computer Sciences

Abstract

A multiple-valued relation is a relation in which the input variables can assume more than two discrete values. Multiple-valued relations arise quite naturally in many contexts. Using characteristic functions to represent relations, we can handle the problem of minimizing multiple-valued relations as a generalization of the conventional minimization problem of regular functions. Our approach is based on a state-of-the-art paradigm for the two level minimization of regular functions. We clarify some special properties of relations, in contrast to functions, which must be carefully considered in realizing a high quality procedure for solving the minimization problem. An efficient heuristic method to find an optimal sum-of-products representation for a multiple-valued relation is proposed and implemented in the program GYOCRO. It uses multiple-valued decision diagrams (MDD's) to represent the characteristic functions for the relations. Experimental results are presented and compared with previous exact and heuristic Boolean relation minimizers to demonstrate the effectiveness of the proposed method.

R. K. Brayton

Professor Robert K. Brayton
Research Advisor

Acknowledgements

I am indebted to my research advisor, Professor Robert K. Brayton, for his effective guidance and consistent encouragement during this research work. His door was always open to me for discussion. Professor Brayton's attitude for research, well-balanced both in the theoretical and in the practical aspects of the problem, is admirable, and has always guided me to the right direction. This project would not have been possible without his continuous guidance, encouragement, and friendship.

I sincerely thank Professor Alberto Sangiovanni-Vincentelli for his interest in my research and for taking the burden of being a second reader of my thesis.

Professor Fabio Somenzi, University of Colorado at Boulder, has been always supportive to my research. He gladly provided me both the source and the binary codes of his exact minimizer for Boolean relations. Whenever I asked a question on his method by electronic mail, he replied me with a perfect answer in 30 minutes. I gratefully acknowledge his support.

Many members of Berkeley CAD group gave me important ideas. I owe to Abhijit Ghosh, Bill Lin, and Hervé Touati the fruitful discussions we had together. Special thanks are to Abhijit Ghosh for his providing me the binary code of his heuristic minimizer of Boolean relations.

This research is supported in part by the National Science Foundation and the Defensive Advanced Research Projects Agency under contract number NSF/DARPA-MIP-871-9546, and grants from AT&T, DEC, IBM, Intel, and Motorola. I acknowledge their support.

Finally, and most emphatically, I want to thank my fiancée, Mika Sugawara. Mika's enormous love and encouragement made my mind to stay at Berkeley to continue the research, and has always been a driving force for me to devote myself to the work. I intend to spend many years proving to her that I am worth the effort.

Contents

List of Figures

Chapter 1

Introduction

1.1 Minimization of Multiple-Valued Relations

Research in logic minimization has been active over the past 40 years. Initial research was directed towards developing techniques to produce an optimum sum-of-products representation of a Boolean function [13] and has evolved toward heuristic approaches for designing programmable logic arrays (PLA's) [2, 9]. In the past 10 years, an increasing amount of research has been in multiple-level logic minimization (e.g. [3, 6]). One of the key features in the minimization process is the use of don't care sets and incompletely specified functions. More recently, a theory of Boolean relations was introduced [4], in which it was shown that don't cares of the traditional kind are insufficient to capture the complete freedom for optimizing multiple output functions. It was claimed that this flexibility is fully described with Boolean relations.

In parallel with this activity has been the minimization of multiple-valued functions [11, 16], in which variables can assume more than two discrete values. The significance of this problem is in its wide applications in areas such as PLA optimization [14] and state assignment for finite state machines [7].

This paper is concerned with the minimization of multiple-valued relations, where the input variables can take multiple values and the outputs are binary variables. A multiple-valued relation $R \subseteq D \times B^m$ is a relation over the two sets D and B^m. The output set B^m is the m dimensional Boolean space and the input set D is the Cartesian product of n sets $D_1 \times \cdots \times D_n$, where D_i is a set of P_i values, $D_i = \{0, 1, \cdots P_i - 1\}$, and P_i is a positive integer. Note that this includes the case where the outputs may also

be multiple-valued since we can encode each multiple-valued output with a set of one-hot variables. In addition, binary-valued input variables are a special case of multiple-valued variables. The image of a minterm[1] $\mathbf{x} \in D$ by R is defined as $r(\mathbf{x}) = \{\mathbf{y} \in B^m \mid (\mathbf{x}, \mathbf{y}) \in R\}$ and may consist of multiple minterms of B^m. Our objective is to find a least cost implementation (completely specified function) of a multiple-valued function $f : D \rightarrow B^m$ compatible with R, i.e. $\forall \mathbf{x} \in D : f(\mathbf{x}) \in r(\mathbf{x})$, if such a function exists. In fact, a compatible function exists if and only if $r(\mathbf{x})$ is not empty for every $\mathbf{x} \in D$. A relation satisfying this condition is called a well-defined relation. With this condition satisfied, R can be represented as a one-to-many multiple-valued mapping $r : D \rightarrow B^m$. Furthermore, if $r(\mathbf{x})$ can be represented as a single cube for every $\mathbf{x} \in D$, the mapping r is an incompletely specified function, i.e. one with don't cares. In this case, the minimization problem is reduced to the conventional minimization of multiple-valued functions.

We represent a relation $R \subseteq D \times B^m$ as its characteristic function $R : D \times B^m \rightarrow B$ such that $R(\mathbf{x}, \mathbf{y}) = 1$ if and only if $(\mathbf{x}, \mathbf{y}) \in R$. Hereafter, we make no distinction between a relation and its characteristic function. The characteristic functions are represented by multiple-valued decision diagrams (MDD's) [18]. An MDD is a data structure to handle multiple-valued functions which employs binary decision diagrams (BDD's) [5] as the internal representation. MDD's have the property of being canonical, and thus there is a one-to-one correspondence between a relation and an MDD for the characteristic function under a given order of the variables of D and B^m. We can complete set operations, such as complementation, containment checking, and cofactoring, very efficiently using MDD's for fairly large functions in many cases.

Multiple-valued relations arise in many contexts [1, 4, 12]. In fact, we should emphasize that in many minimization problems, the situation is naturally formulated as a multiple-valued relation. For example, the behavior of a completely specified finite state machine is given by a function $F : I \times S \times S \times O \rightarrow B$ such that $F(i, p, n, o) = 1$ if and only if the input i and the present state p causes the machine to evolve to the next state n and produce the output o. F is a multiple-valued relation with the input set $I \times S$ and the output set $S \times O$. For a given initial state, a set of equivalent states can be computed as a function $E : S \times S \rightarrow B$ such that $E(n, \tilde{n}) = 1$ if and only if n and $\tilde{n}$ are equivalent. Since a state can be mapped to any of the equivalent states of the next state, we have the possibility of

[1] In this paper, we make no distinction between a minterm and an element of a set.

implementing a more compact machine using the equivalent states. Namely, our objective is to find a least cost machine compatible with the function $\tilde{F} : I \times S \times S \times O \rightarrow B$ such that $\tilde{F}(i,p,n,o) = 1$ if and only if either $F(i,p,n,o) = 1$ or there exists a state $\tilde{n}$ for which $F(i,p,\tilde{n},o) = 1$ and $E(n,\tilde{n}) = 1$. $\tilde{F}$ can be easily computed using MDD's. $\tilde{F}$ provides the complete family of finite state machines equivalent to the original machine under the equivalent states. Similarly, $\tilde{F}$ can be extended to include invalid states, which are defined as a set of states not reachable from some initial set of states. Other problems of synthesizing finite state machines, e.g. optimization of cascaded machines, can be formulated in a similar way with multiple-valued relations.

Multiple-valued relations can be used even in the context of Boolean minimization. Given an incompletely specified Boolean function represented as the on-set $f : B^n \rightarrow B^m$ and don't-care sets $d : B^n \rightarrow B^m$, the problem of finding a least cost implementation of this function is formulated as the minimization problem for a relation $F : B^n \times B^m \rightarrow B$ given by $F(\mathbf{x},\mathbf{y}) = \prod_{j=1}^{m}((y_j \equiv f^{(j)}(\mathbf{x})) + d^{(j)}(\mathbf{x}))$, where y_j is the j-th output variable, $f^{(j)}$ and $d^{(j)}$ are the j-th function of f and d respectively, and $f \equiv g$ designates an XNOR operation between f and g. The formulation of Boolean minimization via a relation has an advantage of handling the care sets and the don't care sets in a uniform way. In practice, there exist cases where one can specify an incompletely specified function by an MDD of the characteristic function for a relation in a more compact form than representing the care set and the don't care set separately in a two level or a multiple-level form.

Therefore, for a given problem, we first formulate it using a multiple-valued relation R. Then the existence condition of functions compatible with R is checked. If there is no compatible function, the problem has no solution. Otherwise, we identify whether the relation can be represented as an incompletely specified function. If this is the case, we extract the care set and the don't care set from R to represent each in some required form, e.g. a sum-of-products form, and invoke a conventional function minimizer (e.g. [14]) to produce an optimal implementation. If, on the other hand, R cannot be expressed by an incompletely specified function, we need to minimize the relation, i.e. find a least cost implementation compatible with R.

1.2 Overview

Somenzi *et al.* have proposed and implemented an exact minimization procedure for binary-valued relations[17], with which an optimum sum-of-products representation compatible with the relation is obtained. The key is that all the prime cubes of all the compatible functions are generated and a binate covering problem is solved to find the best representation. However, since the method is exact, it is expensive to complete the procedure both in CPU time and memory space, so that only small examples can be handled.

Heuristic minimization of relations has been an open problem, even for binary-valued relations. Ghosh *et al.* [8] proposed and implemented an approach for binary-valued relations which makes use of test pattern generation techniques and heuristically finds a two level representation. The method is similar to two level minimizers for Boolean functions (e.g. [2]) in the sense that procedures analogous to expand, irredundant, and reduce are repeatedly applied as long as the cost decreases. However, unlike the most effective two level minimizers that consider multiple variables to be expanded or reduced simultaneously, only one variable is examined at a time. Thus the minimizer of [8] is more likely to get stuck at a bad solution. This drawback is fatal for the method since the simultaneous expansion of multiple variables implies the use of multiple faults, which could be very expensive to detect. Furthermore, the method is a direct application of ATPG methodology to this problem and little new theoretical analysis is provided to contrast some of the properties of relations, which may be useful in the minimization process, to those of regular functions.

We propose a heuristic procedure for the minimization problem of multiple-valued relations, which is based on a paradigm of the more advanced two level minimization techniques for regular functions. The goal of the procedure is to find a compatible representation with the minimum number of the product terms. The final solution is obtained from an initial representation by iterative improvement through reduce and expand procedures. In the reduce procedure, each cube is maximally reduced while still maintaining the compatibility of the representation. In the expand procedure, each cube is maximally expanded so that a maximal number of cubes in the current representation can be removed. We present, in each procedure, some special properties associated with relations not found in functions. These properties must be carefully accounted for while implementing a procedure that is effective in achieving high quality results. We have implemented these algorithms in a program called GYOCRO, a Japanese traditional tea with good taste, and provide

experimental evidence of their effectiveness.

This report is organized as follows. In Chapter 2, terminology is defined and a brief review of multiple-valued relations is provided. In Chapter 3, we describe how to identify whether a given relation is an incompletely specified function as well as a procedure that extracts the care sets and the don't care sets from the relation if it is a function. Chapter 4 presents the minimization procedure employed in GYOCRO in which technical details are described for each sub-procedure along with supporting theoretical analysis. Experimental results of the proposed method are presented in Chapter 5. Some potential modifications of the algorithms are also discussed. Chapter 6 concludes the paper.

Chapter 2

Functions, Mappings, and Relations

In this chapter, we describe the relation among functions, mappings, and relations, and clarify when we need a procedure that is capable of minimizing relations directly. We begin with the terminology.

2.1 Terminology

Definition 1 *A* **multiple-valued relation** R *is a subset of* $D \times B^m$. D *is called the* **input set** *of* R *and is the Cartesian product of* n *sets* $D_1 \times \cdots \times D_n$, *where* $D_i = \{0, \cdots, P_i - 1\}$ *and* P_i *is a positive integer.* D_i *provides the set of values that the* i*-th variable of* D *can assume.* B^m *designates a Boolean space*[1] *spanned by* m *variables, each of which can assume either* 0 *or* 1. B^m *is called the* **output set** *of* R. *The variables of the input set and the output set are called the* **input variables** *and the* **output variables** *respectively.* R *is* **well-defined** *if for every* $\mathbf{x} \in D$, *there exists* $\mathbf{y} \in B^m$ *such that* $(\mathbf{x}, \mathbf{y}) \in R$.

Definition 2 *For a given relation* R *and a subset* $A \subseteq D$, *the* **image** *of* A *by* R *is a set of minterms* $\mathbf{y} \in B^m$ *for which there exists a minterm* $\mathbf{x} \in A$ *such that* $(\mathbf{x}, \mathbf{y}) \in R$, *i.e.* $\{\mathbf{y} \mid \exists \mathbf{x} \in A : (\mathbf{x}, \mathbf{y}) \in R\}$. *The image is denoted as* $r(A)$. $r(A)$ *may be empty.*

Definition 3 *For a given relation* $R \subseteq D \times B^m$, *a multiple-valued function* $f : D \rightarrow B^m$ *is* **compatible** *with* R, *denoted* $f \prec R$, *if for every minterm* $\mathbf{x} \in D$, $f(\mathbf{x}) \in r(\mathbf{x})$. *Otherwise*

[1] In this paper, we make no distinction between a set and a space.

f is incompatible with R. Clearly, $f \prec R$ exists if and only if R is well-defined.

Definition 4 *For the i-th variable x_i of D, a* **literal** *of x_i is the characteristic function of a subset S_i of D_i, and is denoted as $x_i^{S_i}$. S_i may be empty. A* **product term** *p is a Boolean product of literals of all the variables of D. Thus p is the characteristic function of a subset of D.*

Definition 5 *For the j-th output function $f^{(j)}$ of $f : D \rightarrow B^m$, an* **algebraic sum-of-products expression** *of $f^{(j)}$ is a union of product terms such that the resulting characteristic function is equivalent to $f^{(j)}$. An algebraic sum-of-products expression of f is a set of algebraic sum-of-products expressions for all the output functions.*

Definition 6 *For an algebraic sum-of-products expression of a function $f : D \rightarrow B^m$, a* **cube** *is a product term p of the expression specified as a row vector with two parts, $c = [I(c)|O(c)]$, where $I(c) = [I(c)_1, \cdots, I(c)_n]$ and $O(c) = [O(c)_1, \cdots, O(c)_m]$. $I(c)$ and $O(c)$ are called the input part and the output part of c respectively. The i-th component of $I(c)$ represents a set of values contained in the i-th literal of p, and consists of P_i binary bits. Each bit is called a* **part**. *The k-th part of $I(c)_i$, $k \in \{0, \cdots, P_i - 1\}$, is 1 if the i-th literal of p contains the value k. It is 0 otherwise. For the output part, $O(c)_j = 0$ if p is not present in the algebraic expression of $f^{(j)}$. Otherwise, $O(c)_j = 1$. We denote $M(c)$ as a set of minterms of D contained in p. A set of cubes is called a* **representation.**

For two cubes c and d, c *contains* d if c has 1 for every part that d has 1. In addition, c *strictly contains* d if they are not equal.

Throughout the report, we show examples of representations, in which all the inputs are binary variables. For the sake of simplicity, we represent the input part of a cube c as a n-tuple $[I(c)_1, \cdots, I(c)_n]$ such that $I(c)_i$ takes 0 if the i-th literal of p takes a value 0, 1 if the i-th literal of p takes 1, and 2 if the literal takes both 0 and 1.

For a given representation $\mathcal{F}$, a function $f : D \rightarrow B^m$ is uniquely defined, where an algebraic expression of f is given by $\mathcal{F}$. Thus we say that a representation $\mathcal{F}$ is *compatible* with a relation R if the corresponding function f is compatible with R. Similarly, the image of $A \subseteq D$ by the representation $\mathcal{F}$ is the image of A by a relation F given by $F = \{(\mathbf{x}, \mathbf{y}) \in D \times B^m \mid \mathbf{y} = f(\mathbf{x})\}$.

Definition 7 *For a given relation R, a cube c is a* **candidate prime** *(or a c-prime) if there exists a function compatible with R in which c is a prime implicant.*

Definition 8 *For a given relation R and a compatible representation $\mathcal{F}$, a cube $c \in \mathcal{F}$ is prime relative to $\mathcal{F}$ (or* **relatively prime** *in $\mathcal{F}$) if for any cube $\tilde{c}$ which strictly contains c, a replacement of c with $\tilde{c}$ in $\mathcal{F}$ results in an incompatible representation with R. A representation $\mathcal{F}$ is* **relatively prime** *if $\mathcal{F}$ is compatible with R and every cube of $\mathcal{F}$ is relatively prime in $\mathcal{F}$.*

Note that if $c \in \mathcal{F}$ is relatively prime in $\mathcal{F}$ then c is a c-prime, but not the other way around. We distinguish the notion of primality between relations and regular functions, since in relations, the primality of a cube depends upon the other cubes of the representation in which the cube is present. This is not the case for functions.

Definition 9 *For a given relation R and a compatible representation $\mathcal{F}$, a cube $c \in \mathcal{F}$ is* **redundant** *in $\mathcal{F}$ if removal of c from $\mathcal{F}$ maintains the compatibility of the representation $\mathcal{F} - \{c\}$ with R. Otherwise c is irredundant. A representation $\mathcal{F}$ is* **irredundant** *if $\mathcal{F}$ is compatible with R and every cube of $\mathcal{F}$ is irredundant.*

2.2 Representations of Multiple-Valued Relations

Our objective is to find a least cost implementation of a function compatible with a given multiple-valued relation. There are several ways to represent relations. One way is to specify the image of $\mathbf{x}$, $r(\mathbf{x})$, for each minterm $\mathbf{x} \in D$, where $r(\mathbf{x})$ is represented as an algebraic expression. However, the size of the space required to represent a relation this way is exponential in the number of the input variables, and thus the method is not adequate in practice. Another way to represent a relation R is cubical representation. Let p be a product term of the variables of B^n and c be the input part of the cubical specification of p. Similarly, let q be a product term of the variables of B^m and d be the input part of the cubical specification of q. Consider the pair $C = [c|d]$. Then R is represented as a set of pairs $\{C = [c|d]\}$ such that for each minterm $\mathbf{x} \in D$, the characteristic function of the image of $\mathbf{x}$ by R is expressed as a set of cubes $\{d \mid \mathbf{x}$ is covered by $c\}$. In practice, R is represented with a subset of C's described above, where we assume that if $\mathbf{x} \in D$ is not covered by any of the C's then it is assumed that the image of $\mathbf{x}$ by R is a minterm of B^m in which all the output variables appear complemented.

We represent a relation R as its characteristic function $R : D \times B^m \rightarrow B$ such that $R(\mathbf{x}, \mathbf{y}) = 1$ if and only if $(\mathbf{x}, \mathbf{y}) \in R$. We call the characteristic function a **consistency**

function of the relation. The characteristic functions are represented by MDD's [18]. A MDD is a data structure which employs BDD's [5] as the internal representations to express multiple-valued functions. In Chapter 4, we present a heuristic minimizer for multiple-valued relations which takes as input a consistency function of the relation represented as an MDD. As we will see, various operations on relations can be handled efficiently using the MDD operations. Among those, one of the key operations is a *smooth* (or *existential quantification*) operation defined as follows.

Definition 10 *Given a multiple-valued function $f : D \rightarrow B$ and a set or a subset of the input variables of f, $X_k = \{x_1, \cdots, x_k\}$, the* **smooth** *of f by X_k is defined as*

$$S_{X_k}(f) = S_{x_1}(S_{x_2}(\ldots S_{x_{k-1}}(S_{x_k}(f))\cdots)),$$

where $S_{x_i}(g) = \sum_{j=0}^{P_i-1} g_{x_i^{\{j\}}}$, and $g_{x_i^{\{j\}}}$ is the cofactor of g with respect to the literal $x_i^{\{j\}}$.

A cofactor of a multiple-valued function is accomplished on MDD's by performing the BDD cofactor for the internal representations of the MDD's. The smooth operation can be implemented efficiently on MDD's.

In Section 2.5, we will see that the characteristic function of a relation is computed very easily in many applications.

2.3 Questions on Multiple-Valued Relations

A relation we consider in this report is one with binary output variables. In this section, we consider the following two questions for the relations of this form. Namely,

1. whether a relation with multiple-valued outputs can be handled with the relations we consider,

2. whether the minimization problem for a relation we consider can be transformed into the problem of minimizing an incompletely specified function with multiple-valued inputs and a single binary output.

The posing of the second question is motivated by the fact that the answer for this question is yes if the given relation is in fact an incompletely specified function.

2.3.1 Representations of Multiple-Valued Outputs

There are several contexts where the problem is formulated as the minimization of a relation with multiple-valued outputs. We discuss how such a relation is handled with a multiple-valued relation defined in this report, i.e. one with binary-valued outputs. Specifically, we describe how we can represent a multiple-valued output in terms of binary outputs using three encoding schemes known as *the* 1*-hot encoding*, *the* 0*-hot encoding*, and *the log-based encoding* respectively.

Consider a relation $\Gamma \subseteq D \times E$, where E is the Cartesian product of t sets $E_1 \times \cdots E_t$ where E_j consists of L_j integers, i.e. $E_j = \{0, \cdots, L_j - 1\}$. An encoding is the process of assigning a set of binary variables for each multiple-valued variable σ_j such that each value of E_j is associated with a subset of the Boolean space spanned by the binary variables and that the encoding is disjoint with the assignment for any other value of E_j.

The 1-hot encoding is an encoding scheme in which each multiple-valued variable σ_j is represented by L_j binary variables $(y_0^{(j)}, \cdots, y_{(L_j-1)}^{(j)})$ such that $\sigma_j = k \in E_j$ if and only if $y_k^{(j)} = 1$ and $y_i^{(j)} = 0$ for any $i \neq k$. This scheme requires $\sum_{j=1}^{t} L_j$ variables to represent all the variables of E. The 0-hot encoding is the same as the 1-hot encoding in which the meaning of 1 and 0 for each encoded binary variable is switched.

Example 2.1 *Suppose that Figure 2.1 is a specification of a relation* $\Gamma \subseteq B^2 \times E$ *with two binary inputs* x_1, x_2 *and two multiple-valued outputs* σ_1 *and* σ_2, *where both* σ_1 *and* σ_2 *can take three values. Then the transformed relation* $R \subseteq B^2 \times B^6$ *with the* 1*-hot encoding is shown in Figure 2.1.*

$\Gamma \subseteq B^2 \times E$		$R \subseteq B^2 \times B^6$	
x_1x_2	(σ_1, σ_2)	x_1x_2	$(y_0^{(1)}, y_1^{(1)}, y_2^{(1)}, y_0^{(2)} y_1^{(2)}, y_3^{(2)})$
11	{(2, 1), (1, 0)}	11	{(0, 0, 1, 0, 1, 0), (0, 1, 0, 1, 0, 0)}
10	{(1, 2)}	10	{(0, 1, 0, 0, 0, 1)}
01	{(0, 1), (0, 2)}	01	{(1, 0, 0, 0, 1, 0), (1, 0, 0, 0, 0, 1)}
00	{(1, 0)}	00	{(0, 1, 0, 1, 0, 0)}

Figure 2.1: The 1-Hot Encoding of a Relation with Multiple-Valued Outputs

The log-based encoding is a scheme that represents a multiple-valued variable with L_j values using p_j binary variables, where p_j is the smallest integer no less than $\log_2 L_j$.

Each value of E_j is represented as a product term of the encoded variables such that two product terms corresponding to two different values are disjoint and the Boolean union of the product terms over all the values of E_j is the universe of the Boolean space spanned by the encoded variables. Thus the minterm σ of E corresponds to the set given by the Boolean product of the product terms for the values of the variables of E in σ. For the transformed relation R of the original relation Γ, the image of a minterm $\mathbf{x} \in D$ is given by the union of the sets corresponding to the minterms of the image of $\mathbf{x}$ by Γ.

Example 2.2 *Consider the same relation Γ used in Example 2.1. Since both σ_1 and σ_2 can take three values, each variable is represented by two binary variables. Let $y_1^{(j)}$ and $y_2^{(j)}$ be the encoded variables for σ_j. Suppose that 0, 1, and 2 are repreented as $\overline{y}_1^{(j)}\overline{y}_2^{(j)}$, $\overline{y}_1^{(j)}y_2^{(j)}$, and $y_1^{(j)}$ respectively, then the transformed relation $R \subseteq B^2 \times B^4$ is obtained as shown in Figure 2.2.*

$\Gamma \subseteq B^2 \times E$		$R \subseteq B^2 \times B^4$	
x_1x_2	(σ_1, σ_2)	x_1x_2	$(y_1^{(1)}, y_2^{(1)}, y_1^{(2)}, y_1^{(2)})$
11	{(2, 1), (1, 0)}	11	{(1, 1, 0, 1), (1, 0, 0, 1), (0, 1, 0, 0)}
10	{(1, 2)}	10	{(0, 1, 1, 0), (0, 1, 1, 1)}
01	{(0, 1), (0, 2)}	01	{(0, 0, 0, 1), (0, 0, 1, 0), (0, 0, 1, 1)}
00	{(1, 0)}	00	{(0, 1, 0, 0)}

Figure 2.2: The Log-Based Encoding of a Relation with Multiple-Valued Outputs

2.3.2 Transformation of Multiple Outputs to a Multiple-Valued Input

It is shown in [15] that the two-level minimization problem for an incompletely specified function with multiple-valued inputs and binary-valued outputs can be equivalently handled as the problem of minimizing an incompletely specified function with a single binary output by treating the output part of the original function as 1-hot encoded variables for a single multiple-valued variable. The newly defined incompletely specified function is conceptually the characteristic function of the set of pairs of minterms of the inputs and the multiple-valued output $(\mathbf{x}, \sigma)$ such that $\mathbf{x}$ is mapped to σ by the original function, but the characteristic function explicitly uses don't cares.

We are interested in whether this equivalency holds for multiple-valued relations with binary outputs. Specifically, the question is whether there exists an encoding scheme

with which the output part of the relation can be treated as a single multiple-valued variable such that there exists an incompletely specified function with a single binary output with the inputs consisting of the original inputs and the newly defined multiple-valued variable for which the two-level minimization problem is equivalent to the problem of minimizing the original relation. As with the minimization of incompletely specified functions, we consider whether the characteristic function can be well formulated for some encoding schemes. We show that the encoding schemes described in the previous section, i.e. the 1-hot encoding, the 0-hot encoding, and the log-based encoding, do not fall into this category.

Given a relation $R \subseteq D \times B^m$ and an encoding scheme, our objective is to prove or disprove that there is a single output function that has an additional multiple-valued input. The new variable is obtained by encoding the output part of R by the given scheme. Our requirement is, as with regular functions, that the two level minimization of the resulting single function leads to a result that when interpreted correctly is a minimum of the original minimization problem.

Suppose that we employ the 1-hot encoding. Let Y be the multiple-valued variable for the outputs. Y can take m values, $E = \{0, \cdots, m-1\}$. A minterm of B^m corresponds to a set of values of E, i.e. a literal of Y. Thus the relation R maps a minterm $\mathbf{x} \in D$ to a set of literals of Y. Denote the set of literals as $r(\mathbf{x})$. If Y is treated as an input variable, then we need a function $f : D \times E \rightarrow B$ which defines the unique output value for every pair of minterms of D and E. Thus, what is associated with each minterm $\mathbf{x} \in D$ is a set of literals of Y, $r(\mathbf{x})$. There must exist exactly one literal $y \in r(\mathbf{x})$ such that the pair of minterms $(\mathbf{x}, \sigma) \in D \times E$ is mapped to 1 by the implementation obtained by minimizing f if and only if $\sigma \in y$. Therefore, for a minterm $(\mathbf{x}, \sigma)$ for which there exist literals $y \in r(\mathbf{x})$ and $\tilde{y} \in r(\mathbf{x})$ such that $\sigma \in y$ and $\sigma \notin \tilde{y}$, the value of f depends upon which literal we are concerned with. In order to understand this situation, let's divide the values of E into three sets as follows. Let $F(\mathbf{x})$ be a set of values which are contained in all the literals of $r(\mathbf{x})$, $R(\mathbf{x})$ be a set of values which are not contained in any of the literals of $r(\mathbf{x})$, and $D(\mathbf{x})$ be the rest of values. Then we see that the problem above does not arise if for an arbitrary subset $S \subseteq D(\mathbf{x})$, there exists a literal $\hat{y} \in r(\mathbf{x})$ which contains all and only the values of $F(\mathbf{x})$ and S. It is because in this case, we can set the output value of f for $(\mathbf{x}, \sigma)$ to 1 if $\sigma \in F(\mathbf{x})$, 0 if $\sigma \in R(\mathbf{x})$, and 2 or don't care otherwise. If this condition holds for every $\mathbf{x} \in D$, then the function f is well formulated and the minimized result has a correspondence between each $\mathbf{x} \in D$ and exactly one literal of $r(\mathbf{x})$. In this case, we know that by definition the

original relation $R \subseteq D \times B^m$ is reduced to an incompletely specified function. However, for a relation in general, the function f cannot be formulated. Therefore we cannot convert R to a single output incompletely specified function with the 1-hot encoding scheme. The same statement is claimed for the 0-hot encoding scheme.

Now let's consider the log-based encoding scheme. Namely, we treat each minterm of B^m as a single value of a multiple-valued variable Y. Thus Y can take 2^m values, denoted as E. In this case, R maps each minterm $\mathbf{x} \in D$ to a set of values of E. Denote this set of values by $r(\mathbf{x})$. If Y is treated as an input variable, then we need an incompletely specified function $f : D \times E \rightarrow B$ such that for every $\mathbf{x} \in D$, there must exist exactly one value $\sigma \in r(\mathbf{x})$ for which $(\mathbf{x}, \sigma)$ is mapped to 1 by the implementation obtained by minimizing f. Thus we need to decide which value of $r(\mathbf{x})$ must be chosen and once one of the values has been selected, we must not choose any other value of $r(\mathbf{x})$ with which $\mathbf{x}$ is mapped to 1. This constraint cannot be handled in the conventional minimization problem. Therefore, we cannot transform the minimization problem for a relation R to the problem of minimizing an incompletely specified function with a single output and multiple-valued inputs.

As we have seen, any of the encoding schemes considered above cannot be used for transforming multiple binary outputs to a single multiple-valued input. It is conjectured that there is no encoding scheme with which the output part of the original relation can be treated as a single multiple-valued variable such that there exists an incompletely specified function with a single binary output and the newly introduced multiple-valued variable as the additional input for which the two-level minimization problem is equivalent to the problem of minimizing the original relation.

2.4 Functions, Mappings, and Relations

Given a multiple-valued relation $R \subseteq D \times B^m$, a compatible function with R exists if and only if R is well-defined. Equivalently, R is well-defined if and only if $S_Y(R(\mathbf{x}, \mathbf{y}))$ is the tautology in D, where $R(\mathbf{x}, \mathbf{y})$ is the consistency function of R and $Y = \{y_1, \cdots, y_m\}$ is the set of variables of B^m. Thus well-definedness is easily checked on MDD's. With this condition satisfied, R can be represented as a multiple-valued mapping $r : D \rightarrow B^m$ given by $r(\mathbf{x}) = \{\mathbf{y} \in B^m \mid (\mathbf{x}, \mathbf{y}) \in R\}$, where we define a mapping as one which defines at least one minterm of B^m for each minterm of D. In general, the mapping r is a one-to-many mapping, and provides the complete family of functions compatible with R. If,

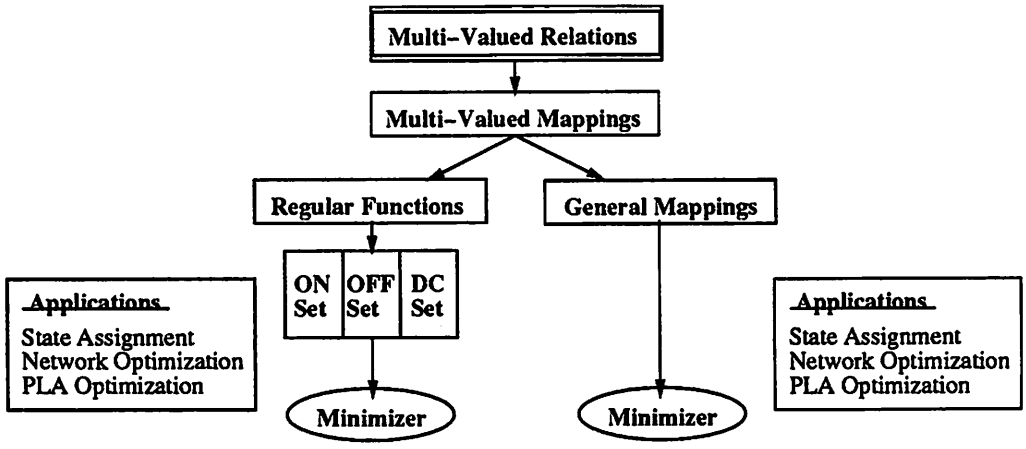

Figure 2.3: The Minimization of Relations

in addition, $r(\mathbf{x})$ can be represented as a single cube for every $\mathbf{x} \in D$, then r can be expressed as an incompletely specified function[4]. Namely, the set D can be divided into the On-set, the Off-set, and the Don't-care set for each output of r. Once these three sets are obtained, our problem is reduced to the conventional minimization problem for regular incompletely specified functions. A number of methods for solving this problem have been proposed (e.g. [3, 6, 9, 13, 14, 16]). If, on the other hand, the mapping r cannot be expressed as an incompletely specified function, we need a procedure that is capable of handling the relation R directly. The relation among functions, mappings, and relations in the minimization problem is illustrated in Figure 2.3.

In Chapter 3, we present an efficient procedure that identifies whether r can be expressed as an incompletely specified function, as well as a procedure for extracting the care sets and the don't care set from the consistency function R if it can represented as a regular function. In this way, logic minimization can be viewed in a uniform fashion, in which the minimizers for regular functions and those for general mappings are sitting in parallel as sub-procedures of the entire problem that is originally formulated with the consistency function of a relation.

2.5 Applications of Multiple-Valued Relations

One can easily imagine applications of the problem of minimizing relations, or general mappings, by taking into account the applications of the minimization of regular functions. One such example is state assignment. Suppose a completely specified finite state machine is given by the characteristic function $F : I \times S \times S \times O \rightarrow B$ such that $F(i,p,n,o) = 1$ if and only if the next state n and the output o are asserted by the input i and the current state p. F is the consistency function of a multiple-valued relation with the input set $S \times I$ and the output set $S \times O$, where the symbolic variable for S in the output set are encoded with the 1-hot encoding scheme. In fact, we know that F can be expressed in a form of a regular function since the given machine is completely specified. However, if we are given a set of equivalent states by a function $E : S \times S \rightarrow B$ such that $E(n,\tilde{n}) = 1$ if and only if the states n and $\tilde{n}$ are equivalent, as well as a set of invalid states by a function $T : S \rightarrow B$ such that $T(p) = 1$ if and only if p is a state not reachable from some initial set of states, then a set of machines equivalent to F under the equivalent states and the invalid states is given by a function $\tilde{F} : I \times S \times S \times O \rightarrow B$ such that $\tilde{F}(i,p,n,o) = 1$ if and only if $F(i,p,n,o) = 1$ or $T(p) = 1$, or there exists a state $\tilde{n}$ for which $F(i,p,\tilde{n},o) = 1$ and $E(n,\tilde{n}) = 1$. The consistency function $\tilde{F}$ is easily computed using MDD's as

$$\tilde{F}(i,p,n,o) = F(i,p,n,o) + T(p) + S_{\tilde{n}}(F(i,p,\tilde{n},o)E(n,\tilde{n})), \tag{2.1}$$

where the last term designates the smooth of the intersection of F and E with respect to the variables for S of the output sets of F and E. In order to obtain a least cost implementation of a machine equivalent to the original one, we first find a least cost machine at the symbolic level. Namely, our objective is to minimize the multiple-valued relation $\tilde{F}$.

Example 2.3 *The case where the use of the equivalent states results in a representation with less cost is illustrated in the following example. Suppose that a completely specified finite state machine is given as shown in Figure 2.4, where each circle designates a state of the machine and the label i/o associated with each arc implies that the input i and the state associated with the tail of the arc causes the machine to produce the state shown on the head of the arc and the output o. The minimized representation for this machine is shown in Figure 2.5(a). If, in addition, we know that the state s_3 and s_4 are equivalent, then we can further minimize the machine and obtain a better representation shown in Figure 2.5(b).*

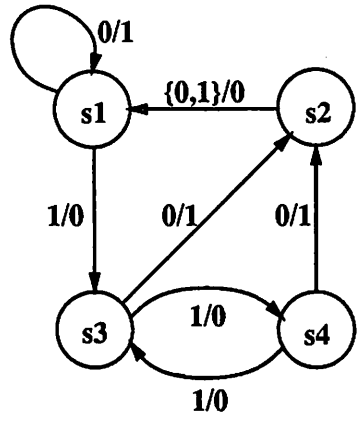

Figure 2.4: Completely Specified Finite State Machine

Other applications where relations arise are abstracted in the following way. For the sake of simplicity, we restrict ourselves for the binary-valued case. Consider the situation illustrated in Figure 2.6. Let $g : B^p \rightarrow B^q$ be a Boolean function, where B^p is a p dimensional Boolean space. Let $h : B^n \rightarrow B^q$ be a Boolean mapping, where the image of a minterm $\mathbf{x} \in B^n$ by h, $h(\mathbf{x})$, may consist of multiple minterms. Let B^r be the subspace spanned by the variables common to both B^n and B^p. B^r may be empty. For the orthocomplement of B^r in B^p, denoted as B^m, where $m = p - r$, consider the problem of finding a function $f : B^n \rightarrow B^m$ such that

$$\forall \mathbf{x} \in B^n : g([\mathbf{v}, f(\mathbf{x})]) \in h(\mathbf{x}), \tag{2.2}$$

I	S	S	O
1	s_1, s_4	s_3	0
–	s_2	s_1	0
1	s_3	s_4	0
0	s_1	s_1	1
0	s_3, s_4	s_2	1

(a)

I	S	S	O
1	s_1, s_3, s_4	s_3	0
–	s_2	s_1	0
0	s_1	s_1	1
0	s_3, s_4	s_2	1

(b)

Figure 2.5: Minimized Representations of the Finite State Machine

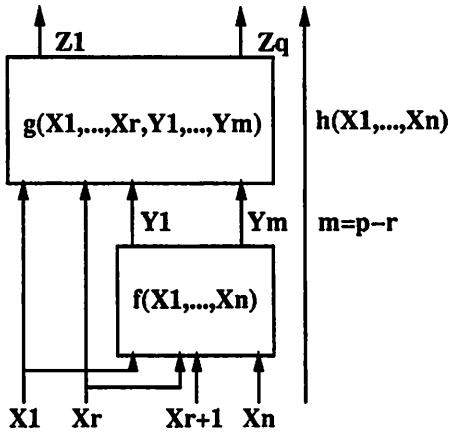

Figure 2.6: Structure where Boolean Relation Arises

where v is the projection of x from B^n to B^r. Our objective is to find an implementation f of least cost with the property (2.2), if such a function exists. Namely, considering a relation R given by $R = \{(\mathbf{x},\mathbf{y}) \in B^n \times B^m \mid g([\mathbf{v},\mathbf{y}]) \in h(\mathbf{x})\}$, our problem is to find a least cost implementation compatible with R. The relation R is computed as follows. Let $H : B^n \times B^q \rightarrow B$ be the consistency function of the relation $\{(\mathbf{x},\mathbf{z}) \mid \mathbf{z} \in h(\mathbf{x})\}$. We call H the consistency function of the Boolean mapping h. Similarly, let $G : B^p \times B^q \rightarrow B$ be the consistency function of $g : B^p \rightarrow B^q$. Then $R(\mathbf{x},\mathbf{y}) = 1$ if and only if there exists $\mathbf{z} \in B^q$ such that $H(\mathbf{x},\mathbf{z}) = 1$ and $G([\mathbf{v},\mathbf{y}],\mathbf{z}) = 1$. Therefore, we obtain R as

$$R(\mathbf{x},\mathbf{y}) = S_Z(G([\mathbf{v},\mathbf{y}],\mathbf{z})H(\mathbf{x},\mathbf{z})), \tag{2.3}$$

where $Z = \{z_1, \cdots, z_q\}$ is a set of the variables of B^q.[2]

This problem naturally arises in several applications. One context is concerned with "engineering changes", where one has built a piece of logic that implements a function g with a highly optimized layout, only to encounter a specification change such that the correct functionality must be h. One possibility for rectifying this situation is to build a block of prelogic which sits between the inputs and the circuit already built. Equation (2.2)

[2]If h is a Boolean function, then R can be computed directly as $R(\mathbf{x},\mathbf{y}) = \prod_{j=1}^{q}(g^{(j)}([\mathbf{v},\mathbf{y}]) \equiv h^{(j)}(\mathbf{x}))$, where $g^{(j)}$ and $h^{(j)}$ are the j-th function of g and h respectively.

gives the condition for the function f of the attached block so that the resulting circuit is functionally equivalent to h. A detailed discussion of the rectification problem can be found in [19]. Another context is the optimization of hierarchical networks [4], where a network with the functionality f feeds another network with the functionality g and the functionality of the hierarchy is given by h. One wants to minimize the area of the network for f while still keeping the functionality of the hierarchy. The existence of f is guaranteed in this case. A similar situation arises in the multiple-level logic synthesis where the post-logic g has been optimized first and it remains to find an optimum representation for the pre-logic f.

Chapter 3

Function Minimization and Relation Minimization

As we have seen in Section 2.4, a multiple-valued relation may happen to be an incompletely specified function. If this is the case, we should not invoke a procedure developed for minimizing relations since the minimization of incompletely specified functions is a simpler problem for which highly optimized software exists, and thus is completed more efficiently with a procedure designed exclusively for regular functions. Therefore, we need a method that identifies whether a given relation is an incompletely specified function. We present one such method with which one can obtain, as a byproduct, the on-set, the off-set, and the don't-care set if a given relation is an incompletely specified function.

The overall flow of the procedure is illustrated in Figure 3.1. The input of the procedure is a consistency function of a well-defined multiple-valued relation $R \subseteq D \times B^m$. Hence there exists an associated mapping $r(\mathbf{x})$. For an output variable y_j, let $F^{(j)}$ be a set of minterms $\mathbf{x} \in D$ such that y_j appears not complemented in all minterms $\mathbf{y} \in r(\mathbf{x})$. Similarly, let $R^{(j)}$ be a set of minterms of $\mathbf{x} \in D$ such that y_j appears complemented in all minterms $\mathbf{y} \in r(\mathbf{x})$. Let $D^{(j)}$ be the the remaining minterms of D. These sets are computed as follows. Let $s_1^{(j)}$ be a set of minterms $\mathbf{x} \in D$ such that there exists a minterm $\mathbf{y} \in r(\mathbf{x})$ in which y_j appears complemented. Let $s_2^{(j)}$ be a set of minterms $\mathbf{x} \in D$ such that there exists a minterm $\mathbf{y} \in r(\mathbf{x})$ in which y_j appears not complemented. Using MDD's, $s_1^{(j)}$ and $s_2^{(j)}$ are given by $s_1^{(j)} = S_Y(R(\mathbf{x},\mathbf{y})\overline{y}_j)$ and $s_2^{(j)} = S_Y(R(\mathbf{x},\mathbf{y})y_j)$, where Y is the set of

```
procedure IDENTIFY(R(x,y))
begin
    T(x,y) ← 1;
    for (each output y_j){
        F^(j) ← {x ∈ D | ∀y ∈ r(x) : y_j = 1};
        R^(j) ← {x ∈ D | ∀y ∈ r(x) : y_j = 0};
        D^(j) ← {x ∈ D | x ∉ (F^(j) ∪ R^(j))};
        T(x,y) ← T(x,y)(F^(j) y_j + R^(j) ȳ_j + D^(j));
    }
    if (T(x,y) ≡ R(x,y))    return 1;
    else                    return 0;
end
```

Figure 3.1: The Procedure to Identify if the Relation is Reduced to a Regular Function

output variables[1]. Then we obtain $F^{(j)} = \neg s_1^{(j)}$, $R^{(j)} = \neg s_2^{(j)}$, and $D^{(j)} = s_1^{(j)} s_2^{(j)}$. Note that the formulas for $F^{(j)}$ and $R^{(j)}$ are valid if and only if R is well-defined. The proposed procedure computes the three sets $F^{(j)}$, $R^{(j)}$, and $D^{(j)}$ for every output y_j in parallel, and then computes $T(\mathbf{x},\mathbf{y}) = \prod_{j=1}^{m}(F^{(j)} y_j + R^{(j)} \overline{y}_j + D^{(j)})$. Due to the following theorem, we can identify if R is in fact an incompletely specified function by comparing R and T.

Theorem 3.1 *A well-defined relation $R(\mathbf{x},\mathbf{y})$ is the characteristic function of an incompletely specified function if and only if $R(\mathbf{x},\mathbf{y})$ is equivalent to $T(\mathbf{x},\mathbf{y})$.*

Proof: Suppose $R(\mathbf{x},\mathbf{y})$ is the characteristic function of an incompletely specified function $\zeta : D \rightarrow B^m$. Then ζ is represented in terms of three functions, $\varphi : D \rightarrow B^m$, $\rho : D \rightarrow B^m$,

[1] In this paper, we make no distinction between a set and the characteristic function of the set.

and $\delta : D \rightarrow B^m$ such that for each j,

$$
\begin{aligned}
&(a) \quad \forall \mathbf{x} \in D : (\varphi^{(j)}(\mathbf{x}) \oplus \rho^{(j)}(\mathbf{x}))\overline{\delta}^{(j)}(\mathbf{x}) + (\rho^{(j)}(\mathbf{x}) \oplus \delta^{(j)}(\mathbf{x}))\overline{\varphi}^{(j)}(\mathbf{x}) = 1, \\
&(b) \quad R(\mathbf{x},\mathbf{y}) = 1 \Leftrightarrow y_j = \begin{cases} 1 & \text{if } \varphi^{(j)}(\mathbf{x}) + \delta^{(j)}(\mathbf{x}) = 1, \\ 0 & \text{if } \rho^{(j)}(\mathbf{x}) + \delta^{(j)}(\mathbf{x}) = 1, \end{cases}
\end{aligned} \tag{3.1}
$$

where $\varphi^{(j)}$, $\rho^{(j)}$, and $\delta^{(j)}$ are the j-th function of φ, ρ, and δ respectively, and $(f \oplus g)$ designates an XOR operation between f and g. Thus $R(\mathbf{x},\mathbf{y})$ is represented as $R(\mathbf{x},\mathbf{y}) = \prod_{j=1}^{m}(\varphi^{(j)}y_j + \rho^{(j)}\overline{y}_j + \delta^{(j)})$. Now for an arbitrary minterm $\mathbf{x} \in D$, if $y_j = 1$ in every minterm $\mathbf{y} \in r(\mathbf{x})$, then $\varphi^{(j)}(\mathbf{x}) = 1$ and $\rho^{(j)}(\mathbf{x}) = \delta^{(j)}(\mathbf{x}) = 0$. Similarly, if $y_j = 0$ in every minterm $\mathbf{y} \in r(\mathbf{x})$, then $\rho^{(j)}(\mathbf{x}) = 1$ and $\varphi^{(j)}(\mathbf{x}) = \delta^{(j)}(\mathbf{x}) = 0$. If $y_j = 1$ in some minterm of $r(\mathbf{x})$ and $y_j = 0$ in another minterm of $r(\mathbf{x})$, then $\delta^{(j)}(\mathbf{x}) = 1$ and $\varphi^{(j)}(\mathbf{x}) = \rho^{(j)}(\mathbf{x}) = 0$, since otherwise the property (3.1)-(a) is violated. Therefore, $\varphi^{(j)}$, $\rho^{(j)}$, and $\delta^{(j)}$ are the characteristic functions of $F^{(j)}$, $R^{(j)}$, and $D^{(j)}$ respectively. Thus $T(\mathbf{x},\mathbf{y})$ is equivalent to $R(\mathbf{x},\mathbf{y})$.

Conversely, suppose that $R(\mathbf{x},\mathbf{y})$ is equivalent to $T(\mathbf{x},\mathbf{y})$. For an arbitrary minterm $\pi \in D$, let $J_0(\pi)$ be the set of indices j such that $\pi \in R^{(j)}$. Similarly, let $J_1(\pi)$ and $J_2(\pi)$ be the set of indices such that $\pi \in F^{(j)}$ and $\pi \in D^{(j)}$ respectively. Let σ be a minterm of B^m such that $y_j = 1$ if $j \in J_1(\pi)$, $y_j = 0$ if $j \in J_0(\pi)$, and y_j may be either 1 or 0 if $j \in J_2(\pi)$, where y_j is the value of the j-th variable of σ. Since $R(\mathbf{x},\mathbf{y}) = \prod_{j=1}^{m}(F^{(j)}y_j + R^{(j)}\overline{y}_j + D^{(j)})$, we see that $R(\pi,\sigma) = 1$. Thus the image of π by the relation R, $r(\pi)$, can be represented as a single product term of the output variables such that the j-th literal contains only 1 (respectively 0) if $j \in J_1(\pi)$ (respectively $j \in J_0(\pi)$) and it contains both 1 and 0 otherwise. Since π is arbitrary, the relation R is the characteristic function of an incompletely specified function. ■

Therefore, we can identify whether a given relation R is an incompletely specified function by checking the equivalency between $R(\mathbf{x},\mathbf{y})$ and $T(\mathbf{x},\mathbf{y})$.

Furthermore, by the proof of Theorem 3.1, we see that if R is an incompletely specified function, then its on-set, off-set and the don't-care set are given by the $F^{(j)}$'s, $R^{(j)}$'s, and $D^{(j)}$'s, respectively. Hence, having checked the well-definedness of a given relation R as described in Section 2.4, we first employ the procedure described above. If the relation is an incompletely specified function, we convert the $F^{(j)}$'s, $R^{(j)}$'s, and $D^{(j)}$'s to some required form, e.g. sum-of-products form, and invoke a conventional minimizer for regular functions,

e.g. Espresso[14]. If it turns out that the relation is not reduced to a regular function, then we minimize the relation directly using a relation minimizer, GYOCRO, presented in the following section.

Chapter 4

GYOCRO: Minimization of Multiple-Valued Relations

4.1 Problem Formulation and Overview

We consider the following problem: *given a well-defined multiple-valued relation* $R \subseteq D \times B^m$, *find a representation* $\mathcal{F}$ *with the minimum number of product terms that is compatible with* R. We propose a heuristic procedure for the problem which takes as input a consistency function of R represented by an MDD and computes a compatible representation with a minimal number of product terms.

The procedure starts with computing an initial representation compatible with the relation. Then three basic procedures, REDUCE, EXPAND, and IRREDUNDANT, are iteratively applied as long as the cost decreases, where the cost is the number of the product terms of the representation. Every procedure takes as input a compatible representation $\mathcal{F}$ and the consistency function R for the relation.

In the REDUCE procedure, each cube $c \in \mathcal{F}$ is reduced to a smallest cube $\hat{c} \subseteq c$ such that $\mathcal{F} - \{c\} \cup \{\hat{c}\}$ is compatible with R, where $\mathcal{F} - \mathcal{S} \cup \mathcal{T}$ designates the replacement of $\mathcal{S} \subseteq \mathcal{F}$ by a set of cubes $\mathcal{T}$. It is guaranteed at the end of the REDUCE procedure that for every cube c in the resulting representation, any cube $\hat{c}$ with the condition above is equal to c. The EXPAND procedure, in turn, takes each cube $c \in \mathcal{F}$ and replaces it with a relatively prime cube containing c so that a maximal number of cubes in $\mathcal{F}$ can be removed. The EXPAND procedure guarantees that every cube of the resulting representation is relatively

prime. The IRREDUNDANT procedure makes the representation $\mathcal{F}$ irredundant. Specifically, a cube $c \in \mathcal{F}$ is removed if and only if $\mathcal{F} - \{c\}$ is compatible. Unlike the irredundant procedure of Espresso [14], which computes a minimal cover of the current representation for a regular function based on the concept of partially redundant cubes, our procedure is dependent on the order that the cubes are processed. In fact, the procedure is a special case of the REDUCE procedure since a redundant cube is reduced to nothing. However, our experimental results show that use of IRREDUNDANT improves the computational time. The overall procedure finally returns a relatively prime and irredundant representation.

4.2 Initial Representation

This section describes how to compute a compatible representation for a given consistency function $R(\mathbf{x}, \mathbf{y})$ of a well-defined relation. Our procedure takes a representation $\mathcal{F}$, which is initially empty, and processes each output adding a set of new cubes to $\mathcal{F}$. $\mathcal{F}$ becomes compatible with R once all the outputs have been processed.

For an output variable y_j, let $F^{(j)}$ be a set of minterms $\mathbf{x} \in D$ such that y_j appears not complemented in all minterms $\mathbf{y}$ of the image of $\mathbf{x}$ by R. Similarly, let $R^{(j)}$ be a set of minterms of $\mathbf{x} \in D$ such that y_j appears complemented in all minterms $\mathbf{y} \in r(\mathbf{x})$. Let $D^{(j)}$ be the the remaining minterms of D. These are the same sets introduced in identifying if a given relation is an incompletely specified function in Chapter 3, and thus are computed in the same way described in the previous chapter. Once these three sets are obtained, a two level minimizer for multiple-valued functions is invoked with $F^{(j)}$, $R^{(j)}$, and $D^{(j)}$ as the on-set, the off-set, and the don't-care set respectively, so that a minimal sum-of-products representation $\mathcal{F}_j$ with n inputs and a single output is computed. Then $\mathcal{F}_j$ is converted to a representation with n inputs and m outputs such that the input part is identical with $\mathcal{F}_j$ and the output part has 1 only in the j-th component. The converted representation is added to $\mathcal{F}$. Once $\mathcal{F}_j$ is computed, the relation R is modified so that for every pair of minterms $(\mathbf{x}, \mathbf{y}) \in R$, y_j appears complemented in $\mathbf{y}$ if and only if $\mathbf{x}$ is not covered by $\mathcal{F}_j$. Equivalently, $R(\mathbf{x}, \mathbf{y})$ is replaced by $R(\mathbf{x}, \mathbf{y})(F_j \equiv y_j)$, where $(f \equiv g)$ designates an XNOR operation between f and g. F_j is a set of minterms of D covered by $\mathcal{F}_j$, which is computed by converting $\mathcal{F}_j$ to an MDD. Once all the output variables have been processed, a compatible representation is obtained in a unwrapped form, i.e. each cube in the representation has only one 1 in its output part.

4.3 Computing a Consistency Function

REDUCE, EXPAND, and IRREDUNDANT procedures process one cube $c \in \mathcal{F}$ at a time. In each operation, we need to compute a consistency function of a function corresponding to $\mathcal{F} - \{c\}$, denoted as $F_c(\mathbf{x},\mathbf{y})$. A consistency function of a function $f : D \rightarrow B^m$ is a consistency function of a relation $F = \{(\mathbf{x},\mathbf{y}) \in D \times B^m \mid \mathbf{y} = f(\mathbf{x})\}$. Throughout the rest of the paper, we call F_c a consistency function of $\mathcal{F} - \{c\}$. In general, a consistency function of a representation $\mathcal{F}$, denoted as $F(\mathbf{x},\mathbf{y})$, can be computed by scanning all the cubes of $\mathcal{F}$ to obtain the function $f^{(j)} : D \rightarrow B$ for each output y_j, followed by setting $F(\mathbf{x},\mathbf{y}) = \prod_{j=1}^{m}(f^{(j)}(\mathbf{x}) \equiv y_j)$. This will be referred to later as the "first" method. However, this method is time consuming, especially when the size of $\mathcal{F}$ is large. In fact, $F_c(\mathbf{x},\mathbf{y})$ can be computed much easier by the following method if F is already available.

Let $\mathcal{S}$ be a subset of $\mathcal{F}$ defined as $\mathcal{S} = \{p \in \mathcal{F} \mid p \neq c \text{ and } M(p) \cap M(c) \neq \phi\}$, where we recall that $M(p)$ is a set of minterms of D covered by the product term corresponding to the input part of p. Let $F_{\mathcal{S}} : D \times B^m \rightarrow B$ be a function such that $F_{\mathcal{S}}(\mathbf{x},\mathbf{y}) = 1$ if and only if

$$\forall j \in \{1,\cdots,m\} \quad : \quad y_j = \begin{cases} 1 & \text{if } O(c)_j = 1 \text{ and } \exists p \in \mathcal{S} \text{ such that } \mathbf{x} \in M(p) \text{ and } O(p)_j = 1, \\ 0 & \text{otherwise,} \end{cases}$$

where y_j is the value of the j-th variable in $\mathbf{y}$. Now we observe that for a minterm $\mathbf{x} \notin M(c)$, the image of $\mathbf{x}$ by F_c is identical with the image of $\mathbf{x}$ by F. For a minterm $\mathbf{x} \in M(c)$, on the other hand, the value of y_j of the image of $\mathbf{x}$ by F_c is identical with y_j of the image by F if $O(c)_j = 0$. If $O(c)_j = 1$ and there exists no $p \in \mathcal{F} - \{c\}$ such that $\mathbf{x} \in M(p)$ and $O(p)_j = 1$, then $y_j = 0$. Otherwise $y_j = 1$. Thus $F_c(\mathbf{x},\mathbf{y}) = 1$ if and only if

$$\forall j \in \{1,\cdots,m\} \quad : \quad y_j = \begin{cases} \tilde{y}_j & \text{if } \mathbf{x} \in M(c) \text{ and } O(c)_j = 1, \\ \hat{y}_j & \text{otherwise,} \end{cases}$$

where $\tilde{y}_j$ is the value of the j-th variable in the minterm $\tilde{\mathbf{y}}$ such that $F_{\mathcal{S}}(\mathbf{x},\tilde{\mathbf{y}}) = 1$ and $\hat{y}_j$ is the value of the j-th variable in the minterm $\hat{\mathbf{y}}$ such that $F(\mathbf{x},\hat{\mathbf{y}}) = 1$. Therefore, using additional variables $T = \{t_1,\cdots,t_m\}$ and $Z = \{z_1,\cdots,z_m\}$ to represent F and $F_{\mathcal{S}}$ in terms of $(\mathbf{x},\mathbf{t})$ and $(\mathbf{x},\mathbf{z})$ respectively, $F_c(\mathbf{x},\mathbf{y})$ is computed as

$$F_c(\mathbf{x},\mathbf{y}) = (\neg M(c))F(\mathbf{x},\mathbf{y}) + S_{(T \cup Z)}(M(c)F(\mathbf{x},\mathbf{t})F_{\mathcal{S}}(\mathbf{x},\mathbf{z}) \prod_{j \in Y_l(c)} (y_j \equiv t_j) \prod_{j \notin Y_l(c)} (y_j \equiv z_j)), \tag{4.1}$$

where $Y_l(c) = \{j \in \{1, \cdots, m\} \mid O(c)_j = 0\}$. In this way, F_c is computed from F by scanning a subset of $\mathcal{F}$.

According to our experiments, although the first method is slightly faster for small examples such as 10 variables and 20 cubes, the CPU time for the second method is almost invariant with the size of the representations and is much faster than the first method for moderate sized examples, including one with 33 binary variables and 553 cubes, for which the second method was 20 times faster.

The second method assumes that a consistency function F for $\mathcal{F}$ is available. The computation of F is done once at the beginning of the entire procedure using the first method. However, it must be updated whenever a cube of $\mathcal{F}$ is replaced by another cube either in the REDUCE or the EXPAND procedure. As a converse of the second method, if we have $\mathcal{F}$, $c \in \mathcal{F}$, and a consistency function of $\mathcal{F} - \{c\}$, F_c, then we can compute $F(\mathbf{x}, \mathbf{y})$ as follows. For a minterm $\mathbf{x} \in D$, the value of y_j of the image of $\mathbf{x}$ by F is identical with y_j of the image by F_c if $\mathbf{x} \notin M(c)$ or $\mathbf{x} \in M(c)$ but $O(c)_j = 0$. Otherwise $y_j = 1$. Thus using new variables $T = \{t_1, \cdots, t_m\}$ to represent F_c in terms of $(\mathbf{x}, \mathbf{t})$, we obtain

$$F(\mathbf{x}, \mathbf{y}) = (\neg M(c))F_c(\mathbf{x}, \mathbf{y}) + S_T(M(c)F_c(\mathbf{x}, \mathbf{t}) \prod_{j \notin Y_l(c)} y_j \prod_{j \in Y_l(c)} (y_j \equiv t_j)). \qquad (4.2)$$

In this way, each of the REDUCE, EXPAND, and IRREDUNDANT procedures takes as an additional input a consistency function F for $\mathcal{F}$, updates it whenever $\mathcal{F}$ changes, and hands it to the succeeding procedure. Note that it is not necessary to update F in the IRREDUNDANT procedure since if c is removed, then F_c becomes the consistency function of the new representation.

4.4 Reduce Procedure

A reduction is an operation which takes a cube c and returns a cube $\hat{c} \subseteq c$, where $\hat{c}$ may be empty. A reduction is *valid* if the replacement of c with the reduced cube results in a compatible representation with R.

Definition 11 *For a given representation $\mathcal{F}$ compatible with R and a cube $c \in \mathcal{F}$, a cube $c^- \subseteq c$ is a maximally reduced cube for c in $\mathcal{F}$ if $\mathcal{F} - \{c\} \cup \{c^-\}$ is compatible with R and there exists no cube $\hat{c}^- \subset c^-$ such that $\mathcal{F} - \{c\} \cup \{\hat{c}^-\}$ is compatible.*

If a maximally reduced cube for c in $\mathcal{F}$ is c itself, we say c is maximally reduced. The goal of the REDUCE procedure is to compute a compatible representation which consists of maximally reduced cubes. It is known that if R is in fact an incompletely specified function, i.e. $r(\mathbf{x})$ can be expressed by a single cube for every $\mathbf{x} \in D$, then the maximally reduced cube for c in $\mathcal{F}$ is unique. It is also known in this case that for any cube $\tilde{c}$ which is contained in c and contains the maximally reduced cube, the replacement of c by $\tilde{c}$ preserves the compatibility. Thus the maximally reduced cube is easily obtained by randomly lowering one part at a time from 1 to 0 for c, followed by checking the compatibility of the resulting representation, until no valid reduction is possible[1]. Therefore, a two level minimizer for regular functions achieves the goal mentioned above by taking each cube and replacing it with the uniquely defined maximally reduced cube.

The REDUCE procedure of the proposed method also processes one cube at a time and replaces it by a maximally reduced cube. Thus we are interested in if the uniqueness holds for relations.

4.4.1 Reduction of the Input Part

The following theorems calim that the reduction of the input part of a cube $c \in \mathcal{F}$ is completed in a similar way to the reduction for regular functions.

Theorem 4.1 *Let $\mathcal{F}$ be a representation compatible with a relation R. The input part of a maximally reduced cube for $c \in \mathcal{F}$ is unique.*

Proof: Suppose that there are two maximally reduced cubes $c^{(1)}$ and $c^{(2)}$. The objective is to show that $M(c^{(1)}) \equiv M(c^{(2)})$. We first show that $M(c^{(1)}) \cap M(c^{(2)}) \neq \phi$. Suppose the contrary. Let $\hat{c}^{(1)}$ be any cube strictly contained in $c^{(1)}$ obtained by reducing only the input part of $c^{(1)}$. $\hat{c}^{(1)}$ may be an empty cube. Let ω be any minterm included in $M(c^{(1)}) \cap \neg M(\hat{c}^{(1)})$. Since $c^{(1)}$ and $c^{(2)}$ are disjoint, $\omega \notin M(c^{(2)})$. However, since $\mathcal{F} - \{c\} \cup \{c^{(2)}\}$ is compatible, the image of ω by $\mathcal{F} - \{c\}$ must be a member of $r(\omega)$. Thus a reduction from $c^{(1)}$ to $\hat{c}^{(1)}$ is valid, which is conflict. Therefore $M(c^{(1)})$ and $M(c^{(2)})$ are not disjoint.

Suppose that $M(c^{(1)}) \not\equiv M(c^{(2)})$. Let $\mathbf{m}$ be any minterm included in $M(c^{(1)}) \cap \neg M(c^{(2)})$. Let m_j be the value of the j-th input variable in $\mathbf{m}$. Let $\hat{c}^{(1)}$ be the cube which is identical with $c^{(1)}$ except that $I(\hat{c}^{(1)})_j = \neg m_j \cap I(c^{(1)})_j$, where j is an index such that

[1]Note that $\mathcal{F}$ is compatible with R if and only if $F(\mathbf{x}, \mathbf{y}) \subseteq R(\mathbf{x}, \mathbf{y})$.

$m_j \notin I(c^{(2)})_j$. Note that $\hat{c}^{(1)}$ is not an empty cube since otherwise $I(c^{(1)})_j \cap I(c^{(2)})_j = \phi$, which implies $M(c^{(1)}) \cap M(c^{(2)}) = \phi$. Since $c^{(1)}$ is maximally reduced, $\mathcal{F} - \{c\} \cup \{\hat{c}^{(1)}\}$ is not compatible. This implies that there exists a minterm $\pi \in M(c^{(1)}) \cap \neg M(\hat{c}^{(1)})$ such that the image of π by $\mathcal{F} - \{c\}$ is not a member of $r(\pi)$. However, we know that $\pi_j = m_j$, and thus $\pi \notin M(c^{(2)})$. Since $\mathcal{F} - \{c\} \cup \{c^{(2)}\}$ is compatible and since $\pi \notin M(c^{(2)})$, the image of π by $\mathcal{F} - \{c\}$ must be a member of $r(\pi)$, which is a contradiction. Thus the assumption that $M(c^{(1)}) \not\equiv M(c^{(2)})$ is incorrect. ■

Theorem 4.2 *Given a relation R and a cube $c \in \mathcal{F}$, where $\mathcal{F}$ is a representation compatible with R, let $\tilde{c} \subseteq c$ be a cube such that $O(\tilde{c}) = O(c)$ and $M(\tilde{c}) \supseteq M(c^-)$, where c^- is a maximally reduced cube for c in $\mathcal{F}$. Then the reduction from c to $\tilde{c}$ is valid.*

Proof: For any minterm $\mathbf{m} \in M(\tilde{c})$, we have $\mathbf{m} \in M(c)$. Since $O(\tilde{c}) = O(c)$, the image of $\mathbf{m}$ by $\mathcal{F} - \{c\} \cup \{\tilde{c}\}$ is equal to the image by $\mathcal{F}$. For any minterm $\mathbf{m} \in M(c) \cap \neg M(\tilde{c})$, $\mathbf{m} \notin M(c^-)$. Thus the image of $\mathbf{m}$ by $\mathcal{F} - \{c\} \cup \{\tilde{c}\}$ is equal to the image by $\mathcal{F} - \{c\} \cup \{c^-\}$, which is a member of $r(\mathbf{m})$. Therefore, $\mathcal{F} - \{c\} \cup \{\tilde{c}\}$ is a compatible representation. ■

Thus by fixing the output part of c, we can lower each part of $I(c)$ with the value 1 to 0, one at a time, and accept the reduction if the new representation is compatible with R. The unique input part of a maximally reduced cube is obtained once all the input components have been processed. Note that c is redundant in $\mathcal{F}$ if there is a component $I(c)_j$ in which every part is 0.

We can check the compatibility of the new representation as follows, without computing its consistency function. Let $\tilde{c}$ be a cube which is identical with c except that a single part of $I(c)$ has been lowered from 1 to 0. Let $Q = M(c) \cap \neg M(\tilde{c})$. We see that $\mathcal{F} - \{c\} \cup \{\tilde{c}\}$ is compatible if and only if for any minterm $\mathbf{m} \in Q$, the image of $\mathbf{m}$ by $\mathcal{F} - \{c\}$ is a member of $r(\mathbf{m})$. In terms of MDD's, $\mathcal{F} - \{c\} \cup \{\tilde{c}\}$ is compatible with R if and only if $Q \equiv S_Y(R(\mathbf{x},\mathbf{y})F_c(\mathbf{x},\mathbf{y})Q)$.

4.4.2 Reduction of the Output Part

We have seen that the reduction of the input part is completed in a straightforward way. Unlike the input part, however, the following example show that the output part of the maximally reduced cube is not unique.

Example 4.1 *Suppose that a representation $\mathcal{F}$ shown in Figure 4.1 is compatible with R, where part of its relations are shown in Figure 4.1. Then $c_1^{(1)} = [1021 \mid 1000]$ and $c_1^{(2)} = [1021 \mid 0110]$ are both maximally reduced cubes for c_1 in $\mathcal{F}$.*

Representation $\mathcal{F}$			Relation R	
cube	Input	Output	$x \in B^4$	$y \in B^4$
c_1	1021	1110	1011	{1111, 0111, 1001}
c_2	1201	0100	1001	{1111, 0111, 1101, 1001, 0001}
c_3	2021	0001	1101	{0100}
c_4	2211	0001	1111	{0001}

Figure 4.1: Example where a Maximally Reduced Cube is not Unique

Since we want to reduce c as much as possible, we want to find the smallest maximally reduced cube, i.e. the maximally reduced cube with the minimum number of 1's in the output part. One difficulty to find such a cube is that the compatibility does not necessarily hold for a cube which is contained in c and contains a maximally reduced cube. For Example 4.1, although $\mathcal{F}$ and $\mathcal{F} - \{c_1\} \cup \{c_1^{(1)}\}$ are both compatible with R, for $p = [1021 \mid 1100]$ or $q = [1021 \mid 1010]$, the replacement of c by either p or q results in an incompatible representation. Thus the smallest maximally reduced cube may not be found by greedily reducing the output part of c.

Now consider a cube c such that its input part is maximally reduced. We define a *feasible cube* for c as follows: c^* is feasible if $\mathcal{F} - \{c\} \cup \{c^*\}$ is compatible with R, the input part of c^* is equal to that of c, and $c^* \subseteq c$. The feasible cube with the minimum number of 1's in the output part is the maximally reduced cube we seek. For a set of components $Y_h = \{j \in \{1, \cdots, m\} \mid O(c)_j = 1\}$, consider a Boolean space $B^{|Y_h|}$ defined by the output variables of Y_h. For a minterm $\mathbf{y} \in B^{|Y_h|}$, we define a reduced cube c^* for c with respect to $\mathbf{y}$ as follows. The input part is equal to that of c, and $O(c^*)_j = 1$ if and only if $j \in Y_h$ and $y_j = 1$. Let $h : B^{|Y_h|} \rightarrow B$ be a function such that $h(\mathbf{y}) = 1$ if and only if c^* is feasible, where c^* is a reduced cube for c with respect to $\mathbf{y}$. By definition, $h(\mathbf{y}) = 1$ if and only if the following property holds for every minterm $\mathbf{x} \in M(c)$:

$$\exists \tilde{\mathbf{y}} \in B^m : \quad R(\mathbf{x}, \tilde{\mathbf{y}}) = 1, \quad \forall j \in \{1, \cdots m\} : \quad \tilde{y}_j = \begin{cases} \hat{y}_j + y_j & \text{if } j \in Y_h, \\ \hat{y}_j & \text{if } j \notin Y_h, \end{cases} \tag{4.3}$$

where $\hat{y}_j$ is the value of the j-th variable of the minterm $\hat{\mathbf{y}} \in B^m$ such that $F_c(\mathbf{x}, \hat{\mathbf{y}}) = 1$.

Let $H : M(c) \times B^{|Y_h|} \rightarrow B$ be a function such that $H(\mathbf{x},\mathbf{y}) = 1$ if and only if (4.3) holds for $(\mathbf{x},\mathbf{y})$. Using additional variables $T = \{t_1, \cdots, t_m\}$ and $Z = \{z_1, \cdots, z_m\}$ to represent R and F_c in terms of $(\mathbf{x},\mathbf{z})$ and $(\mathbf{x},\mathbf{t})$, $H(\mathbf{x},\mathbf{y})$ is computed as

$$H(\mathbf{x},\mathbf{y}) = S_{(T \cup Z)}(M(c)R(\mathbf{x},\mathbf{z})F_c(\mathbf{x},\mathbf{t}) \prod_{j \in Y_h} (z_j \equiv (t_j + y_j)) \prod_{j \notin Y_h} (z_j \equiv t_j)).$$

Observing that $h(\mathbf{y}) = 0$ if and only if there exists $\mathbf{x} \in M(c)$ such that $H(\mathbf{x},\mathbf{y}) = 0$, $h(\mathbf{y})$ is obtained as $h(\mathbf{y}) = \neg(S_X(M(c)(\neg H(\mathbf{x},\mathbf{y}))))$.

Note that $h(\mathbf{y})$ is represented as a BDD, since the output variables are all binary. Once $h(\mathbf{y})$ is computed, the maximally reduced cube with the minimum number of 1's is obtained efficiently. In fact, the following theorem is analogous to a result of a BDD based approach for the binate covering problem [12].

Theorem 4.3 *The maximally reduced cube for c in $\mathcal{F}$ with the minimum number of 1's is given by the shortest path connecting a 1 leaf to the root of a BDD for the function $h(\mathbf{y})$ defined above, where the length of an edge of the BDD is 1 if the edge is a 1 edge and 0 if the edge is a 0 edge.*

Proof: For a reduced cube c^* with respect to $\mathbf{y}$, $O(c^*)_j = 1$ if and only if $y_j = 1$. Thus the maximally reduced cube with the minimum number of 1's is a reduced cube for $\mathbf{y}$ with the minimum number of 1's such that $h(\mathbf{y}) = 1$. For any path from the root of the BDD for h which ends with a 1 leaf, one can obtain a minterm $\mathbf{y}$ such that $h(\mathbf{y}) = 1$, by setting $y_j = 1$ only if the path contains a 1 edge incident with a node for y_j. The number of 1's of the minterm is minimal among all the minterms represented by the path. Let $\mathbf{y}$ be the minterm with the minimal number of 1's associated with the shortest path P that ends with a 1 leaf. The proof is done if we show that $\mathbf{y}$ has the minimum number of 1's. Suppose that there is another minterm $\mathbf{y}'$ such that the number of the 1's of $\mathbf{y}'$ is less than that of $\mathbf{y}$. Then there exists at least one path in the BDD corresponding to $\mathbf{y}'$. Let P' be the shortest such path. Since the number of 1's of $\mathbf{y}'$ is at least the length of P', P' must be shorter than P, which is a contradiction. ■

Therefore, once the input part of a cube $c \in \mathcal{F}$ has been maximally reduced, the smallest maximally reduced cube for c is obtained by computing a function $h(\mathbf{y})$, followed by performing a shortest path algorithm, which runs in linear time with respect to the number of the nodes of BDD's.

4.4.3 Reduce Procedure

We have shown how to compute a maximally reduced cube for each cube of $\mathcal{F}$. However, we want to compute a representation in which every cube is maximally reduced. Due to a nature of sum-of-products representations for relations, a property of a cube like maximal reduction, which may hold as some time in the reduction process, may cease to hold when other cubes are subsequently reduced. We must deal with this difficulty which does not arise for regular functions. This is illustrated by the following example.

Example 4.2 *In Example 4.1, suppose* c_1 *has been replaced by a maximally reduced cube* $c_1^{(1)}$. *If we have replaced* c_2 *by its maximally reduced cube* $c_2^{(1)} = [1101 \mid 0100]$, *then* $c_1^{(1)}$ *can be further reduced to* $[1011 \mid 1000]$ *while still keeping the compatibility of the resulting representation.*

One solution for this problem is to iterate the entire procedure, until no cube is replaced by a smaller cube. Then at the end, it is guaranteed that every cube of the final representation is maximally reduced.

As with regular functions, the result of this procedure depends upon the order of the cubes to be processed. We order the cubes with the same strategy employed in Espresso [2], in which the largest cube is processed first and the rest of the cubes are sorted in increasing order of the number of mismatches (distance) of each cube against the largest one.

4.5 Expand Procedure

The objective of the EXPAND procedure is to remove as many cubes as possible from a given compatible representation. Furthermore, we want the final representation of the procedure to consist of relatively prime cubes. The proposed procedure achieves this goal by processing one cube at a time, in which the cube is expanded maximally so that a maximal number of cubes are removed. An expansion is an operation which takes a cube c and returns a cube $\hat{c} \supseteq c$. The expansion is *valid* if the replacement of c with $\hat{c}$ results in a compatible representation. The result of the procedure is order dependent and we sort the cubes in the order of decreasing size as in Espresso.

The procedure for each cube $c \in \mathcal{F}$, EXPAND1, is designed as an extension of the expand procedure of Espresso [2], and is illustrated in Figure 4.2. EXPAND1 employs a

covering matrix C which has been introduced in Espresso. C has $(t+m)$ columns and as many rows as cubes of $\mathcal{F}-\{c\}$, where t is the number of the parts of $I(c)$, which is given by the sum of the number of values that each input variable can assume. Each element of C, C_{ij}, is defined as follows:

$$\forall j \in \{1,\cdots,t\}: \quad C_{ij} = \begin{cases} 1 & \text{if the } j\text{-th part of } I(c) \text{ is 0 and the } j\text{-th part of } I(\mathcal{F}^{(i)}) \text{ is 1,} \\ 0 & \text{otherwise,} \end{cases}$$

$$\forall j \in \{1,\cdots,m\}: \quad C_{i(t+j)} = \begin{cases} 1 & \text{if } O(c)_j = 0 \text{ and } O(\mathcal{F}^{(i)})_j = 1, \\ 0 & \text{otherwise,} \end{cases}$$

where $\mathcal{F}^{(i)}$ is the i-th cube of $\mathcal{F}-\{c\}$. The covering matrix allows us to handle each part of c in a uniform way, without making any distinction among the input or the output variables.

Throughout EXPAND1, we maintain two sets of columns of C, $\mathcal{R}$ and $\mathcal{L}$, where $\mathcal{R}$ is initially a set of columns at which c has 1, and $\mathcal{L}$ is empty. $\mathcal{R}$ is called a *raising set* and $\mathcal{L}$ is called a *lowering set.* $\mathcal{R}$ is used to store the columns that have been raised, while $\mathcal{L}$ maintains the columns that have been determined not to be raised.

4.5.1 The Maximal Feasible Covering Operation

Among all the operations of EXPAND1, the maximal-feasible-covering (MFC) operation is the key, in which directions of expansion for c are determined, while other operations are used to complete the procedure efficiently. The objective of MFC is to expand c so that the maximum number of cubes of $\mathcal{F}-\{c\}$ are removed. Overall flow of the operation is as follows. First, for each cube $p \in \mathcal{F}-\{c\}$ corresponding to each row of the covering matrix C, compute the smallest cube $c^*(p)$ such that $\mathcal{F}-\{c,p\}\cup\{c^*(p)\}$ is compatible. $c^*(p)$ may not exist. We choose the smallest such cube since we want to leave freedom of expansion for eliminating other cubes. Note that $c^*(p)$ may not cover p.[2] If $c^*(p)$ exists, then we compute the maximum subset $S(p)$ of $\mathcal{F}-\{c,p\}$ such that $\mathcal{F}-\{c,p\}-S(p)\cup\{c^*(p)\}$ is compatible. Once all the cubes corresponding to the rows of C have been processed, and if no $c^*(p)$ exists, then we exit the operation. Otherwise, a cube $c^*(q)$ with the maximum cardinality of $S(q)$ is chosen. Then the rows of C corresponding to the cubes of $q\cup S(q)$ are removed and the consistency function for the new representation except c is computed as F_c. Finally, the operation is exited with the set of newly raised columns.

[2] In fact $c^*(p)$ may not cover even c, but we exclude this case in our procedure.

```
procedure EXPAND1(c, F, R, F)
begin
    F_c ← CFc(F, F);   /* compute a consistency function for F − {c} */
    C ← COVERING_MATRIX(c, F);
    R ← {j | The j-th part of c is 1.};
    L ← φ;
    while(|R| + |L| < (t + m) and C ≠ φ){
        X_E ← ESSENTIAL(c, F_c, R, R, L);
        L ← L ∪ X_E;
        C ← ELM1(C, X_E);
        J ← MFC(C, c, F_c, R, R);
        if(|J| = 0)   J ← EG(C);
        R ← R ∪ J;
        C ← ELM2(C, J);
    }
    if(|R| + |L| < (t + m))   {R, L} ← GREEDY(c, F_c, R, R, L);
    c ← RAISE(c, R);
    if(c has been expanded)   F ← CF(F_c, c);
end
```

Figure 4.2: EXPAND1

Practically, it is expensive to accomplish these procedures completely and thus we introduce some restrictions. First, $S(p)$ is restricted to those covered by $c^*(p)$, and $c^*(q)$ is chosen as the one that covers the largest number of $c^*(p)$'s over all the p's. Thus we do not compute $S(p)$ at all and the rows of C covered by $c^*(q)$ are eliminated by another operation. Now, we still need to update F_c once a cube $c^*(q)$ is chosen. Furthermore, in order to compute $c^*(p)$ for each $p \in \mathcal{F} - \{c\}$, we need to compute a consistency function for $\mathcal{F} - \{c, p\}$. These consistency functions are necessary so that we can estimate the functionality of $\mathcal{F} - \{c, q\} - S(q) \cup \{c^*(q)\}$ (and $\mathcal{F} - \{c, p\} \cup \{c^*(p)\}$) without computing the real consistency functions for the representations. However, we can observe that both of these tasks are necessary because of the fact that $c^*(p)$ may not cover p. This is because if $c^*(q)$ covers q as well as all the cubes of $S(q)$, then the functionality of $\mathcal{F} - \{c, q\} - S(q) \cup c^*(q)$ is identical with the functionality of $\mathcal{F} - \{c\} \cup c^*(q)$. Thus as long as the consistency function for $\mathcal{F} - \{c, q\} - S(q)$ is used to estimate the functionality of the entire representation, the consistency function for $\mathcal{F} - \{c\}$, F_c, can be used as alternative. Note that F_c does not reflect the functionality of $\mathcal{F} - \{c, q\} - S(q)$, but still the functionality of $\mathcal{F} - \{c, q\} - S(q) \cup c^*(q)$ can be correctly estimated with the help of $c^*(q)$. The same statement holds for the consistency function of $\mathcal{F} - \{c, p\}$. Since F_c is already available, with the additional restriction that $c^*(p) \supseteq p$, the entire procedure of MFC can be completed without computing consistency functions. Due to the enormous improvement in computational time, we employ this restriction.

What remains in MFC is a computation of $c^*(p)$, the smallest cube containing both c and p such that $\mathcal{F} - \{c, p\} \cup \{c^*(p)\}$ is compatible with the relation R. The smallest cube containing both c and p, denoted as $\hat{c}$, is obtained by taking the part-wise union between c and p [14]. In other words, if p corresponds to the i-th row of C, then for $j \in \{1, \cdots, t\}$, the j-th part of $I(\hat{c})$ is 0 if and only if $C_{ij} = 0$ and $j \notin \mathcal{R}$, while $O(\hat{c})_j = 0$ if and only if $C_{i(t+j)} = 0$ and $(t + j) \notin \mathcal{R}$ for $j \in \{1, \cdots m\}$. If R is a regular function, then $c^*(p) = \hat{c}$ if $\mathcal{F} - \{c, p\} \cup \hat{c}$ is compatible, otherwise $c^*(p)$ does not exist. For relations, however, it is claimed, as with the REDUCE procedure, that there exist cases where an incompatible representation becomes compatible by raising the output part of $\hat{c}$. One sees that raising the input part of $\hat{c}$ does not help since the image of a minterm for which the incompatibility occurs is not changed unless either the output part of $\hat{c}$ or other cubes of $\mathcal{F} - \{c, p\}$ are modified. Considering that the functionality of $\mathcal{F} - \{c, p\} \cup \{\hat{c}\}$ is identical with that of $\mathcal{F} - \{c\} \cup \{\hat{c}\}$, we define a *feasible cube* for $\hat{c}$ as a cube c^* such that $\mathcal{F} - \{c\} \cup \{c^*\}$ is

compatible with R, the input part of c^* is equal to that of $\hat{c}$, and $c^* \supseteq \hat{c}$. Then our objective is to find the smallest feasible cube for $\hat{c}$.

This is done with a similar technique used in REDUCE. Let $Y_l = \{j \in \{1, \cdots, m\} \mid O(\hat{c})_j = 0\}$, and consider a cube corresponding to a minterm $\mathbf{y} \in B^{|Y_l|}$ such that the input part is identical with that of $\hat{c}$ and the j-th output part is 0 if and only if $j \in Y_l$ and $y_j = 0$. Let $l : B^{|Y_l|} \rightarrow B$ be a function such that $l(\mathbf{y}) = 1$ if and only if $\mathcal{F} - \{c\} \cup \{c^*\}$ is compatible with R, where c^* is the cube corresponding to $\mathbf{y}$. There is one-to-one correspondence between a feasible cube and a minterm $\mathbf{y}$ such that $l(\mathbf{y}) = 1$, and thus our objective is to find the minterm $\mathbf{y}$ with the minimum number of 1's in the output part for which $l(\mathbf{y}) = 1$. We see that $l(\mathbf{y}) = 1$ if and only if the following property holds for every minterm $\mathbf{x} \in M(\hat{c})$:

$$\exists \tilde{\mathbf{y}} \in B^m : \quad R(\mathbf{x}, \tilde{\mathbf{y}}) = 1, \quad \forall j \in \{1, \cdots m\} : \quad \tilde{y}_j = \begin{cases} \hat{y}_j + y_j & \text{if } j \in Y_l, \\ 1 & \text{if } j \notin Y_l, \end{cases} \tag{4.4}$$

where $\hat{y}_j$ is the value of the j-th variable of the minterm $\hat{\mathbf{y}}$ such that $F_c(\mathbf{x}, \hat{\mathbf{y}}) = 1$. Then as seen in the previous section, using a function $L : M(\hat{c}) \times B^{|Y_l|} \rightarrow B$ such that $L(\mathbf{x}, \mathbf{y}) = 1$ if and only if (4.4) holds for $(\mathbf{x}, \mathbf{y})$, $l(\mathbf{y})$ is computed as $l(\mathbf{y}) = \neg(S_X(M(\hat{c})(\neg L(\mathbf{x}, \mathbf{y}))))$, where $L(\mathbf{x}, \mathbf{y})$ is given by

$$L(\mathbf{x}, \mathbf{y}) = S_{(T \cup Z)}(M(\hat{c}) R(\mathbf{x}, \mathbf{z}) F_c(\mathbf{x}, \mathbf{t}) \prod_{j \in Y_l} (z_j \equiv (t_j + y_j)) \prod_{j \notin Y_l} z_j). \tag{4.5}$$

Note that, unlike REDUCE, $l(\mathbf{y})$ may be tautologically zero. This is the case where there is no feasible cube for $\hat{c}$.

Now, as a variation of Theorem 4.3, it is claimed that if $l(\mathbf{y})$ is not tautologically zero, then the minterm $\mathbf{y}$ with the minimum number of 1's such that $l(\mathbf{y}) = 1$ is given by the shortest path connecting a 1 leaf to the root of a BDD of $l(\mathbf{y})$. Hence, the smallest cube $c^*(p)$ containing both c and p such that $\mathcal{F} - \{c, p\} \cup \{c^*(p)\}$ is obtained by computing $l(\mathbf{y})$ for a cube $\hat{c}$, followed by performing a shortest path algorithm.

The restriction that $c^*(p)$ must cover p brings another contribution for efficiency: once there is no such $c^*(p)$ for p at some time during the expansion of c, then there is no hope that $c^*(p)$ exists in the future. Thus we mark a row of C corresponding to p if $c^*(p)$ does not exist, so that a function $l(\mathbf{y})$ for p is never computed for the rest of the procedure. We note if we do not restrict $c^*(p)$ to cover p, then this does not hold for relations in general. The following example illustrates the situation.

Example 4.3 *Suppose that a representation $\mathcal{F}$ is compatible with R in Figure 4.3. Suppose we are in the process of expanding c. There is no cube $c^*(p) \supseteq c$ such that $\mathcal{F} - \{c, p\} \cup \{c^*(p)\}$ is compatible, and thus p cannot be eliminated. On the other hand, q is eliminated from $\mathcal{F}$ if c is expanded to $c^{(1)} = [110 \mid 101]$. Then if $c^{(1)}$ is further expanded to $c^{(2)} = [112 \mid 101]$, $\{c^{(2)}\}$ is a compatible representation and thus p is eliminated. Note that the expansion from c to $c^{(2)}$ is not valid if q is present in $\mathcal{F}$.*

Representation $\mathcal{F}$			Relation R	
cube	Input	Output	$\mathbf{x} \in B^3$	$\mathbf{y} \in B^3$
c	110	001	110	{011, 101}
p	121	101	101	{000, 101}
q	112	010	111	{111, 001, 101}

Figure 4.3: Example of Expansion for Binary-Valued Relation

4.5.2 EXPAND1

We have seen how to determine the direction of expansion for c in MFC. However, no $c^*(p)$ might exist for any $p \in \mathcal{F} - \{c\}$. In this case, an operation EG is invoked. If, in addition, C is empty, then another operation GREEDY is called. Otherwise, as Espresso does, the operation chooses a single column not in $\mathcal{R}$ or $\mathcal{L}$ with the maximum column count in C. Then a set of columns obtained by either MFC or EG are included to $\mathcal{R}$. Now C might have the rows that have a zero in every column not in $\mathcal{R}$ or $\mathcal{L}$. This means that the cubes of $\mathcal{F} - \{c\}$ corresponding to these rows are covered by c if all the columns in $\mathcal{R}$ are raised. Thus these rows are eliminated by ELM2.

The operation ESSENTIAL finds essential columns. A column $j \notin \{\mathcal{R} \cup \mathcal{L}\}$ is essential if there is no feasible cube for a cube $\hat{c}$ in which all the columns of $\mathcal{R} \cup \{j\}$ are raised. This is checked by computing $l(\mathbf{y})$ for $\hat{c}$, and seeing if l is tautologically zero. Once a set of essential columns are obtained, these are included in $\mathcal{L}$. If there is a row in C that has a 1 in one of the columns which have been included in $\mathcal{L}$, then there is no hope that the cube corresponding to the row is covered by expanding c. Therefore ELM1 eliminates all such rows of C.

In case C becomes empty but some columns are not in either $\mathcal{R}$ or $\mathcal{L}$, GREEDY is invoked, where we first examine for each such column of the input part at a time if the

column is essential. If the column is not essential then it is put in $\mathcal{R}$, otherwise included in $\mathcal{L}$. Then for a cube $\hat{c}$ in which all the columns of $\mathcal{R}$ are raised, we compute the *largest* feasible cube for $\hat{c}$, i.e. the feasible cube with the maximum number of 1's in the output part. Such a cube is obtained by computing the longest path to the 1 leaf of a BDD for $l(\mathbf{y})$ of $\hat{c}$. Namely, the j-th column of the output part such that $j \notin \mathcal{L}$ is included in $\mathcal{L}$ if there is a 0-edge incident with a node for y_j in the longest path. The rest of the columns are put in $\mathcal{R}$.

Finally all the columns in $\mathcal{R}$ are raised in c by RAISE and a consistency function for the new representation is computed. The following theorem guarantees that at the end of EXPAND1, c is prime relative to $\mathcal{F}$.

Theorem 4.4 *Suppose EXPAND1 expands c to c^+. Then c^+ is prime relative to $\mathcal{F} - \{c\} \cup \{c^+\}$.*

Proof: Suppose for contrary that there exists a cube $c^{++} \supset c^+$ for which $\mathcal{F} - \{c\} \cup \{c^{++}\}$ is compatible. Let j be any part at which c^+ is lowered and c^{++} is raised. Then j was added to $\mathcal{L}$ either by ESSENTIAL or GREEDY. In either case j was essential at that time. Let $\hat{c}$ be the cube for which the existence of feasible cubes was examined for j. Note that $M(c^{++}) \supseteq M(\hat{c})$. For a cube $p = [I(\hat{c}) \mid O(c^{++})]$, the image of any minterm $\mathbf{x} \in M(\hat{c})$ by $\mathcal{F} - \{c\} \cup \{p\}$ is equal to that by $\mathcal{F} - \{c\} \cup \{c^{++}\}$. Since there was no feasible cube for $\hat{c}$ and $p \supseteq \hat{c}$, there exists a minterm $\mathbf{x} \in M(p)$ such that the image of $\mathbf{x}$ by $\mathcal{F} - \{c\} \cup \{p\}$ is not a member of $r(\mathbf{x})$. Thus $\mathcal{F} - \{c\} \cup \{c^{++}\}$ is incompatible, which is a contradiction. ■

As with the REDUCE procedure, it is claimed that there is a case where relatively prime cube in $\mathcal{F}$ may become non-prime by expanding another cube of $\mathcal{F}$. Therefore, we iterate the entire procedure, until no cube is further expanded. At the end of EXPAND, we obtain a compatible representation in which every cube is prime relative to the representation.

4.6 Irredundant Procedure

In the IRREDUNDANT procedure, a compatible representation is made irredundant. The procedure takes one cube c at a time and checks if the representation $\mathcal{F} - \{c\}$ is compatible. If this is the case, c is removed. The results of the procedure depends upon the

order of the cubes, and the procedure processes them in decreasing order of size. However, since the IRREDUNDANT procedure is always applied as a successor of the EXPAND procedure which also sorts the cubes in the order of decreasing size and since the EXPAND procedure is iterated until no cube changes, the cubes are already sorted when given as input to the IRREDUNDANT procedure. Thus we do not sort them in this routine. The IRREDUNDANT procedure is considered as a special case of the REDUCE procedure and thus it is claimed that as with Example 4.2 an irredundant cube in $\mathcal{F}$ may become redundant once another cube is modified. Thus we iterate this procedure, until no cube is removed. Note that the decreasing order of the cubes is preserved even if some cubes are removed.

Chapter 5

Implementation and Results

The proposed procedure has been fully implemented in the program called GYOCRO. The system computes an initial representation only if it is not given externally. Once an initial representation has been verified to be compatible, the proposed procedure is applied. We use the same data structure as Espresso-MV [14] to represent sum-of-products expressions. In the EXPAND procedure, employing the techniques introduced in [14], we have implemented the proposed algorithm by directly using a representation $\mathcal{F}$ instead of a covering matrix.

We have tried some examples of binary-valued relations to the proposed procedure and measured the CPU time required to complete the minimization. The results were compared with the other two approaches [8, 17] developed for binary-valued relations, where the programs for both procedures have been provided to the authors. Table 5.1 shows the number of the product terms and the CPU time (seconds) measured on a DECstation 5000 for all three methods. **Exact** is the exact procedure [17], **Herb** is a heuristic approach proposed in [8] and **GYOCRO** is the proposed approach. The proposed approach performs quite well both in CPU time and results.

In order to speed up the proposed method, we tried a modification. In the original procedure, each of the REDUCE, EXPAND, and IRREDUNDANT procedures is iterated until no cube of the representation is replaced by a different cube, or is removed from the representation. This is because we want to guarantee that every cube of the final representations of these procedures is maximally reduced, relatively prime, and irredundant respectively. We tried another way where each of these procedure is exited after a single sweep of the representation. In this case some cubes of the final representation of the entire

Name	In	Out	Exact		Herb		GYOCRO		GYOCRO-I	
			Terms	Time	Terms	Time	Terms	Time	Terms	Time
int1	4	3	5	15.94	8	0.2	5	0.65	5	0.39
int2	4	3	5	14.93	8	0.2	5	0.60	5	0.40
int3	4	3	7	1.87	8	0.2	7	1.79	7	1.04
int4	4	3	7	0.03	9	0.5	9	1.25	9	0.83
int5	4	3	7	0.18	8	0.6	7	1.96	7	0.59
int6	4	3	7	0.20	7	0.5	7	1.01	7	0.62
int7	6	4	time out		16	2.2	15	7.99	15	4.58
int8	4	3	8	0.02	8	0.4	8	3.12	8	1.71
int9	6	4	time out		23	2.9	14	8.75	14	5.18
int10	6	4	25	42446.33	32	3.6	25	8.23	25	4.89
c17a	5	3	5	0.51	5	0.5	5	0.82	5	0.67
c17b	5	3	7	18.15	7	0.4	7	1.10	7	0.75
c17c	5	3	12	8.08	13	2.4	16	3.40	16	1.97
c17d	5	3	12	0.96	15	2.6	16	3.37	16	1.92
c17e	5	3	5	0.49	5	1.2	5	1.73	5	1.09
c17f	5	3	5	1.01	7	1.4	6	1.17	6	0.71
c17g	5	3	7	0.26	7	0.5	7	2.39	7	1.49
c17h	5	3	5	0.66	5	1.9	5	0.92	5	0.64
c17i	5	3	13	0.85	14	1.7	15	2.89	15	1.68
she1	7	3	6	70.26	9	96.5	6	3.23	6	1.70
she2	5	5	time out		14	18.0	10	8.91	10	4.47
she3	7	4	time out		10	358.5	9	7.41	9	4.55
she4	5	6	time out		27	22.4	20	13.63	20	7.49
yosi	5	13	*	*	16	19.6	11	77.43	11	52.99
gr	15	11	*	*	×	×	53	453.94	53	224.68
b9	16	5	*	*	452	1439.3	270	565.99	270	275.30
int15	24	14	*	*	×	×	131	1751.00	131	697.90
vtx	27	6	*	*	424	1272.3	*	*	424	12596.52
ib	48	17	*	*	out of memory		out of memory		out of memory	

* : the method has not been applied for the example.

× : the method does not produce a correct result for the example.

Table 5.1: Experimental Results

procedure may not be prime relative to the representation or irredundant. However, we note that every cube is guaranteed to be a c-prime since the cube was made prime relative to some representation compatible with the relation. The results of the modified procedure are shown in the last column of Table 5.1. The CPU time was improved roughly by a factor of two and the results were precisely the same.

Chapter 6

Conclusion

We have considered the minimization problem of multiple-valued relations. We have shown that the problem of minimizing multiple-valued relations are naturally described with consistency functions, and have proposed a heuristic procedure which achieves the minimizations using consistency functions. Based on the paradigm of a state-of-the-art two level minimization technique for regular functions, in which a final solution is obtained from an initial solution through expand and reduce procedures, we have described some special properties of relations that do not hold for functions. These properties are easily handled through consistency functions represented as MDD's. The proposed procedure has been fully implemented in the program GYOCRO. The results are encouraging in the sense that, for those examples where we know the minimum solution, our heuristic minimizer reproduces the result most of the times or comes very close. On large examples where the exact minimizer can not complete, our method outperforms the only other existing heuristic binary-valued relation minimizer. Computing times are reasonable.

Appendix A

GYOCRO: User's Manual

This chapter provides the user's manual for GYOCRO. GYOCRO has been developed so that it can be used as either a stand-alone system or an internal package of another system, e.g. misII [3]. In Section A.1, we present a general description on the use of GYOCRO as a stand-alone system. Section A.2 describes the format of an input file that GYOCRO takes. Section A.3 provides a manual for the internal use of GYOCRO, in which we present interface data structure as well as a set of exported functions.

A.1 System Description

GYOCRO takes as input a table format of a binary-valued (or multiple-valued) relation, and produces a minimal sum-of-product expression compatible with the relation. GYOCRO is used from a command line by specifying as follows.

gyocro [*options*] [*file1*] [*file2*]

GYOCRO reads the *file1* provided, performs the minimization, and writes the result in the *file2* (or standard output if *file2* is not specified). The format of the output file is a PLA format employed in Espresso [14]. The format of the input file is described in Section A.2. GYOCRO automatically verifies that the minimized function is compatible with the given relation. The command line options that GYOCRO can accept are listed below.

-g By default, GYOCRO performs the modified version of the minimization procedure which has been described in Chapter 5. With this option, the original minimization procedure is performed, and thus it is guaranteed that the minimized result is relatively prime and irredundant.

-i With **gyocro** -i [*file0*] [*file1*] [*file2*], GYOCRO takes a sum-of-products expression specified in *file0* as an initial representation. The *file0* must be given in a PLA format. GYOCRO verifies whether the expression specified in *file0* is compatible with the relation specified in *file1* before performing the minimization.

-t With this option, GYOCRO first identifies whether the given relation is an incompletely specified function. If this is the case, Espresso [14] is invoked to perform the minimization. Otherwise, the minimization procedure for relations is employed.

-c With **gyocro** -c [*file0*] [*file1*] [*file2*], GYOCRO verifies whether a sum-of-products expression specified in *file0* is compatible with the relation specified in *file1*. If it is not compatible and if *file2* is specified, then the set of minterms of the inputs for which incompatibility arises is written in *file2* as a PLA format.

A.2 Input File Format

GYOCRO accepts as input a table format of a binary-valued or a multiple-valued relation. This is described with keywords embedded in the input file to specify the size of input and output variables.

The following four keywords are recognized by GYOCRO. If the given relation is binary-valued, **.i** and **.o** are required, while **.mv** is required if the relation contains a multiple-valued variable.

.i d The number of the input variables (d).

.o d The number of the output variables (d).

.mv num_var num_binary_var d1 ⋯ dn The sum of the number of input variables and the number of the output variables (num_var), the number of binary input variables (num_binary_var), and the size of each of the multiple-valued variable, where dn is the number of output variables.

.e The end of the file.

A table format of a relation is a set of rows, where each row consists of the input part and the output part. The input part and the output part may be separated by a vertical bar |. We first describe a table format for a binary-valued relation. An example of a binary-valued relation in a table format is shown in Figure A.1. The input part of each row corresponds to a product term of the input variables. Namely, each position of the input part corresponds to an input variable where a 0 implies the corresponding input variable appears as the complemented literal in the product term, a 1 implies the corresponding input variable appears as the uncomplemented literal in the product term, and a 2 or − implies the corresponding input variable does not appear in the product term. The output part of each row consists of a set of strings separated by a space. Each string corresponds to a product term of the output variables, and thus the length of the string is equal to the number of the output variables. Each string can contain characters 0, 1, 2, and −, and the characters are used in the same way with the input part. Therefore, each row consists of a product term of the input variables and a set of product terms of the output variables. The table format specifies a relation in such a way that for a minterm x of the input set, the image of x by the relation is the sum of the product terms of the output variables for all the rows whose input part covers x. One exception is that the image of x by the relation is

```
.i 3
.o 3
120  |  001 100
202  |  010
210  |  121
010  |  010 100
.e
```

Figure A.1: A table format of a Binary-Valued Relation

Input	Output
000	{010}
001	{010}
010	{010, 100, 101, 111}
011	{000}
100	{001, 100, 010}
101	{010}
110	{001, 100, 101, 111}
111	{000}

Figure A.2: The Image of the Relation in the Previous Example

the minterm in which all the output variables appear as complemented literals if x is not covered by any row of the table. The images for all the minterms of the relation shown in Figure A.1 is shown in Figure A.2.

A multiple-valued relation can be specified in a similar way. In this case, the size of the relation must be specified using the keyword **.mv**. If the relation also contains binary-valued variables, they should be given as the first variables for each row. Note that a binary-valued variable can be specified as a multiple-valued variable with 2 values, so that it is specified in an arbitrary place in the row. The binary variables are specified as described above. Each of the multiple-valued variables is specified as a bit vector with the length equal to the number of the values the variable can assume, where a 0 implies the variable does not take the corresponding value in the product term corresponding to the row, and a 1 implies the variable takes the value in the product term. A vertical bar | may be used to separate multiple-valued variables. Figure A.3 shows the relation given in Figure A.1 in which the third input variable is treated as a multiple-valued variable with size 2.

A.3 GYOCRO for Internal Use

GYOCRO has been developed so that it can be used as an internal package of another system, e.g. misII [3]. GYOCRO provides user friendly interface data structure

```
.mv 4 2 2 3
12  |10|  001 100
20  |11|  010
21  |10|  121
01  |10|  010 100
.e
```

Figure A.3: A table format of a Multiple-Valued Relation

and a set of exported functions. It employs stand-alone packages for BDD's and MDD's, both of which have been developed at U.C. Berkeley. Documentation for these packages is found in [10].

A.3.1 Data Structure

In order to use GYOCRO internally, the consistency function of the relation being minimized and the information on the size of the relation must be provided. If the given relation is binary-valued, then the consistency function is represented by a BDD. In this case, a data structure br_t should be used. The entries of br_t are listed below.

1. bdd_t *cR;
   The consistency function of the relation being minimized.

2. bdd_manager *mg;
   The manager of *cR.

3. unsigned int *Xvar;
   The BDD ID's of the input variables of the relation. The size of Xvar is equal to the number of the input variables, and the Xvar[i] specifies the ID of the BDD variable corresponding to the i-th input variable.

4. unsigned int *Zvar;
   The BDD ID's of the output variables of the relation. The size of Zvar is equal to the number of the output variables, and the Zvar[j] specifies the ID of the BDD variable corresponding to the j-th output variable.

If the relation being minimized is multiple-valued, then the consistency function is repreented by a MDD, and a data structure mvr_t should be used. The entries of mvr_t is listed below.

1. mdd_t *cR;
   The consistency function of the relation being minimized.

2. mdd_manager *mg;
   The manager of *cR.

3. int *Xvar;
   The MDD ID's of the input variables of the relation. The size of Xvar is equal to the number of the input variables, and the Xvar[i] specifies the ID of the MDD variable corresponding to the i-th input variable.

4. int *Zvar;
   The MDD ID's of the output variables of the relation. The size of Zvar is equal to the number of the output variables, and the Zvar[j] specifies the ID of the MDD variable corresponding to the j-th output variable.

The size of the relation, e.g. the number of the input variables, is specified by a data structure brsize_t. There is an exported function that sets up the entries of brsize_t.

In summary, if a binary-valued relation is minimized, one should provide br_t and brsize_t, while mvr_t and brsize_t must be given for minimizing a multiple-valued relation.

A.3.2 Exported Functions

A set of exported functions are listed below. In fact, most of the functions take data structure different from br_t, mvr_t, or brsize_t, and thus the user should convert the original data structure to appropriate ones using converting functions.

br_t *
br_alloc()

Allocates a data structure br_t. It is the user's responsibility to set up the entries of the data.

void
br_free(r)
br_t *r;

Frees up r. The consistency function cR and the manager mg are not freed.

mvr_t *
mvr_alloc()

Allocates a data structure mvr_t. It is the user's responsibility to set up the entries of the data.

void
mvr_free(r)
mvr_t *r;

Frees up r. The consistency function cR and the manager mg are not freed.

brsize_t *
set_brsize(b_nin, m_nin, mnum, nout)
int b_nin, m_nin, *mnum, nout;

Returns a data structure brsize_t with the number of the binary input variables b_nin, the number of the multiple-valued variables m_nin, and the number of the output variables nout. mnum is an array of integers with the size m_nin in which mnum[i] provides

the number of the values that the i-th multiple-valued variable can assume.

void
brsize_free(size)
brsize_t *size;
Frees up brsize_t.

tbmvbr_t *
tbmvbr_alloc()
Allocates a data structure tbmvbr_t, which is used to read a relation specified as a table format in a file.

int
read_mvbr(r, file)
tbmvbr_t *r;
char *file;
Reads a relation specified as a table format in a file with name file to r. 0 is returned if the relation is successfully read. Otherwise 1 is returned.

void
tbmvbr_free(r)
tbmvbr_t *r;
Frees up r.

mvbr_t *
brt2mvbrt(r)
br_t *r;
Converts br_t to a data structure mvbr_t, which is used in most of the minimization routines.

mvbr_t *
mvrt2mvbrt(r)
mvr_t *r;
Converts mvr_t to mvbr_t.

mvbr_t *
tbmvbrt2mvbrt(r)
tbmvbr_t *r;
Converts tbmvbr_t to mvbr_t.

void
mvbr_free(R)
mvbr_t *R;
Frees up R. The BDD manager used in this data structure is not freed in this func-

tion. It is the user's responsibility to free the manager using a function of a BDD package.

```
pset_family
mvbr_cfnc2pla(R, size)
mvbr_t *R;
brsize_t *size;
```

Returns a compatible representation for the relation R in a PLA format used in Espresso. This routine newly allocates cube data structure defined in Espresso and frees it up at the end of the routine. Therefore, cube data structure allocated before calling this routine will be destroyed.

```
int
mv_chk_compatible(F, R, cF)
pset_family F;
mvbr_t *R;
mdd_t **cF;
```

Checks whether a representation F is compatible with the relation R. The consistency function of F is returned as cF. If the representation is compatible, 1 is returned. Otherwise 0 is returned. It is assumed that cube data structure has been set up reflecting the representation F.

```
int
gyocro(R, init_F, fnl_F, iterate)
mvbr_t *R;
pset_family init_F, *fnl_F;
int iterate;
```

Minimizes the relation R starting with an initial representation init_F. The minimized result is returned to fnl_F. If iterate is 0, a modified procedure that is completed faster is used. Otherwise, the original procedure is invoked. If the minimization succeeds, 1 is returned. Otherwise 0 is returned.

```
int
tea_identify(R, f, d, r)
mvbr_t *R;
pset_family *f, *d, *r;
```

Identifies whether the relation R is an incompletely specified function. If this is the case, 1 is returned, and the on-set, the don't-care set, and the off-set will be retuned to f, d, and r, respectively. Otherwise 0 is returned and null pointers are set to f, d, and r. It is assumed that cube data structure has been set up reflecting the size of the input and the output variables of R.

```
int
tea_party(R, size, init_F, fnl_F, iterate, cube_set)
mvbr_t *R;
```

```
brsize_t *size;
pset_family init_F, *fnl_F;
int iterate, cube_set;
```

Minimizes the relation R starting with an initial representation init_F. The minimized result is returned to fnl_F. The initial representation may not be given, in which case init_F should be set to null. If cube_set is not 0, it is assumed that cube data structure has been already set up reflecting the size of the relation R. Otherwise, cube data structure is newly allocated. Unlike **gyocro**(), **tea_party**() identifies whether the relation R is a function or not. If it is a function, **espresso** is invoked, otherwise **gyocro** is used. If the minimization succeeds, 1 is returned. Otherwise 0 is returned.

```
void
mvbr_print_cover(F, fp)
pset_family F;
FILE *fp;
```

Writes a representation F to a file fp. Failure occurs unless cube data structure has been already set up reflecting the size of F.

Bibliography

[1] R. K. Brayton. New Directions in Logic Synthesis. In *Proceedings of the Synthesis and Simulation Meeting and International Interchange*, Kyoto, Japan, 1990.

[2] R. K. Brayton, G. D. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis.* Kluwer Academic Publishers, Boston, 1984.

[3] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. MIS: Multiple-Level Logic Optimization System. *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, Vol. CAD-6(No. 6):1062 – 1081, November 1987.

[4] R. K. Brayton and F. Somenzi. Boolean Relations and the Incomplete Specification of Logic Networks. In *International Conference on Very Large Scale Integration*, Munich, August 1989.

[5] R. E. Bryant. Graph Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, Vol. C-35(No. 8):677–691, August 1986.

[6] J. Darringer, W. Joyner, L. Berman, and L. Trevillyan. Logic Synthesis through Local Transformations. *IBM J. Res. Develop.*, pages 272–280, JULY 1981.

[7] G. De Micheli, R. K. Brayton, and A. Sangiovanni-Vincentelli. Optimal State Assignment for Finite State Machines. *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, CAD-4:269 – 285, July 1985.

[8] A. Ghosh, S. Devadas, and A. R. Newton. Heuristic Minimization of Boolean Relations using Testing Techniques. In *IEEE International Conference on Computer Design*, Cambridge, September 1990.

[9] S. J. Hong, R. G. Cain, and D. L. Ostapko. MINI: A Heuristic Approach for Logic Minimization. *IBM J. Res. Develop.*, pages 443–458, September 1974.

[10] T. Kam and R. K. Brayton. Multi-valued Decision Diagrams. Technical Report M90/125, U.C. Berkeley, 1990.

[11] L. Lavagno, S. Malik, R. K. Brayton, and A. Sangiovanni-Vincentell. MIS-MV: Optimization of Multi-level Logic with Multiple-valued Inputs. In *IEEE International Conference on Computer-Aided Design*, 1990.

[12] B. Lin and F. Somenzi. Minimization of Symbolic Relations. In *IEEE International Conference on Computer-Aided Design*, November 1990.

[13] E. J. McCluskey Jr. Minimization of Boolean Functions. *Bell System Technical Journal*, Vol. 35:1417–1444, November 1956.

[14] R. L. Rudell and A. Sangiovanni-Vincentelli. Multiple-Valued Minimization for PLA Optimization. *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, Vol. CAD-6(No. 6):727 – 750, September 1987.

[15] T. Sasao. An Application of Multiple-Valued Logic to a Design of Programmable Logc Arrays. In *International Symposium on Multiple Valued Logic*, 1978.

[16] T. Sasao. Input Variable Assignment and Output Phase Optimization of PLA's. In *IEEE Transaction on Computers*, October 1984.

[17] F. Somenzi and R. K. Brayton. An Exact Minimizer for Boolean Relations. In *IEEE International Conference on Computer-Aided Design*, November 1989.

[18] A. Srinivasan, T. Kam, S. Malik, and R. K. Brayton. Algorithms for Discrete Function Manipulation. In *IEEE International Conference on Computer-Aided Design*, pages 92–95, November 1990.

[19] Y. Watanabe and R. K. Brayton. Incremental Synthesis for Engineering Changes. In *International Workshop on Logic Synthesis*, May 1991.