Multiple Boolean Relations

Ellen M. Sentovich, Vigyan Singhal, and Robert K. Brayton
Department of Electrical Engineering and Computer Science
University of California, Berkeley, CA 94720

Abstract

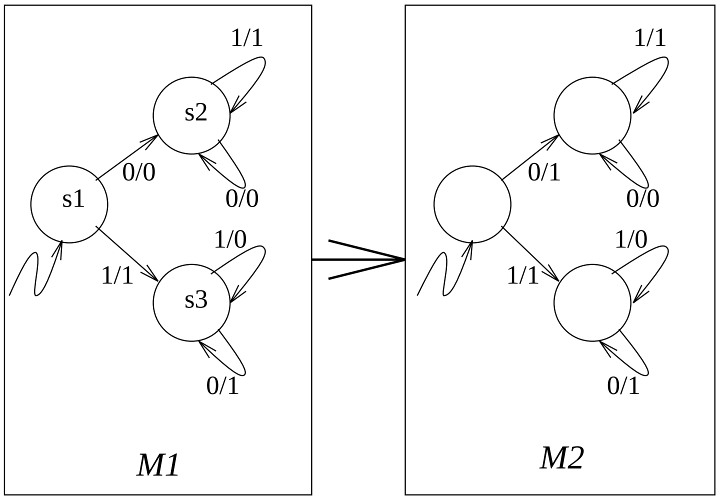
Flexibility in selecting the Boolean functions to implement a digital circuit has various forms which have been studied in the literature such as don't care conditions, Boolean relations, and synchronous recurrence equations. Each of these represents a particular degree of flexibility that may be given in the description, inherent in the current representation, or derived from the surrounding environment. This flexibility is used to find an optimal implementation. In this paper, we propose a **Multiple Boolean Relation** (**MBR**) as a model that encompasses all degrees of freedom in choosing a set of Boolean functions to implement. The novel aspect of MBRs is their capability of expressing *all* types of flexibility for combinational logic-level functions, both combinational flexibility and flexibility derived from a sequential environment. This formulation unifies some of the recent work in logic synthesis, which has introduced new types of flexibility. We give examples of synthesis problems in which MBRs arise and are the only model that represents all the flexibility available. We offer algorithms for obtaining an optimal solution to an MBR.

1 Introduction

Optimization of Boolean functions and Boolean relations is an important step in the synthesis of digital circuits. The quality of the final implementation is strongly influenced by the ability to exploit *flexibility* in realizing the underlying functions. Here, *flexibility* means *choice* in the actual function implemented (as opposed to choice in how that function is implemented). Flexibility in various forms at the functional level has been studied extensively, and includes don't care conditions for optimization of combinational [2, 1, 9, 12] and sequential [6, 10] two-level and multilevel circuits as single entities (each output can be optimized independently), Boolean relations [4, 3, 15] which express relationships between outputs and are derived from the surrounding environment, and synchronous recurrence equations [7] for optimization of sequential circuits in the context of continuous operation over time.

A **Multiple Boolean Relation** (**MBR**) is a set of Boolean relations that encompasses all possible flexibility in choosing the functions to be implemented. It is a way of representing any collection of multioutput functions. MBR's arise naturally in synthesis problems that occur in the context of a surrounding environment and that have special constraints imposed upon them that are either internal or derived from the environment. MBR's include as special cases all types of flexibility published thus far. The nature and origin of the flexibility is not as important as the fact that it can be captured in a multiple Boolean relation. While we address only binary-valued functions and relations, any solution to a multivalued function or relation is contained in the MBR representation.

Example 1.1 *A system of two interacting finite state machines is shown in Figure 1(a) [14]. Machine $M1$ has a single output that is connected to the input of Machine $M2$. Analysis of this system reveals that the output of $M1$ can be changed slightly without affecting the I/O behavior of the combined machines. This is depicted in the state transition table in Figure 1(b), where each column in the output part contains a set of output values for the edges in $M1$. There are four possible output functions which cannot be combined into a single incompletely specified function. A multifunction, the single-output special case of a multiple Boolean relation, must be used to represent this flexibility.*



PS	IN	OUT			
s1	0	0	0	1	1
s1	1	1	0	0	1
s2	0	0	0	1	1
s2	1	1	1	0	0
s3	0	1	0	0	1
s3	1	0	1	1	0

(b)

(a)

Figure 1: (a) Finite state machine $M1$ drives machine $M2$. (b) The specification for permitted output labels on machine $M1$, given that the behavior of the two-FSM system remains unchanged.

There are several ways in which multiple Boolean relations arise in synthesis problems. First, they may arise precisely in multiple Boolean relation form, i.e., the problem as given is to find an optimum function contained in a set of Boolean relations. Example 1.1 is in this category. (In this simple example, the flexibility derived applies to only a single output but the same phenomenon could arise in a similar multiple output case.) The allowed output functions cannot be combined and expressed as either a Boolean relation or an incompletely specified function. The second source is a single Boolean relation with a set of constraints. The problem is to find an optimum function that is contained in the Boolean relation and that satisfies the constraints. In this case, a single relation is given, but the imposed constraints exclude some of the functions in the relation from the solution space. The range of solutions is a set of functions which may no longer be expressible by a single Boolean relation. Finally, multiple Boolean relations can arise from sequential circuits when optimizations over several time periods are considered. Examples from recent literature of all of these sources of MBR's are given in more detail in Section 4.

We begin by reviewing the classes of Boolean functions and relations that have been studied to date, with particular attention to the types of flexibility that can be expressed within each class. This is followed by the introduction of multiple Boolean relations, a study of their properties and methods for solving them, and applications of MBR's in synthesis problems.

In the sequel, we assume a set of n input variables $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$ and m output variables $\mathbf{z} = \{z_1, z_2, \ldots, z_m\}$. x^i represents an input minterm, $x^i \in B^n$, and z^i represents an output minterm, $z^i \in B^m$. A superscripted output variable indicates several choices for a single function, e.g., z_1^1, z_1^2 indicates that the output z_1 can be implemented by either function z_1^1 or z_1^2. A *legal* or *permissible* function is one that is allowed by the specification.

2 Boolean Functions and Boolean Relations

The hierarchy of Boolean functions and relations organized according their capability for expressing flexibility is shown in Figure 2. An arrow from representation A to representation B indicates that A is strictly more expressive than B: A can represent a larger set of *specifications*, but A is usually more difficult to represent and solve than B. Each class of Boolean function and Boolean relation is discussed below. The function or relation is defined as a mapping or set of mappings. For comparison, the number of possible *specifications* in each category is given. Current methods for obtaining a single solution (a function or set of functions

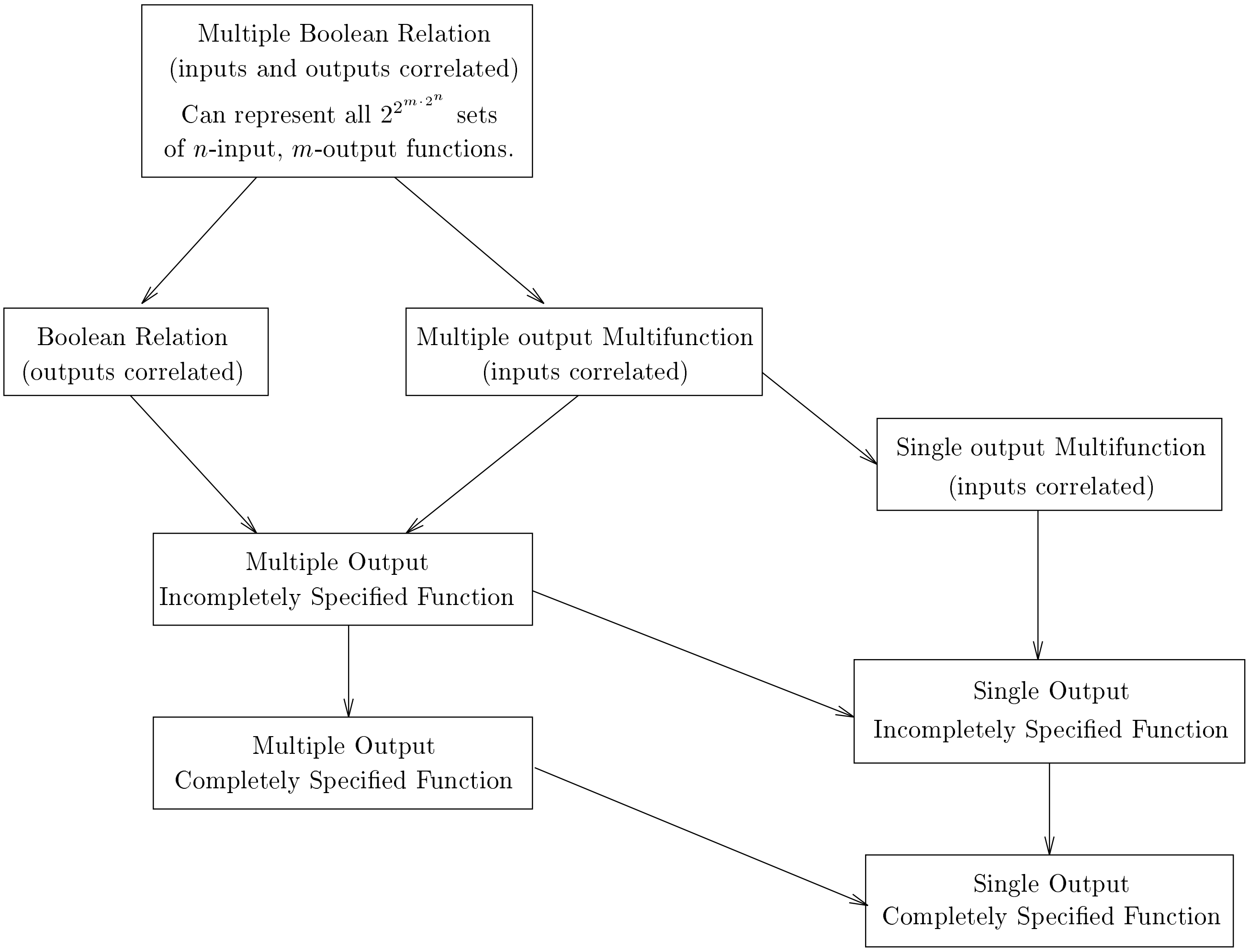


Figure 2: Boolean Function/Relation Hierarchy

allowed by the specification) and an optimal solution are outlined. An example in each class is given in Figure 3. For each specification type, an example specification is given and the legal output functions for the specification are enumerated.

2.1 Completely Specified Functions

Completely specified functions lie at the bottom of the hierarchy. A single output completely specified function f is a many-to-one mapping $f : B^n \rightarrow B$; there are 2^{2^n} possible functions f. A multiple output completely specified function F is a many-to-one mapping $F : B^n \rightarrow B^m$; there are $\left(2^{2^n}\right)^m = 2^{m \cdot 2^n}$ possible functions F. In both cases, there is no choice in the function to be implemented; each input minterm maps to exactly one output minterm.

Completely specified functions are typically given in a sum-of-products representation for each output, where each product term represents a set of minterms that map to 1 for the output (collectively referred to as the *ON-set*). A single solution is given by the description itself, and an optimal implementation is obtained by using standard two-level minimization techniques (e.g., [2]). (While completely specified functions can be represented in multilevel form, this introduces new variables which in turn introduces some don't care conditions inherent in the representation. Consequently, this representation is placed in the incompletely specified function class.)

SPECIFICATION — LEGAL FUNCTIONS FOR $z_1 z_2$ (or z_1)

Multiple Boolean Relation

x_1	x_2	$(z_1 z_2)^1$	$(z_1 z_2)^2$
0	0	00, 11	01, 10
0	1	01	00, 11
1	0	01, 11	00
1	1	10, 11	00, 10

=

00	00	00	00	11	11	11	11	01	01	01	01	10	10	10	10
01	01	01	01	01	01	01	01	00	00	11	11	00	00	11	11
01	01	11	11	01	01	11	11	00	00	00	00	00	00	00	00
10	11	10	11	10	11	10	11	00	10	00	10	00	10	00	10

Boolean Relation

x_1	x_2	$(z_1 z_2)$
0	0	00, 11
0	1	01
1	0	01, 11
1	1	10, 11

=

00	00	00	00	11	11	11	11
01	01	01	01	01	01	01	01
01	01	11	11	01	01	11	11
10	11	10	11	10	11	10	11

Multiple output Multi-function

x_1	x_2	z_1^1	z_1^2	z_1^3	z_2^1	z_2^2	z_2^3
0	0	0	0	0	0	1	0
0	1	0	0	1	1	0	0
1	0	0	1	0	1	1	1
1	1	1	0	0	0	0	1

=

00	01	00	00	01	00	00	01	00
01	00	00	01	00	00	11	10	10
01	01	01	11	11	11	01	01	01
10	10	11	00	00	01	00	00	01

Multiple Output Incompletely Specified Function

x_1	x_2	z_1	z_2
0	0	0	2
0	1	0	1
1	0	2	1
1	1	1	0

=

00	01	00	01
01	01	01	01
01	01	11	11
10	10	10	10

Multiple Output Completely Specified Function

x_1	x_2	z_1	z_2
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

=

00
01
01
10

Single output Multi-function

x_1	x_2	z_1^1	z_1^2	z_1^3
0	0	0	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

=

0	0	0
0	0	1
0	1	0
1	0	0

Single Output Incompletely Specified Function

x_1	x_2	z_1
0	0	0
0	1	0
1	0	2
1	1	1

=

0	0
0	0
0	1
1	1

Single Output Completely Specified Function

x_1	x_2	z_1
0	0	0
0	1	0
1	0	0
1	1	1

=

0
0
0
1

Figure 3: Examples of Boolean Functions and Relations

2.2 Incompletely Specified Functions

A single output incompletely specified function ff is a many-to-one mapping $\mathit{ff} : B^n \rightarrow D$, where $D = \{0, 1, 2\}$ and 2 indicates that the corresponding output is unspecified (it is allowed to be either 0 or 1). There are 3^{2^n} possible single output incompletely specified functions. A multiple output incompletely specified function $\mathcal{F}$ is a many-to-one mapping $\mathcal{F} : B^n \rightarrow D^m$; there are $3^{m \cdot 2^n}$ possible functions. The input minterms that map to 2 for the output z_i are collectively referred to as the don't care or *DC-set* for z_i. Each minterm in the don't care set for z_i represents a *choice* that can be made for the value of z_i, and each choice can be made independently.

Incompletely specified functions typically arise as functions that have external don't cares (don't cares given by the user or derived from the environment) or as functions in a multilevel circuit that have don't cares based on the particular interconnection of gates. The description is a function for the *ON-set* and a function for the *DC-set*, both given in sum-of-products form. A single solution is given by the description itself with any choice for the don't care minterms. An optimal implementation is obtained by using standard two-level minimization techniques or standard multilevel minimization techniques that derive a don't care set for each function and use two-level minimization techniques to optimize it.

2.3 Multifunctions

A single output multifunction $\mathbf{f}$ is a set of many-to-one mappings $\mathbf{f} : B^n \rightarrow B$. Each mapping represents a valid function for the single output. The number of specifications is the combination (power set) of all possible single output completely specified functions: $2^{2^{2^n}}$. A multiple output multifunction $\mathbf{F}$ is m sets of mappings $\mathbf{F} : B^n \rightarrow B$. Each set of mappings represents a set of valid functions for one of the outputs. Since the outputs are chosen independently, the number of possible specifications is the product of the number of specifications for each output: $\left(2^{2^{2^n}}\right)^m = 2^{m \cdot 2^{2^n}}$.

Multifunctions allow a *choice* among several given completely specified functions. Each output function is chosen independent of the others. As Damiani et al. observed in [7] in the context of synchronous recurrence equations, the inputs are *correlated*. Given an input minterm, a choice for one output may constrain the choice made for that output under a different input minterm, but it will not affect the choices on the other outputs. In the multiple output multifunction example in Figure 3, a choice of $z_1 = 0$ for $x_1x_2 = 10$ and $x_1x_2 = 01$ forces the choice of $z_1 = 1$ for $x_1x_2 = 11$; the flexibility in x_2 is unaffected by this selection. An incompletely specified function is a special case of a multifunction in which the choices are independent.

Multifunctions have not been studied to date because they express a type of flexibility that has only recently arisen naturally in logic synthesis problems (except for the special case of multifunctions that can be expressed as incompletely specified functions). In addition, most examples requiring multifunction representations are multiple output examples that arise from Boolean relations (e.g., the outputs are not independent), and so require multiple Boolean relations. Since multifunctions are a special case of multiple Boolean relations, we defer the discussion of their representation and solution to Section 3.

2.4 Boolean Relations

A Boolean relation $\mathcal{R}$ is a many-to-many multiple output mapping $\mathcal{R} : B^n \rightarrow B^m$. Each of 2^n input minterms maps to a subset of the 2^m possible output minterms, so there are $2^{2^n \cdot 2^m}$ possible Boolean relations; $\mathcal{R} \subseteq B^n \times B^m$. Boolean relations allow a *choice* for each input minterm among several output minterms. A value chosen for a particular output may force the values on some remaining outputs; the output values are *correlated.*

Boolean relations typically arise in the context of optimizing a function that is part of a larger system. Information derived from the surrounding environment is captured as a Boolean relation and used to optimize the given function. A single solution to a Boolean relation is found by selecting a function for one output, and successively choosing functions for the remaining outputs while satisfying the constraints between outputs. Both exact [3] and heuristic [15] techniques for finding optimum and optimal solutions based on two-level implementations have been published. These techniques are similar to two-level minimization techniques.

3 Multiple Boolean Relations

Multiple Boolean relations can arise in several ways as discussed in Section 1. Each application may have its own efficient representation and solution methods particular to its origins. Since they all share the underlying model of multiple Boolean relations, we first study the properties of general MBR's.

3.1 Definition

Definition 3.1 *A* **Multiple Boolean Relation** $\mathcal{M}$ *is a set of Boolean relations* $\mathcal{M} = \{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_l\}$, *where* $\mathcal{R}_i \subseteq B^n \times B^m$.

Theorem 3.1 *A multiple Boolean relation can be used to represent* **any set** *of n-input, m-output functions.*

Proof. Follows directly from the definition. Each function is also a (trivial) relation, so one MBR expressing all the functions is simply the collection of functions. □

$\mathcal{M}$ represents a collection of multiple output functions. There are $2^{m \cdot 2^n}$ such functions and $\mathcal{M}$ represents a subset of these, so there are $2^{2^{m \cdot 2^n}}$ possible specifications. Each Boolean relation in the MBR expresses a set of functions in which the *outputs are correlated*, while the set of mappings together express Boolean relations in which the *inputs are correlated.* In the MBR example in Figure 3, a choice of $z_2 = 1$ for $x_1x_2 = 11$ forces both $z_1 = 1$ for $x_1x_2 = 11$ (output correlation) and precludes the choice of $z_1z_2 = 01, 10$ for $x_1x_2 = 00$, $z_1z_2 = 00, 11$ for $x_1x_2 = 01$, and $z_1z_2 = 00$ for $x_1x_2 = 10$ (input correlation).

Recall from Section 2.3 there are $2^{m \cdot 2^{2^n}}$ multiple output multifunctions and $2^{2^n \cdot 2^m}$ Boolean relations. With 2 inputs and 2 outputs, there are 2^8 functions, 2^{16} Boolean relations, 2^{32} multiple output multifunctions, and 2^{256} multiple Boolean relations. The number of Boolean relations increases faster than the number of multiple output multifunctions as m increases because Boolean relations capture output correlations. Similarly, the number of multiple output multifunctions increases faster than the number of Boolean relations as n increases. The number of multiple Boolean relations is greater than or equal to the number of Boolean relations and multiple output multifunctions for all n and m.

The multiple Boolean relation in Figure 3 is represented in truth table form. This is not a canonical form since each relation represents a set of functions, and a function may be contained in several relations. Furthermore, techniques for finding the minimum function contained in the set of Boolean relations are not immediately apparent from this representation. The next section begins with a set of definitions that are used to identify a canonical form for an MBR and to construct single functions that represent MBR's.

It is important to emphasize that MBRs represent simply an arbitrary set of multiple ouptut functions. These could be termed "functions sets", but the term multiple Boolean relations was chosen for two reasons. First, a collection of Boolean relations is the most compact explicit representation of an arbitrary collection of functions (see Section 3.2). Second, *Boolean relation* indicates output correlation and *multiple* indicates input correlation.

3.2 Representing an MBR with a Single Function

The following definitions are analogous to those found in the literature on two-level minimization and Boolean relation minimization. Recall from [4] that a function $f : B^n \rightarrow B^m$ is *compatible* with a Boolean relation $\mathcal{R}$, $f \propto \mathcal{R}$, if and only if $\forall x^i \in B^n$, $(x^i, z^i) \in \mathcal{R}$, where $z^i = f(x^i)$.

Definition 3.2 $\mathcal{R}_i$ **contains** $\mathcal{R}_j$, *denoted* $\mathcal{R}_j \subseteq \mathcal{R}_i$, *if and only if* $(x^i, z^i) \in \mathcal{R}_j \Rightarrow (x^i, z^i) \in \mathcal{R}_i$.

Definition 3.3 *A function* $f : B^n \rightarrow B^m$ *is* **compatible** *with* $\mathcal{M}$, $f \propto \mathcal{M}$, *if and only if there exists* $\mathcal{R}_i \in \mathcal{M}$ *such that* $f \propto \mathcal{R}_i$.

Definition 3.4 $\mathcal{R}$ *is an* **implicant relation** *of* $\mathcal{M}$ *if and only if* $f \propto \mathcal{R} \Rightarrow f \propto \mathcal{M}$.

Definition 3.5 $\mathcal{R}_i$ *is a* **prime relation** *of* $\mathcal{M}$ *if and only if* $\mathcal{R}_i$ *is an implicant relation of* $\mathcal{M}$ *and there exists no* $\mathcal{R}_j$ *such that* $\mathcal{R}_i \subseteq \mathcal{R}_j$ *and* $\mathcal{R}_j$ *is an implicant relation of* $\mathcal{M}$.

A multiple Boolean relation can be represented by a single Boolean function, where each minterm in the function represents a valid solution (single function) of the MBR. Since there are $2^{2^{m \cdot 2^n}}$ possible MBR's, a minimum of $\log(\log(2^{2^{m \cdot 2^n}})) = m \cdot 2^n$ Boolean variables is required for this single-function representation.

We introduce 2^n multivalued variables a_i, $i \in N = \{0, 1, \ldots, 2^n - 1\}$ (one for each input minterm $x^i \in B^n$). Each variable takes a value in $M = \{0, 1, \ldots, 2^m - 1\}$ (which represents the output minterms $z^j \in B^m$). Since there are $2^{2^{m \cdot 2^n}}$ possible MBR's, we require a minimum of $\log_{|M|}(\log(2^{2^{m \cdot 2^n}})) = 2^n$ variables to represent any arbitrary multiple Boolean relation. For the MBR in Figure 3 (call it MBR1), we introduce the variables a_0, a_1, a_2, and a_3, and each takes a value in $\{0, 1, 2, 3\}$ which represent the output minterms 00, 01, 10, and 11 respectively. Borrowing the terminology for multivalued variables and literals from [11], a_i is a variable taking a value from M, $J_i \subseteq M$, and ${a_i}^{J_i}$ is a literal of variable a_i representing the Boolean function

$${a_i}^{J_i} = \begin{cases} 0 \text{ if } a_i \notin J_i \\ 1 \text{ if } a_i \in J_i \end{cases}$$

For MBR1, the literal ${a_0}^{\{0,3\}}$ represents the two input minterm/output minterm pairs for the first input minterm in the first Boolean relation: (00, 00) and (00, 11). A *product term* p in the space $M^{|N|}$ is a Boolean product of literals: $p = \prod_{i=0}^{2^n-1} {a_i}^{J_i}$. A *minterm* in the space $M^{|N|}$ is a product term such that the cardinality of each J_i is 1. There is a one-to-one onto mapping between a product term p and a Boolean relation $\mathcal{R}$. p is mapped to $\mathcal{R}$ if and only if $(x^i, z^j) \in \mathcal{R} \Leftrightarrow j \in J_i$. p maps to $\mathcal{R}$ is denoted by $p = \varphi(\mathcal{R})$ or $\mathcal{R} = \varphi^{-1}(p)$. For MBR1, the product term ${a_0}^{\{0,3\}}\ {a_1}^{\{1\}}\ {a_2}^{\{1,3\}}\ {a_3}^{\{2,3\}}$ represents the first Boolean relation. Finally, $f_{mv} = \sum_{k=1}^{l} \varphi(\mathcal{R}_k)$ is the Boolean function of multivalued variables representing the multiple Boolean relation $\mathcal{M}$. For MBR1, $f_{mv} = {a_0}^{\{0,3\}}\ {a_1}^{\{1\}} {a_2}^{\{1,3\}}\ {a_3}^{\{2,3\}} + {a_0}^{\{1,2\}}\ {a_1}^{\{0,3\}} {a_2}^{\{0\}}\ {a_3}^{\{0,2\}}$.

The following lemma is directly from the definition of the mapping φ:

Lemma 3.1 *If a product term p is comprised of minterms $\{m_1, m_2, \ldots, m_q\}$, then the set of completely specified functions contained in the Boolean relation $\varphi^{-1}(p)$ is exactly $\{\varphi^{-1}(m_1), \varphi^{-1}(m_2), \ldots, \varphi^{-1}(m_q)\}$. Conversely, if a Boolean relation $\mathcal{R}$ is comprised of completely specified functions $\{F_1, F_2, \ldots, F_q\}$, then the set of minterms contained in the product term $\varphi(\mathcal{R})$ is exactly $\{\varphi(F_1), \varphi(F_2), \ldots, \varphi(F_q)\}$.*

Using the above Lemma, we can prove the following theorem which relates prime product terms to prime relations.

Theorem 3.2 *p is a prime implicant of f_{mv} if and only if $\varphi^{-1}(p)$ is a prime relation of the multiple Boolean relation $\mathcal{M} = \{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_l\}$.*

Proof. Suppose p is not a prime implicant. Thus there exists a product term $\tilde{p} \supset p$ ($\tilde{p}$ contains p [11]), such that $\tilde{p}$ is an implicant of f_{mv}. Consider the set of minterms $\{m_1, m_2, \ldots, m_q\}$ covered by the product term $\tilde{p}$. Since $\tilde{p}$ is an implicant of $f_{mv} = \sum_{k=1}^{l} \varphi(\mathcal{R}_k)$, for each $i \in \{1, 2, \ldots, q\}$, completely specified function $\varphi^{-1}(m_i)$ is an implicant relation of $\mathcal{M}$. Now consider the Boolean relation $\varphi^{-1}(\tilde{p})$. From Lemma 3.1, this Boolean relation is an implicant relation of $\mathcal{M}$. Since $\tilde{p} \supset p$, we have $\varphi^{-1}(\tilde{p})$ contains $\varphi^{-1}(p)$. Thus, $\varphi^{-1}(p)$ is not a prime relation of $\mathcal{M}$.

Conversely, suppose that $\varphi^{-1}(p)$ is not a prime relation of $\mathcal{M}$. Then there exists a Boolean relation $\tilde{\mathcal{R}}$ which contains $\varphi^{-1}(p)$ and is an implicant relation of $\mathcal{M}$, and all completely specified functions contained in $\tilde{\mathcal{R}}$ are implicant relations of $\mathcal{M}$. From Lemma 3.1, the product term $\varphi(\tilde{\mathcal{R}})$ is an implicant of f_{mv}. Also, since $\tilde{\mathcal{R}}$ contains $\varphi^{-1}(p)$, $\varphi(\tilde{\mathcal{R}}) \supset p$. Thus, p is not a prime implicant of f_{mv}. □

The function f_{mv} represents the multiple Boolean relation $\mathcal{M}$ using 2^n 2^m-valued variables. In the special case of a single output, f_{mv} reduces to a function of 2^n binary-valued variables. In the single Boolean relation case, f_{mv} reduces to a single product term.

As an alternative to the multivalued representation, a binary-valued representation can be obtained by encoding each 2^m-valued variable in m Boolean variables. In this representation, the product terms correspond to incompletely specified Boolean functions instead of Boolean relations. We introduce $m \cdot 2^n$ Boolean variables, $b_{i,j}$, where $i \in N$ and $j \in \{1, 2, \ldots, m\}$. Each variable represents a distinct (input minterm, output bit)-tuple. A literal of $b_{i,j}$ is either $b_{i,j}$ or $\bar{b}_{i,j}$, indicating that the j^{th} output is either 1 or 0 (respectively) for the i^{th} input minterm. For MBR1, the literal $\bar{b}_{0,1}$ will appear in the terms representing the first and third output minterm choices (00 and 01) for input minterm 00. A product term in the space $B^{m \cdot 2^n}$ is the Boolean product of these literals. There is a one-to-one onto mapping between a product term p and an incompletely specified Boolean function $\mathcal{F} : B^n \to B^m$. The product term p is mapped to $\mathcal{F}$, $p = \psi(\mathcal{F})$, if for all i, j, the value of the j^{th} output for i^{th} minterm, $\mathcal{F}^j(x^i)$, is 0 if $\bar{b}_{i,j}$ appears in p, 1 if $b_{i,j}$ appears in p, and 2 if neither appears in p. $f_{bv} = \sum_{k=1}^{s} \psi(\mathcal{F}_k)$ is the Boolean function of Boolean variables representing the multiple Boolean relation $\mathcal{M} = \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_s\}$. For MBR1, $f_{bv} = (\bar{b}_{0,1}\bar{b}_{0,2} + b_{0,1}b_{0,2}) \cdot \bar{b}_{1,1}b_{1,2} \cdot b_{2,2} \cdot b_{3,1} + (\bar{b}_{0,1}b_{0,2} + b_{0,1}\bar{b}_{0,2})(\bar{b}_{1,1}\bar{b}_{1,2} + b_{1,1}b_{1,2}) \cdot \bar{b}_{2,1}\bar{b}_{2,2} \cdot \bar{b}_{3,2}$.

Definition 3.6 *$\mathcal{F}_i$ is a* **prime incompletely specified function** *of M if and only if $\mathcal{F}_i$ is an incompletely specified function, is an implicant relation of $\mathcal{M}$, and there exists no other incompletely specified function $\mathcal{F}_j$ which is an implicant relation of $\mathcal{M}$ and $\mathcal{F}_i \subseteq \mathcal{F}_j$.*

Theorem 3.3 *p is a prime implicant of the Boolean function f_{bv} if and only if $\psi^{-1}(p)$ is a prime incompletely specified function of the multiple Boolean relation $\mathcal{M}$.*

Proof. Similar to the proof of Theorem 3.2. □

Several **canonical** forms for MBR's arise from the single Boolean function representation. In particular, either the sum of prime relations of f_{mv}, the sum of multiple output functions (minterm relations) of f_{mv}, the sum of prime incompletely specified functions of f_{bv}, or the sum of multiple output functions (minterm functions) of f_{bv} is a canonical form for an MBR. This follows directly from the relationship between an MBR, f_{mv}, and f_{bv} as given by Theorems 3.2 and 3.3.

The number of variables introduced can be unacceptably large for these representations; however, this is the minimum number of variables for any single function representation of an arbitrary MBR because of the size of the problem space ($2^{2^{m \cdot 2^n}}$). For some applications which operate on only small circuit fragments at a time, this number may be acceptable. For others, it may be more efficient to retain and manipulate the given representation (which may be multiple relations, multifunctions, or a single relation with constraints, where additional variables are introduced to represent the constraints; see Sections 3.5 and 4).

3.3 Solving Multiple Boolean Relations

There are several different representations for multiple Boolean relations; the representation used depends on the application (this will be illustrated in Section 4). Finding an optimal solution to an MBR is dependent on the particular representation. Below we list several representations and give methods for finding solutions for each. The first two representations are based on the correspondence between a multiple Boolean relation and a multivalued function f_{mv}, and between a multiple Boolean relation and a Boolean function f_{bv}. These representations for multiple Boolean relations can be derived easily from all applications that we have seen so far, especially in situations where the multiple Boolean relation manifests itself in the form of a single Boolean relation and constraints between input minterms (Section 3.5).

3.3.1 Boolean Function of Multivalued Variables

As a result of Theorem 3.2, prime relations of $\mathcal{M}$ are obtained from the primes of f_{mv}. Minimizing f_{mv} using ESPRESSO-MV [11] yields a minimum set of prime relations without excluding any solutions. An exact Boolean relations minimizer such as [3] or a heuristic minimizer such as GYOCRO [15] is called on each Boolean relation (each product term in f_{mv}) to find a minimal function satisfying the Boolean relation, and the minimum of these over all Boolean relations is taken as the best solution. By first minimizing the number of product terms in f_{mv} with ESPRESSO-MV, the number of calls to GYOCRO is minimized. The final result is either the exact minimum solution if an exact Boolean relations minimizer is called in the second step, or a heuristic minimal solution if a heuristic minimizer is used in the second step. All Boolean relations minimizers reported to date target two-level implementations.

3.3.2 Boolean Function of Boolean Variables

Similarly, the prime incompletely specified functions can be obtained from f_{bv} using ESPRESSO-MV. The optimum solution to each can be determined using standard two-level techniques such as ESPRESSO, and the minimum over all prime incompletely specified functions is taken as the best solution. Again, the final result is the exact minimum one if an exact minimizer is used in the second step, and a heuristic minimal one if a heuristic minimizer is used in the second step.

Note that this provides a new way of finding an exact minimum solution to a single Boolean relation: form the binary-valued function representation, find the prime incompletely specified functions, and find the minimum solution to each. For some problems, this method may be faster than [3] while providing a better solution than [15]; however, the number of prime incompletely specified functions may be prohibitively large. Our initial experimental results indicate that indeed, for many examples, the number of prime incompletely specified functions is very large for reasonably sized Boolean relations. For the MBR in Figure 3, the two Boolean relations (which constitute all prime relations) represent 16 functions and 6 prime incompletely specified functions. For both of the above representations, we may be able to remove some of the primes from consideration if we can determine a priori that they will not contain the optimal solution. These are areas of current investigation.

3.3.3 Synchronous Relations

Multiple Boolean relations arise naturally in the minimization of sequential circuits. Consider a sequential circuit implemented as a network of combinational gates and latches. The goal is to optimize a subcircuit using information from the surrounding logic. The flexibility in the input/output specification for the subcircuit can be completely specified in terms of finite sequential input/output traces. Damiani and De Micheli [7] have demonstrated how one can derive such traces (which they call synchronous recurrence equations). The depth of these traces is no greater than the number of latches in the surrounding logic.

The target logic for the subcircuit under consideration will have md outputs and nd inputs. The synchronous relation $S(x^1, x^2, \ldots, x^d, y^1, y^2, \ldots, y^d)$ is a Boolean relation in the Boolean space $B^{d(m+n)}$. A function $f : B^m \rightarrow B^n$ is said to be *compatible* with S if and only if for any d-vector of minterms $(x^1, x^2, \ldots, x^d) \in B^{nd}$, $S(x^1, x^2, \ldots, x^d, f(x^1), f(x^2), \ldots, f(x^d)) = 1$ [13]. The synchronous relation represents a set of functions which cannot be represented using an ordinary Boolean relation.

Synchronous relations can be converted to a Boolean function of Boolean variables as done in [7]. We will discuss that method in more detail in Section 4. If the number of latches in the surrounding logic is small, or if only a partial optimization is desired (only part of the solution space is considered), the depth of the synchronous relation can be small. In such situations, the representation of the multiple Boolean relation as a synchronous relation will be more compact, i.e., it will require fewer variables ($(m+n)d$ as opposed to $m \cdot 2^n$). Thus, it might be better to minimize the synchronous relation directly.

[13] have presented a *heuristic* method to optimize synchronous relations directly. They have represented and manipulated both the synchronous relation and the compatible function implicitly using binary-decision diagrams [5]. Their basic methodology is based on the ESPRESSO paradigm [2] which manipulates one product term at a time, maintaining compatibility with the desired specification and reducing cost greedily.

It is interesting to understand the relation between the multiple Boolean relations and the depth d required to represent them as a synchronous relation. It has been shown in [13] that any arbitrary set of Boolean functions (i.e., any arbitrary MBR) can be represented as a synchronous relation. Given a set of Boolean relations $\{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_l\}$, the synchronous relation $S(x^1, x^2, \ldots, x^l, y^1, y^2, \ldots, y^l)$ is 1 if and only if there exists an $\mathcal{R}_i$, $i \in \{1, 2, \ldots, l\}$, such that $\prod_{j=1}^{l} \mathcal{R}_i(x^j, y^j) = 1$. However the depth required to express the synchronous relation is dependent on the particular relations in the set. In particular, [13] have shown that that the depth can be as large as the cardinality of the number of Boolean relations needed to represent the MBR. If we represent an MBR as a Boolean function of multivalued variables in the space $M^{|N|}$ as discussed in Section 3.2, then the minimum number of Boolean relations needed to represent the MBR corresponds to the minimum number of cubes to represent a function in $M^{|N|}$. If $m = 1$, this number may be as large as 2^{n-1}; if $m > 1$, this number may be as large as 2^n.

3.4 Representing a Multiple Boolean Relation as an NFA

The behavior of a circuit is sometimes viewed as a non-deterministic finite automaton. For a combinational circuit with no additional constraints, i.e., one that can be represented by a Boolean relation, the automaton is simple: it contains two internal states, inputs $\mathbf{x}, \mathbf{z}$, and one binary output. The accepting state indicates

Single Boolean Relation with constraints		
x_1	x_2	$z_1 z_2$
0	0	00, 11
0	1	01, 10
1	0	01, 11
1	1	10, 11

(a)

Equivalent Multiple Boolean Relation			
x_1	x_2	$(z_1 z_2)^1$	$(z_1 z_2)^2$
0	0	11	00
0	1	01, 10	01, 10
1	0	01, 11	01, 11
1	1	10	11

(b)

Prime Incompletely Specified Functions					
x_1	x_2	$(z_1 z_2)^1$	$(z_1 z_2)^2$	$(z_1 z_2)^3$	$(z_1 z_2)^4$
0	0	00	00	11	11
0	1	01	10	01	10
1	0	21	21	21	21
1	1	11	11	10	10

(c)

Figure 4: Single Boolean Relation with Constraints = Multiple Boolean Relation

that only acceptable $\mathbf{x}, \mathbf{z}$ pairs have been seen, and the non-accepting state indicates that an unacceptable $\mathbf{x}, \mathbf{z}$ pair has been seen. One the automaton transitions to the non-accepting state, it stays there indefinitely. The output indicates the occurrence or absence of an unacceptable $\mathbf{x}, \mathbf{z}$ pair. The automaton therefore represents the permissible behaviors for the combinational circuit.

A multiple Boolean relation can be thought of as an automaton analogously. The automaton has inputs x, z and one binary output, but in this case there are many internal states. All but one state indicate that only permissible input/output pairs or combinations of input/output pairs have occurred. The single non-accepting state indicates that a non-permissible input/output pair or combination of input/output pairs has occurred. A path in the automaton keeps track of input/output pair combinations and thus insures that the input correlations in the specification are satisfied. The resulting automaton is in general a non-deterministic finite automaton, underscoring the expressibility of multiple Boolean relations for capturing sequential behaviors.

3.5 Single Boolean Relation with Constraints

There are several synthesis problems in which a multiple Boolean relation arises in the form of a single Boolean relation with constraints. The constraints are given between the input minterms of the relation. The task is to find an optimal solution to the Boolean relation that satisfies the constraints.

Example 3.1 *In Figure 4(a), a Boolean relation is given with constraints between the input minterms, e.g., if the output 00 is chosen for input minterm 00, then output 11 must be chose for input minterm 11. This is equivalent to the multiple Boolean relation in Figure 4(b).*

The constraints on the Boolean relation can be added to the functional representation for the relation. f_{mv} is constructed for the single Boolean relation (it is a single product term), clauses representing the constraints are constructed using the multivalued variables, and the clauses are intersected with f_{mv}. The resulting function represents all solutions to the original problem. In Example 3.1, $f_{mv} = a_0{}^{\{0,3\}} a_1{}^{\{1,2\}} a_2{}^{\{1,3\}} a_3{}^{\{2,3\}}$. The constraints are $(a_0{}^{\{1,2,3\}} + a_3{}^{\{3\}})$ (which represents the implication $a_0{}^{\{0\}} \Rightarrow a_3{}^{\{3\}}$) and $(a_0{}^{\{0,1,2\}} + a_3{}^{\{2\}})$. The final function is obtained by taking the product of f_{mv} and the constraints and is given by $a_0{}^{\{3\}} a_1{}^{\{1,2\}} a_2{}^{\{1,3\}} a_3{}^{\{2\}} + a_0{}^{\{0\}} a_1{}^{\{1,2\}} a_2{}^{\{1,3\}} a_3{}^{\{3\}}$. The same technique can be applied for the binary-valued representation. $f_{bv} = (\bar{b}_{0,1}\bar{b}_{0,2} + b_{0,1}b_{0,2})(\bar{b}_{1,1}b_{1,2} + b_{1,1}\bar{b}_{1,2})b_{2,2}b_{3,1}$, the constraints are $(b_{0,1} + b_{0,2} + b_{3,1}b_{3,2})$ and $(\bar{b}_{0,1} + \bar{b}_{0,2} + b_{3,1}\bar{b}_{3,2})$, and the final function is $\bar{b}_{0,1}\bar{b}_{0,2}\bar{b}_{1,1}b_{1,2}b_{2,2}b_{3,1}b_{3,2} + \bar{b}_{0,1}\bar{b}_{0,2}b_{1,1}\bar{b}_{1,2}b_{2,2}b_{3,1}b_{3,2} + b_{0,1}b_{0,2}\bar{b}_{1,1}b_{1,2}b_{2,2}b_{3,1}\bar{b}_{3,2} + b_{0,1}b_{0,2}b_{1,1}\bar{b}_{1,2}b_{2,2}b_{3,1}\bar{b}_{3,2}$. The corresponding prime incompletely specified functions are shown in Figure 4(c).

For some applications, it may be impractical to enumerate the constraints to build the additional clauses. Depending on the application, it may be possible to add the set of constraints in a more efficient way (perhaps implicitly, as sets) or to incorporate them directly in the representative function while it is being constructed. This latter technique is used in [8].

4 Applications

Multiple Boolean relations have arisen in a number of synthesis problems. One example was shown in Section 1. The multifunction in Example 1.1 was derived by analysis of don't care sequences in finite state machines. Several additional applications are described below. Each application can be expressed and solved using one of the single function forms for MBR's given in Section 3.2. However, depending on the origin of the Boolean relations and the additional constraints, more efficient techniques may be applied to specific problems.

4.1 FPGA Rectification

The goal of FPGA rectification as given in [8] is to change the output functions of the FPGA by modifying the functions of certain LUT's (look-up tables) within the FPGA, while maintaining the interconnections between LUT's.

FPGA Rectification Problem: *Given*

- *a network of LUT's, with inputs* $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$ *and outputs* $\mathbf{z} = \{z_1, z_2, \ldots, z_m\}$
- *a set of LUT's to be modified, with inputs* $\mathbf{u} = \{u_1, u_2, \ldots, u_s\}$ *and outputs* $\mathbf{v} = \{v_1, v_2, \ldots, v_t\}$*, where* $v_i = f_i(\mathbf{u})$ $(SUPP(v_i) \subseteq \mathbf{u})$ *and* $u_i = g_i(\mathbf{x})$
- *the current output functionality* $z_i = h_i(\mathbf{x}, \mathbf{v})$
- *the desired output functionality as a characteristic function* $SPEC(\mathbf{x}, \mathbf{z})$*, where* $SPEC = 1$ *for all pairs* (x^i, z^i) *in the desired terminal function*

determine new functions $v_{i_{new}}$ *such that the new output functionality is realized and the support of the* v_i*'s is unchanged:* $SUPP(v_i) = SUPP(v_{i_{new}})$.

Solution [8]:

1. *Form a Boolean relation* R *containing all* (u^i, v^j) *pairs, where* $u^i \in B^s$, $v^j \in B^t$*, that satisfy the specification* $SPEC$ *given that the rest of the network is unchanged.*
2. *Generate constraints that ensure that the support of each* v_i *is unchanged.*
3. *Find a solution to the constrained Boolean relation. The exact method assigns a new variable to each input minterm in the support* v_i*, and does so for each* v_i*. A solution is found by substituting the new variables into the expression for* R *and determining a satisfying assignment. The exact method always finds a solution if one exists. The heuristic method is a branch and bound technique which chooses a compatible function for* v_i *at each branch point* i *and updates the relation to reflect this choice. The number of backtracks is limited to avoid an exhaustive search.*

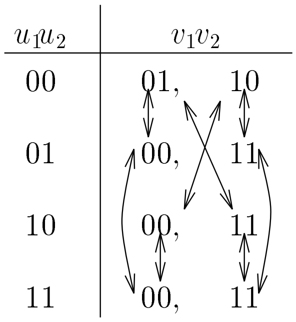


Figure 5: FPGA Rectification Example

The Boolean relation arises from the encapsulation of the environmental information used to determine functions for implementing the u_i's; the constraints on it arise from the support constraints in reimplementing the LUT functions of the subnetwork. The exact solution in [8] is similar to the Boolean function formulation from Sections 3.2 and 3.3.2, but the support of each v_i is limited by the number of inputs to the LUT so the number of new Boolean variables is $2^{|SUPP|} \cdot t$ (for outputs $v_1, v_2, \ldots v_t$, and where $|SUPP| = 5$ for Xilinx architectures). The multivalued function for the single Boolean relation with constraints in Figure 5 is given by

$$f_{mv} = {a_0}^{\{1,2\}} {a_1}^{\{0,3\}} {a_2}^{\{0,3\}} {a_3}^{\{0,3\}}$$

the constraints are

$$\begin{array}{c} ({a_0}^{\{0,2,3\}} + {a_1}^{\{0\}}) \\ ({a_0}^{\{0,2,3\}} + {a_2}^{\{3\}}) \\ ({a_0}^{\{0,1,3\}} + {a_1}^{\{3\}}) \\ ({a_0}^{\{0,1,3\}} + {a_2}^{\{0\}}) \\ ({a_1}^{\{1,2,3\}} + {a_3}^{\{0\}}) \\ ({a_1}^{\{0,1,2\}} + {a_3}^{\{3\}}) \\ ({a_2}^{\{1,2,3\}} + {a_3}^{\{0\}}) \\ ({a_2}^{\{0,1,2\}} + {a_3}^{\{3\}}) \end{array}$$

and the final function is 0 (there is no compatible function that satisfies the constraints). In general, the result is a function in which any satisfying assignment to the multivalued variables yields a valid solution. f_{bv} and its constraints are given by

$$f_{bv} = (\bar{b}_{0,1} b_{0,2} + b_{0,1} \bar{b}_{0,2})(\bar{b}_{1,1} \bar{b}_{1,2} + b_{1,1} b_{1,2})(\bar{b}_{2,1} \bar{b}_{2,2} + b_{2,1} b_{2,2})(\bar{b}_{3,1} \bar{b}_{3,2} + b_{3,1} b_{3,2})$$

Constraints:

$$\begin{array}{c} (b_{0,1} + \bar{b}_{0,2} + b_{2,1} b_{2,2}) \\ (b_{0,1} + \bar{b}_{0,2} + \bar{b}_{1,1} \bar{b}_{1,2}) \\ (\bar{b}_{0,1} + b_{0,2} + \bar{b}_{2,1} \bar{b}_{2,2}) \\ (\bar{b}_{0,1} + b_{0,2} + b_{1,1} b_{1,2}) \\ (b_{1,1} + b_{1,2} + \bar{b}_{3,1} \bar{b}_{3,2}) \\ (b_{2,1} + b_{2,2} + \bar{b}_{3,1} \bar{b}_{3,2}) \\ (\bar{b}_{1,1} + \bar{b}_{1,2} + b_{3,1} b_{3,2}) \\ (\bar{b}_{2,1} + \bar{b}_{2,2} + b_{3,1} b_{3,2}) \end{array}$$

Again, $f_{bv} \cdot constraints = 0$.

In [8], the constraints are merged directly into the Boolean relation by variable substitution. For this particular type of constraint, merging is more practical than enumerating constraints and intersecting them with the Boolean relation.

4.2 Synchronous Recurrence Equations

In [7], the notion of synchronous recurrence equations is used to optimize sequential subnetworks at the gate level. The goal is to reimplement a subnetwork in an optimal way, given information about the environment.

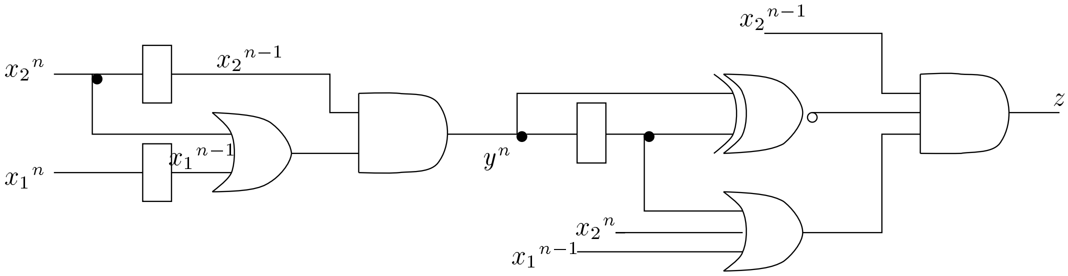


Figure 6: Synchronous Recurrence Equation Example

Optimization with Synchronous Recurrence Equations: *Given*

- *a network of gates and latches, with inputs* $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$ *and outputs* $\mathbf{z} = \{z_1, z_2, \ldots, z_m\}$; $z_i = g_i(\mathbf{x}^1, \mathbf{x}^2, \ldots \mathbf{x}^d)$, *where* d *is the circuit depth and* $\mathbf{x}^d$ *is the set of variables* $\mathbf{x}$ *in time frame* d
- *a set of internal variables* $\mathbf{y} = \{y_1, y_2, \ldots, y_p\}$, $y_i = h_i(\mathbf{x}^1, \mathbf{x}^2, \ldots \mathbf{x}^d)$, *to be reimplemented*

find a set of functions $y_i = f_i(\mathbf{x}^1, \mathbf{x}^2, \ldots \mathbf{x}^d)$ *that have a minimum two-level implementation and that replace* $h_i(\mathbf{x}^1, \mathbf{x}^2, \ldots \mathbf{x}^d)$ *without changing the output functions.*

Solution [7]:

1. *Form a Boolean relation between the inputs* $\mathbf{x}$ *and the intermediate variables* $\mathbf{y}$ *that represents all valid conditions in which the terminal behavior is unchanged.*
2. *From the Boolean relation, create a Boolean function* $\mathcal{K}$ *such that each satisfying assignment to* $\mathcal{K}$ *represents a function that is compatible with the Boolean relation and such that the function implementing* $y_i{}^j$ *is the same as the function implementing* $y_i{}^{j-1}$ *with all the input variables time-shifted by one, e.g., if* $y_1{}^n = f(\mathbf{x}^n, \mathbf{x}^{n-1})$ *then* $y_1{}^{n-1} = f(\mathbf{x}^{n-1}, \mathbf{x}^{n-2})$.

There is some flexibility in choosing Boolean relation $\mathcal{R}$ above. In particular, the depth d is chosen for both the input and output variables; the larger the depth value, the greater the amount of flexibility in implementation. See [7] for a detailed explanation.

Example 4.1 *In the example in Figure 6 [7], the goal is to reimplement* y^n *without changing the behavior at* z. *The terminal behavior is*

$$z = x_2{}^{n-2} x_2{}^{n-1} (x_1{}^{n-1} + x_2{}^n)$$

which is expressed in terms of y^n *as*

$$z = x_2{}^{n-1} (y^n \bar{\oplus} y^{n-1})(x_2{}^n + x_1{}^{n-1} + y^{n-1}$$

The resulting synchronous recurrence equation is

$$x_2{}^{n-2} x_2{}^{n-1} (x_1{}^{n-1} + x_2{}^n) \bar{\oplus} (y^n \bar{\oplus} y^{n-1})(x_2{}^n + x_1{}^{n-1} + y^{n-1}) = 1$$

The corresponding relation table is

$x_1{}^n$	$x_2{}^n$	$x_1{}^{n-1}$	$x_2{}^{n-1}$	$x_1{}^{n-2}$	$x_2{}^{n-2}$	$y^n y^{n-1}$
2	2	2	0	2	2	22
2	0	0	1	2	2	02, 20
2	2	1	1	2	0	01, 10
2	1	2	1	2	0	01, 10
2	2	1	1	2	1	00, 11
2	1	2	1	2	1	00, 11

The restriction is that the function used to implement y^n, *which for this example is chosen to be* $f(x_2{}^n, x_1{}^{n-1}, x_2{}^{n-1})$, *must be the same as that selected for* y^{n-1} *time-shifted by one:* $y^{n-1} = f(x_2{}^{n-1}, x_1{}^{n-2}, x_2{}^{n-2})$.

In [7], the function $\mathcal{K}$ is built by assigning a new variable to each input minterm for each output, and generating constraints directly from the Boolean relation. From Example 4.1, line 2 of the relation table, if f is chosen to be 1 at ${x_2}^n = 0$, ${x_1}^{n-1} = 0$, ${x_2}^{n-1} = 1$, then f must be chosen to be 0 at ${x_2}^{n-1} = 1$, ${x_1}^{n-2} = 0$, ${x_2}^{n-2} = 0$. [7] uses all such constraints to extract a function representing all valid solutions, and finds a two-level function of minimum cost function that satisfies the function. This formulation has an equivalent single multivalued function f_{mv} that can be used to find an optimum solution.

4.3 Don't Care Sequences

In [10], the problem of optimizing interacting finite state machines in considered. In the presence of a surrounding environment of interacting finite state machines, the observability and the controllability of a finite state machine is reduced. This sequential flexibility is referred to as sequential controllability and observability don't cares, quite similar to the case for combinational circuits [12].

As [10] have shown, the sequential don't cares for the interacting finite state machine environment can be represented as finite sequences of input-output pairs. This can be naturally cast into the synchronous recurrence equations that we discussed in the last section. We can then use the techniques we discussed there to solve the recurrence equation.

As we saw in the example in Section 1, sequential don't cares cannot be completely specified by ordinary Boolean relations. The specification in Figure 1 corresponds to a multiple Boolean relation. We can transform the specification into a multivalued function, f_{mv}, or a Boolean function, f_{bv}, as we described in Section 3.2. Then we can use the techniques we developed in Sections 3.3.1 and 3.3.2 to find an optimum or optimal solution.

5 Final Notes

It is interesting place the multiple Boolean relation model of representing flexibility in the context of current trends in logic synthesis. A general trend has developed in which individual components of larger systems are considered. The synthesis process is to extract the component of interest, derive information from the environment about the flexibility afforded the component, and choose an optimal function to implement for the component. This has been done exactly for combinational circuits using Boolean relations and for sequential circuits using non-deterministic finite state machines [16]. While the state machines derived in [16] contain all permissible behaviors (assuming a particular initial state) in the form of a single symbolic state machine, a multiple Boolean relation could be derived in the same context that would contain all permissible output behaviors for a given encoding in the form of logic functions. (An expensive but interesting modification would be the derivation of a single multiple Boolean relation representing all permissible output behaviors under all possible encodings.) Minimization of state machines is done by state minimization, where the effects on the final implementation are not clear. Minimization of the logic-level functions stored as an MBR can provide a more direct measure of the size of the final implementation. Thus, MBRs can be a valuable tool for representing a rich set of behaviors while maintaining the logic-level view.

Finally, we note that just as combinational logic functions require MBRs in the most general representation, sequential logic functions require multiple symbolic state machines in their most general representation. In [16], a single symbolic machine sufficed to represent all permissible behaviors since only a single initial state was considered. If the initial state is not known, multiple symbolic machines may be required to represent all permissible behaviors.

6 Conclusions

We have shown that multiple Boolean relations can express any set of multiple output functions for a network. The given set of functions may not be expressible by a single incompletely specified function or a single Boolean relation. We have given methods for finding both the exact minimum and a minimal solution based on a two-level implementation. These methods rely on the relationship between a multiple Boolean relation and a single multivalued or binary-valued function. Multiple Boolean relations have arisen in several synthesis problems, and we have shown how these different applications can be handled in a uniform way.

The notion of multiple Boolean relations inspires several directions for future research. It would be interesting to extract the most compact form of a multiple Boolean relation (i.e., a Boolean relation with constraints) given an arbitrary collection of functions. This capability would imply the ability to extract underlying structure from a seemingly random collection of functions. Multiple Boolean relations may be useful in choosing an optimal implementation of a gate-level sequential circuit, where previous optimization techniques were applied at the symbolic state machine level.

7 Acknowledgements

Discussions with both Richard Rudell and Fabio Somenzi contributed to the inception of these ideas and are gratefully acknowledged. This research was supported by NSF/DARPA under grant number 8719546, and NSF under grant number EMC-8419744.

References

[1] K.A. Bartlett, R.K. Brayton, G.D. Hachtel, R.M. Jacoby, C.R. Morrison, R.L. Rudell, A.L. Sangiovanni-Vincentelli, and A.R. Wang. Multi-level Logic Minimization Using Implicit Don't Cares. *IEEE Transactions on Computer-Aided Design*, CAD-7(6):723–740, June 1988.

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

[3] R.K. Brayton and F. Somenzi. An Exact Minimizer for Boolean Relations. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 316–319, November 1989.

[4] R.K. Brayton and F. Somenzi. Boolean Relations and the Incomplete Specification of Logic Networks. In *Proceedings of the International Conference on VLSI'89*, Munich, August 1989.

[5] R.E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[6] M. Damiani and G. De Micheli. Don't Care Set Specifications in Combinational and Synchronous Logic Circuits. Technical Report CSL-TR-92-531, Stanford University, Computer Systems Laboratory, Stanford, CA 94305-4055, July 1992.

[7] M. Damiani and G. De Micheli. Recurrence Equations and the Optimization of Synchronous Logic Circuits. In *Proceedings of the 29^{th} Design Automation Conference*, pages 556–561, June 1992.

[8] Y. Kukimoto and M. Fujita. Rectification Method for Lookup-Table Type FPGA's. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 54–61, November 1992.

[9] S. Muroga, Y. Kambayashi, H. C. Lai, and J. N. Culliney. The Transduction Method - Design of Logic Networks Based on Permissible Functions. *IEEE Transactions on Computers*, C-38(10):1404–1424, October 1989.

[10] J.-K. Rho, G. Hachtel, and F. Somenzi. Don't Care Sequences and the Optimization of Interacting Finite State Machines. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 418–421, November 1991.

[11] R. Rudell and A. Sangiovanni-Vincentelli. Multiple-valued Minimization for PLA Optimization. *IEEE Transactions on Computer-Aided Design*, CAD-6(5):727–750, September 1987.

[12] Hamid Savoj. *Don't Cares in Multi-Level Network Optimization.* PhD thesis, University of California Berkeley, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, May 1992.

[13] V. Singhal, Y. Watanabe, and R.K. Brayton. Heuristic Minimization of Synchronous Relations. In *Proceedings of the International Conference on Computer Design*, pages 428–433, October 1993.

[14] H.Y. Wang and R.K. Brayton. Permissible Observability Relations in Interacting Finite State Machines. In *Proceedings of the International Workshop on Logic Synthesis*, Tahoe City, California, May 1993.

[15] Y. Watanabe and R.K. Brayton. Heuristic Minimization of Multiple-Valued Relations. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 126–129, November 1991.

[16] Yosinori Watanabe. *Logic Optimization of Interacting Components in Synchronous Digital Systems.* PhD thesis, University of California Berkeley, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, April 1994.