A FULLY IMPLICIT QUINE-MCCLUSKEY PROCEDURE USING BDD'S

by

Gitanjali M. Swamy, Patrick McGeer, and Robert K. Brayton

Memorandum No. UCB/ERL M92/127

10 November 1992

A FULLY IMPLICIT QUINE-MCCLUSKEY PROCEDURE USING BDD'S

by

Gitanjali M. Swamy, Patrick McGeer, and Robert K. Brayton

Memorandum No. UCB/ERL M92/127

10 November 1992

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A Fully Implicit Quine-McCluskey Procedure using BDD's

Gitanjali M. Swamy, Patrick McGeer, Robert K. Brayton

Abstract

We present an exact method for minimizing logic functions using BDD's to represent our functions. This approach differs from the classical approach in that it exploits the properties of the BDD data structure and the properties of a new extended space that we define, in order to implicitly compute the Primes, Minterms and Covering table for the Quine-McCluskey procedure. In this method the function is mapped to an extended space which endows it with special properties that may be exploited to compute the function Primes and Minterms. The next step consists of conceptually creating a covering table whose rows represent the minterms and whose columns represent the primes. We formulate conditions for row and column dominance and remove dominated rows and columns iteratively until no more reduction is possible. The final step consists of finding a minimum column cover for the remaining cyclic core of the problem. All functions are implemented using implicit BDD operations.

Table of Contents

Introduction

The objective of logic minimization is to create a representation for a given logic function which requires a minimal number of logic devices for its implementation. This problem is CoNP-hard in nature and hence any exact algorithm to solve it is thought to be of exponential complexity[8]. The Quine-McCluskey procedure[7] is one such exact algorithm.

Before we begin our method, we need to the discuss the Quine-McCluskey procedure. The basic Quine-McCluskey tabular minimization procedure is as follows:

1. Find all the prime implicants of the function.

2. Construct the prime-implicant table

3. Determine the dominated rows of the table and delete them. Next, determine dominated columns of the table and delete them.

4. Repeat step 3 until no more reduction is possible. At this point we are left with a *cyclic core*.

5. Find the minimum column cover for the remaining problem.

Though this algorithm may be used effectively for small examples, it often fails in its explicit form for larger examples. For example the espresso-exact algorithm[6] fails in the case of problems like the *mish*[6] example with a large number of primes even when the actual cover is quite small in comparison.

Recent work at Bull Research [1], [3], gives us a new implicit approach to this problem. In those papers, O. Coudert & J.C. Madre [1,3] have developed a new method of representing primes of Boolean functions. Through their techniques they have been able to arrive at a collection of the primes of the largest and most difficult of the public benchmark functions. We extend the techniques of [1],[3] to exact minimization of boolean functions. The method we propose here relies on the following statement: *Any precise set can be phrased as a propositional sentence over the appropriate boolean space*. Thus the primes, minterms, as well as the Quine-McCluskey dominators may be formulated as propositional sentences.

Briefly we represent the primes and the minterms required and the covering table implicitly, and express step (3) of the Quine-McCluskey procedure as operations over this implicit representation. We then arrive at the cyclic core of the combinatorial problem in an implicit representation and derive the actual primes and minterms implicitly for this cyclic core; since the primes and minterms of the cyclic core are just a fraction of the total primes and minterms we hope to solve those problems which have not yet been solved by explicit methods[6].

Recapping from Bryants paper[4], a BDD or a binary decision diagram is a tree data structure (defined in the next section). Operations on this type of data structures are a function of the number of nodes in the tree, whereas the number of

terms it represent are dependent on the number of paths down the tree. It is possible to perform logic operations like AND, OR, XOR, NOR etc. as well as logical quantification by performing the basic BDD operations as given by Bryant's paper[4]. The BDD[4] data structure lends itself very well to implicit operations. This is because operations on BDD's are dependent on the number of nodes in the BDD, however the terms it represents are determined by the number of paths in the BDD and the BDD representation for complex combinatorial functions turn out to be surprisingly compact in the number of nodes involved. The main disadvantage of using BDD's is that given a bad ordering for the input variables, it is highly likely that the size of the BDD becomes inordinately large. We have explored this problem extensively and arrived at what we think is a good ordering for the input and output variables for a combinatorial function.

Given any proposition over a finite boolean space, one can find the solution set to the proposition by a series of BDD operations on the proposition and in fact there is a direct correlation between operations in the proposition and BDD operations. These insights were given by Coudert & Madre [1],[3]. While the two insights are not extremely remarkable, what is remarkable is that the BDD representations of formidable propositions are often small, making this an attribute for the solution of boolean problems. It is easy to see that it is possible to write the Quine-McCluskey procedure as a sequence of BDD operations.

In the case of BDD operations the major bottlenecks are quantification (defined in the next section). Thus it is essential for the success of this approach to reduce the use of quantifiers as much as possible. Thus the attempt will be to reduce the use of quantifiers at each stage.

The rest of the paper is devoted to the translation of the Quine-McCluskey algorithm to a series of formulae over the appropriate boolean space and to their computations using implicit BDD techniques. To make the paper more concise and readable, we have relegated all proofs to the appendix for reference.

Definitions

Boolean space: A boolean space B^n is a space where variables may only take the values 0 and 1.

Logic Function: Let $X_1, X_2, X_3, \ldots, X_n$ be variables on a Boolean space B^n . A completely specified logic function is a mapping from B^n to B. An incompletely specified function consists of 3 parts; f, d and r. f is a completely specified function which is called the onset and consists of the points where the function is 1, d is the don't care function and consists of all the points where the value of the function may be both 0 or 1 and r is the offset and consists of all the points where the function will take the value 0. f, d and r together form the incompletely specified function.

Literal: A *literal* is an ordered pair of the form *(variable, value)*. By convention the pair $(X_i,0)$ is written as $\overline{X}_i$ and the pair $(X_i,1)$ is written as X_i. If the variable takes on the value 0 then the literal X_i is said to be 1 and $\overline{X}_i$ is said to be 0. If the variable takes on the value 0 then the literal X_i is said to be 0 and the literal $\overline{X}_i$ is said to be 1

Vertex: A vertex is a single point in the subspace corresponding to the function input and output variables.

Minterm: A *minterm* of an incompletely specified function *(f,d,r)*, is a vertex of the space which is in the onset of *f*.

Monotonically decreasing function: A monotonically decreasing function is a function such that changing any (boolean) variable from value 1 to value 0 causes the function value, if it changes, to go from value 0 to value 1.

Cube: A cube is a subspace $C_1 \times C_2 \times \ldots \times C_n$ of B^n where C_i is a subset of $\{0,1\}$. It can also be written as a product of literals. A vertex $(v_1, v_2, \ldots, v_n)$ is contained in a cube $C_1 \times C_2 \times \ldots \times C_n$ iff $v_i \in C_i$ for all i. For convenience a cube written as a product of literals with the connection that neither literal for a variable are present if C_i is *{0,1}*, e.g. the cube $\{0, 1\} \times \{0\} \times \{1\}$ over B^3 is written as $\bar{X}_2 X_3$. A cube D $D_1 \times D_2 \times \ldots \times D_n$ is said to be contained in a cube C $C_1 \times C_2 \times \ldots \times C_n$ if $D_i \subseteq C_i$ for all i. The size of a cube $C_1 \times C_2 \times \ldots \times C_n$ is given by $|C_1| \times |C_2| \times \ldots \times |C_n|$. A vertex of the space can also be defined as a cube of size 1.

Null Cube: A null cube is a cube of size 0. The null cube space is the subspace consisting of all the null cubes of a given space. As a result of the way cubes have been defined, it is observed that there are $(4^n - 3^n)$ null cubes.

Implicant: An implicant of an incompletely specified function *(f,d,r)* is a cube *C*, such that, no vertex contained in this cube belongs to the offset *r*. The definition for implicant containment is identical to cube containment.

Prime Implicant: A *prime implicant* is an implicant which is not contained in any other implicant of the function.

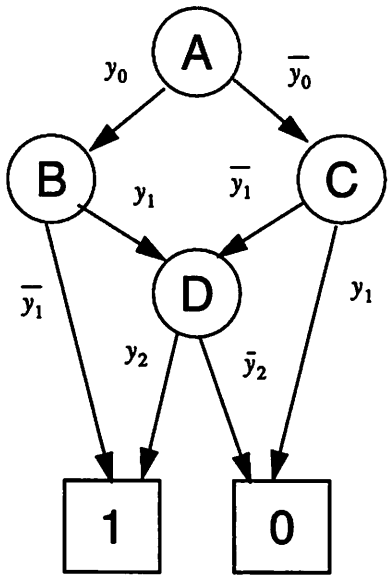
Cofactors: The cofactor of a function *f* with respect to a literal (x, i), i.e. variable *x* in the i^{th} value, is the function obtained by evaluating the function *f* on the plane *x=i*. Conventionally the cofactor of *f* with respect to (x,1) is written as f_x and the cofactor of f with respect to (x,0) is written as $f_{\bar{x}}$.

Shannon Expansion: A function may be written in terms of its cofactors with respect to a variable *x*. This is written as $f = \bar{x} f_{\bar{x}} + x f_x$. This leads to the concept of a Shannon tree. If we recursively compute the value of the function using the above expression and computing the cofactor with respect to a new variable in the support of the function at each stage of the recursion and terminate when we reach the condition that the remaining function is either 0 or 1 in value, we get an expanded function. If we represent each level in this recursion by a unique node with the left and right branches of this node being representing the two cofactors, then the resulting structure becomes a tree and is called a Shannon tree. Each level of recursion represents a new level of the tree.

Binary Decision Diagram: [4] *The binary decision diagram for a function is the folded form of the Shannon tree for the function. A function graph is a rooted directed graph with a vertex set V containing two types of vertices; a non terminal vertex v has as attributes an* $index(v) \in \{0, 1 \ldots n-1\}$ *and two children low(v) and high(v) belonging to V. A terminal vertex has as attribute a value* $value(v) \in \{0, 1\}$. *A function graph is reduced if it contains no vertex v with* $low(v) = high(v)$ *and no distinct vertices v and v′ such that the subgraphs rooted at them are isomorphic.*A BDD,

also called a ROBDD is then defined as a reduced function graph. This tree has labelled internal nodes corresponding to the variable with respect to whom the function is expanded at the given level.

FIGURE 1. BDD for function F



This representation is a canonical form. The root node corresponds to the variable with respect to whom we cofactors and the left branch corresponds to BDD for the cofactor of the function with respect to $\bar{x}$ and the right branch corresponds to the BDD for the cofactor of the function with respect to x The example in Figure 1 illustrates the BDD for the function $F = y_0 \cdot \bar{y}_1 + y_0 \cdot y_2 + \bar{y}_1 \cdot y_2$

Covering Table: The covering table M_f of function f, represents the problem of finding the smallest prime irredundant cover of the function. The rows of this table correspond to the minterms and the columns of the table correspond to the primes. $M_f(i, j) = 1$ if minterm i is contained in a prime j and 0 otherwise. A *column cover* of this table is a set of columns of this table such that each row has a '1' entry in at least one of the columns of the cover. A column cover for this table corresponds to a prime cover for the function it represents. We are looking for a *minimum prime cover* for our function, this corresponds to minimum column cover for this table.

Row Dominance: A row (minterm) is said to dominate another row (minterm) iff any cover which covers the first row, automatically covers the second row. This occurs when all the primes containing the first minterm (row) also contain the second minterm (row) and there is a prime containing the second minterm which does not contain the first.

Column Dominance: A column (prime) is said to dominate another column (prime) iff any cover which contains the first column automatically contains the second column. This occurs when all the minterms contained in the second prime (column) are also contained in the first prime (column) and there exists a minterm (row entry =1) which is contained in the first prime (column) but not in the second prime (column).

Quantification: There are two different *quantifiers*, $\exists x$ and $\forall x$. The first quantifier is the existential quantifier. If there exists a vertex x such that some condition $f(x)$ is 1, this is shown as $\exists x f(x) = 1$. The second quantifier is called the "for all" quantifier. If for all variables x some condition $f(x)$ is 1, this is written as $\forall x f(x) = 1$. The relation between these operators is expressed as $\forall x F(x) \Leftrightarrow \overline{\exists x (\bar{F}(x))}$ and $\exists x F(x) \Leftrightarrow \overline{\forall x \bar{F}(x)}$. Thus each quantifier may be written in terms of the other quantifier.

The Smoothing operator: The smoothing operator S_x is given by $S_x(f) = f_x + f_{\bar{x}}$. The smoothing operator distributes over variable sets, thus $S_{xy} = S_x \cdot S_y = S_y \cdot S_x = S_{yx}$. In addition it can be shown that [2] $\exists x F(x, y) = S_{x_1 x_2 \ldots x_n} F(x, y)$

The Consensus operator: The consensus operator C_x is given by $C_x(f) = f_{\bar{x}} \cdot f_x$. This operator too, distributes over a set of variables, i.e. $C_{xy} = C_x \cdot C_y = C_y \cdot C_x = C_{yx}$. In addition we have the relation that $\forall x F(x, y) = C_{x_1 x_2 \ldots x_n} F(x, y)$

The Quine-McCluskey procedure

The Quine-McCluskey tabular minimization procedure follows the following steps.

1. Find all the prime implicants of the function.

2. Construct the covering table. The rows in the covering table correspond to minterms of the onset of the function, the columns of the covering table correspond to the primes computed in step 1. An entry in the table is 1 if the corresponding row minterm is contained in the its column prime, otherwise the entry is 0. Our problem is to find a *minimum column cover* for all the rows. The essential prime are those columns one of whose row entries is not contained in any other column.

3. Determine the dominated rows and remove them from the table, next determine the dominated columns and remove them from the covering table

4. Repeat the process until no reduction is possible. When no more reduction is possible the remaining problem is called the *cyclic core*. There are no dominated rows or dominated columns in the cyclic core.

5. At this point find a *minimum column cover* for the cyclic core.

The main bottlenecks here are the problems of prime explosion. The number of primes for n input variables can potentially be as large as $3^n/n$ and hence for larger examples the number of primes become too large to enumerate, even when the size of the cyclic core is small. For e.g. the Espresso-exact algorithm fails on the example circuit "mish" [6] which has 10^{14} primes but a cyclic core with just 82 primes.

The extended space and the implicant characteristic

Our goal is to represent all the cubes over B^n as terms in some space and the Quine-McCluskey algorithm as a series of propositional formulae on that space.

We must note that functions work over *points* but cubes are *collections of points*. The key objective is to map cubes onto a space in which we can perform minimization algorithms by operations on functions which implicitly represent the set of implicants, minterms, primes etc.

Consider an arbitrary cube $C = C_1 \times C_2 \times \ldots \times C_n$. Each C_j is an arbitrary subset of $\{0,1\}$. Since there are 4 subsets, it follows there are 4^n vertices of any extended space; i.e. the extended space of B^n is B^{2n}. This differs from the conventional practise of assuming 3^n products in the original space. The difference comes from the way we have defined a cube which leads to $(4^n - 3^n)$ null cubes.

Before we begin let us clarify the notation used. We will represent the variables in the original space as Y_i and the variables in the new extended space as X_{ij}, z_{ij}, u_{ij}, where $1 \le i \le n$ and $j \in \{0, 1\}$

Assuming all variables are binary valued in the original space. We choose the following extended space, using $2n$ variables:

Definition $\Theta(C)$: Consider any cube $C_1 \times C_2 \times \ldots \times C_n$ in the original space. *The corresponding vertex in the higher order space is given by* $\Theta(C) = (x_{10} \cdot x_{11} \cdot \ldots \cdot x_{n1})$, *where* $x_{ij} = 0$ *if* $j \in C_i$ *and* $x_{ij} = 1$ *if* $j \notin C_i$, e.g. The cube $\bar{Y}_2 \cdot Y_3$ over B^3 is represented as the vertex (1,1,1,0,0,1) = $X_{10} \cdot X_{11} \cdot X_{20} \cdot \bar{X}_{21} \cdot \bar{X}_{30} \cdot X_{31}$ over B^6.

Theorem 1.1: The mapping Θ is 1-1 for all non null cubes.

Proof: Assume a contradiction.

Let us assume there exists a cube $C_1 \times C_2 \times \ldots \times C_n$ which maps to at least 2 points. Let us call these 2 points a and b. Consider some variable X_{ij} in which a and b differ.

$X_{ij} = 0$ for a.
$X_{ij} = 1$ for b.

$$(X_{ij} = 0) \Leftrightarrow (j \notin C_i) \tag{1}$$

$$(X_{ij} = 1) \Leftrightarrow (j \in C_i) \tag{2}$$

For any cube $C_1 \times C_2 \times \ldots \times C_n$ each C_i is a unique set $C_i \subseteq \{0, 1\}$.

Hence Equation1 and Equation 2 lead to a contradiction.

This implies that the mapping is unique for all non-null points.

This space is also known as the *positional notation* and is commonly used for representing multi-valued functions. To understand this refer to Figure. 2 of the extended space, also called the *coded cube space(ccs)*. Every non null cube has a unique representation in this space. The figure shows the mapping of points in a 2-dimensional space to the 4-dimensional extended space. In the figure shown the minterm $\overline{Y}_0\overline{Y}_1$ translates to the point $X_{00}\overline{X}_{01}X_{10}\overline{X}_{11}$ and the cube Y_1 translates to the point $X_{00}X_{01}\overline{X}_{10}X_{11}$ in the extended space.

The variables z and u are used to provide temporary intermediate storage for computations.

Definition $\chi^F(x)$ We define the *characteristic of function* F in the extended space χ^F as a mapping from B^{2n} to B such that

$$\chi^F(x) = 1 \Leftrightarrow x = \{\Theta(C) | (C \in cube(F))\} \tag{3}$$

The characteristic function of F in the extended space is also called the *implicant characteristic.*

Theorem: 1.2: The characteristic function of F in the extended space satisfies the following property.

$$\chi^{FG} = \chi^F \cdot \chi^G \tag{4}$$

Proof:

$\chi^F(x) = 1$ iff $x = \{\Theta(C) | (C \in cube(F))\}$.

$(\chi^{FG}(x) = 1) \Leftrightarrow$

$x = \{\Theta(C) | (C \in cube(F)) and (C \in cube(G))\}$

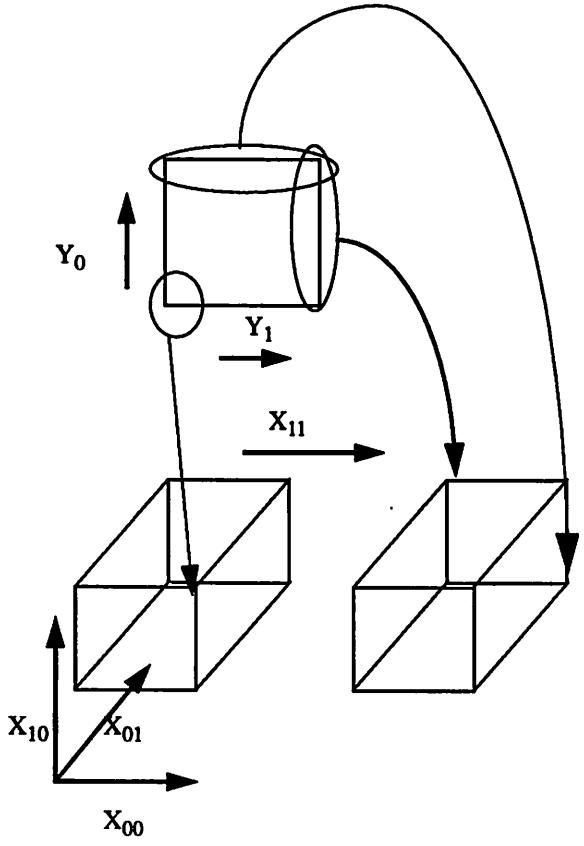
(5)

$$\Leftrightarrow (x = \{\Theta(C) | (C \in cube(G))\}) \text{ and } x = \{\Theta(C) | (C \in cube(F))\} \tag{6}$$

$$\Leftrightarrow (x \in \chi^F) \bullet and \bullet (x \in \chi^G) \tag{7}$$

This implies that $\chi^{FG} = \chi^F \cdot \chi^G$

FIGURE 2. The Coded Cube Space



Theorem: 1.3: If every prime of *F+G* is a prime of *F* or a prime of *G* then the characteristic functions of in the extended space satisfies the following property.

$$\chi^{(F+G)} = \chi^F + \chi^G \tag{8}$$

Proof:

$$\chi^F(x) = 1 \text{ iff } x = \{\Theta(C) | (C \in cube(F))\}. \quad (9)$$

$(\chi^{F+G}(x) = 1) \Leftrightarrow$

$$x = \{\Theta(C) | (C \in cube(F)) or (C \in cube(G))\} \quad (10)$$

$$\Leftrightarrow (x = \{\Theta(C) | (C \in cube(G))\})$$
or
$$x = \{\Theta(C) | (C \in cube(F))\} \quad (11)$$

Equation 11 comes as a results of our statement that every prime of F+G is either a primes of F or a prime of G, if some cube is formed by combining cubes that are individually either in F or G, then it follows that there must be a larger prime covering this cube which is not contained in either F or G and this contradicts our assumption.

$$\Leftrightarrow (x \in \chi^F) \bullet or \bullet (x \in \chi^G) \quad (12)$$

This implies that $\chi^{(F+G)} = \chi^F + \chi^G$

Key Theorem of implicant characteristic
Theorem 1.4: The characteristic of a function F is given by

$$\chi^F = (X_{i0} \Rightarrow \chi^{F_{\bar{Y}_i}})(X_{i1} \Rightarrow \chi^{F_{Y_i}}) \quad (13)$$

Proof:

The above is equivalent to

$$\chi^F = (\bar{X}_{i0} + \chi^{F_{\bar{Y}_i}})(\bar{X}_{i1} + \chi^{F_{Y_i}}) \quad (14)$$

We will prove this by induction. Let us consider the base case; i.e.

$$\chi^0 = 0 \quad (15)$$

$$\chi^1 = 1 \quad (16)$$

Consider the case when F is a function of a single variable. The possibilities for the function F are the following.

$F = Y_i$ this gives $\chi^F = \bar{X}_{i0}X_{i1} = \bar{X}_{i0}X_{i1} + \bar{X}_{i0}\bar{X}_{i1} = \bar{X}_{i0}$ since the term $\chi^F = \bar{X}_{i0}\bar{X}_{i1}$ is a null cube. (EQ 7,8) give the same answer

$F = \bar{Y}_i$ this gives $\chi^F = X_{i0}\bar{X}_{i1} = X_{i0}\bar{X}_{i1} + \bar{X}_{i0}\bar{X}_{i1} = \bar{X}_{i1}$ since the term $\chi^F = \bar{X}_{i0}\bar{X}_{i1}$ is a null cube.(EQ 7,8) give the same answer.

F=0 and F=1 reduce to the base case.

The function F can be written as

$$F = Y_i \cdot F_{Y_i} + \bar{Y}_i \cdot F_{\bar{Y}_i} + F_{Y_i} \cdot F_{\bar{Y}_i} \tag{17}$$

which factors as

$$F = (\bar{Y}_i + F_{Y_i}) \cdot (Y_i + F_{\bar{Y}_i}) \tag{18}$$

hence translating the function to the extended space and using theorem 1.2 we have

$$\Rightarrow \chi^F = \chi^{(Y_i + F_{\bar{Y}_i})} \cdot \chi^{(\bar{Y}_i + F_{Y_i})} \tag{19}$$

Using the fact that (Y_i *and* $F_{\bar{Y}_i}$) and ($\bar{Y}_i$ *and* F_{Y_i}) have no cubes in common (excluding the null cube) and theorem 1.3 we have the following.

$$\Rightarrow \chi^F = (\chi^{Y_i} + \chi^{F_{\bar{Y}_i}}) \cdot (\chi^{\bar{Y}_i} + \chi^{F_{Y_i}}) \tag{20}$$

$$\Rightarrow \chi^F = (\bar{X}_{i0} + \chi^{F_{\bar{Y}_i}})\,(\bar{X}_{i1} + \chi^{F_{Y_i}}) \tag{21}$$

Hence proved

Technical Functions over the CCS

The Null Cube space ($\phi(x)$)

Theorem 1.5: The null cube set is given by

$$\phi(X) = \sum_i \bar{X}_{i0} \cdot \bar{X}_{i1} \tag{22}$$

Proof:

$$X_{i0} = 0 \Rightarrow 0 \notin C_i \tag{23}$$

$$X_{i1} = 0 \Rightarrow 1 \notin C_i \tag{24}$$

$$(0 \notin C_i)\, and\, (1 \notin C_i) \Rightarrow C_i = \phi \tag{25}$$

$$C_i = \phi \Rightarrow C = \phi \tag{26}$$

Adding the null cube to an expression does not add any vertices to the function as the null cubes contain no vertices. However adding the null cube to any function in this extended space makes the function monotonically decreasing in this space (Theorem 1.6) and hence gives it special properties which will be exploited in order to calculate the primes.

Theorem 1.6: For any function F, $\chi^F + \phi$ form a monotonically decreasing function in the extended space.

Proof:

Consider any cube of the form $\bar{X}_{i0} \cdot X_{i1} \cdot A$

$\bar{X}_{i0} \cdot \bar{X}_{i1}$ is a member of the null space. Thus $\bar{X}_{i0} \cdot X_{i1} \cdot A = \bar{X}_{i0} \cdot X_{i1} \cdot A + \bar{X}_{i0} \cdot \bar{X}_{i1}$ but we know $\bar{X}_{i0} \cdot \bar{X}_{i1} A \subseteq \bar{X}_{i0} \cdot \bar{X}_{i1}$. Thus we have

$$\bar{X}_{i0} \cdot X_{i1} \cdot A = \bar{X}_{i0} \cdot X_{i1} \cdot A + \bar{X}_{i0} \cdot \bar{X}_{i1} A + \bar{X}_{i0} \cdot \bar{X}_{i1}$$

Which is equal to $\bar{X}_{i0} A + \bar{X}_{i0} \cdot \bar{X}_{i1}$

- Thus $\bar{X}_{i0} A$ is a cover for the cube.

It can be similarly argued for any cube of the form $X_{i0} \cdot \bar{X}_{i1} A$

- $\bar{X}_{i1} A$ is a cover for this cube

Consider any cube of the form

$$X_{i0} \cdot X_{i1} \cdot A \tag{27}$$

Existence of this expression implies that X_i occurs in both its complemented and its non-complemented form in the original space. This implies that in the extended space both the cubes.

$$\bar{X}_{i0} \cdot X_{i1} \cdot A \tag{28}$$

$$X_{i0} \cdot \bar{X}_{i1} \cdot A \tag{29}$$

occur. Hence if the null cube is added to these expressions both the X_{i0} and the X_{i1} dependences disappear.

- Thus A is a cover for this cube.

We can repeat this sequence of operations on the cube A to remove all cubes which have a variable present in the non-complemented form and replace them with cubes in the complemented form alone.

It follows as a result, that each cube may be replaced by another cube which depends only on the complemented literals.

Hence all cubes of the function may be replaced by ones which have no variable in its non-complemented form. Hence *there is a cover of this function in the extended space which has variables only complemented variables* [2]. This implies that the function is ***monotonically decreasing*** in the extended space.

This is known as the *Unateness theorem.*

The Vertex Function ($\upsilon(x)$)

The vertex function is the representation in the new space of all the vertices of the old space B^n.

Theorem1.7: The vertices of the original function space in the new extended space are given by

$$\upsilon(X) = \prod_i (\bar{X}_{i0} \cdot X_{i1} + X_{i0} \cdot \bar{X}_{i1}) \tag{30}$$

Proof: The vertices of a function are those cubes C such that $|C| = 1$. If $|C| = 1$ we must have the condition that $\forall i$ $|C_i| = 1$.

$$|C_i| = 1 \Leftrightarrow C_i = \{0\} \cdot or \cdot C_i = \{1\} \tag{31}$$

If $|C_i| = 1$ then if $j \in C_i \Leftrightarrow 1 - j \notin C_i$

Thus in the extended space for a given i, the two C_i's of size 1 are $\bar{X}_{i0} \cdot X_{i1}$ and $X_{i0} \cdot \bar{X}_{i1}$. Thus Equation 30 is obtained by taking the product of all such C_i.

The Minterm Image and the Prime Image

Minterms ($\mu(x)$)

The minterms for the function in the extended space, are members of the vertex space which lie in the onset of the function.Thus for a term to be a minterm it must satisfy

$$\chi^F(X) = 1 \tag{32}$$

i.e. it belongs to the onset of the function and it must belong to the vertex function.

$$\upsilon(X) = 1. \tag{33}$$

This gives

$$\mu(X) = \chi^F(X)\upsilon(X) \tag{34}$$

The Primes ($\pi(x)$)

In order to calculate the primes of the function the following theorems are required. We also need to understand the concept of a *maximum point*; a maximum point is a point having a maximal number of variables in the non-complemented phase.

Definition: x is a maximal point of the function F iff

$$\forall z\,(\exists i\,(z \in F), \overline{z_i \geq x_i})\,. \tag{35}$$

Lemma 1.1: let C^α, C^β be cubes of B^n, then

$$C^\alpha \subseteq C^\beta \Leftrightarrow \Theta(C^\alpha) \leq \Theta(C^\beta) \tag{36}$$

Proof:

$$C^\alpha = C_1^\alpha \times C_2^\alpha \times \ldots \times C_n^{\ \alpha}$$

$$\text{if } C^\alpha \subseteq C^\beta \tag{37}$$

$$\Rightarrow \exists(i,j\,(j \in C_i^\beta)\,(j \notin C_i^\alpha))\,and\,\overline{\exists k\,(k \notin C_p^\beta)\,(k \in C_p^\alpha)}\;\forall p \tag{38}$$

$$\Rightarrow \exists x_{ij}\,((x_{ij}=0)_{in\Theta(C^\alpha)}\,and\,(x_{ij}=1)_{in\Theta(C^\beta)}) \tag{39}$$

$$\text{and} \quad \Rightarrow \forall x_{ij}\,((x_{ij}=1)_{in\Theta(C^\alpha)} \Rightarrow (x_{ij}=1)_{in\Theta(C^\beta)}) \tag{40}$$

$$\Rightarrow \Theta(C^\alpha) \leq \Theta(C^\beta)$$

$$\text{if } \Theta(C^\alpha) \leq \Theta(C^\beta) \tag{41}$$

$$\Rightarrow \exists x_{ij}\,((x_{ij}=0)_{in\Theta(C^\alpha)}\,and\,(x_{ij}=1)_{in\Theta(C^\beta)}) \tag{42}$$

$$\text{and} \quad \Rightarrow \forall x_{ij}\,((x_{ij}=1)_{in\Theta(C^\alpha)} \Rightarrow (x_{ij}=1)_{in\Theta(C^\beta)}) \tag{43}$$

$$\Rightarrow \exists(i,j\,(j \in C_i^\beta)\,(j \notin C_i^\alpha))\,and\,\overline{\exists k\,(k \notin C_p^\beta)\,(k \in C_p^\alpha)}$$

$$\Rightarrow C^\alpha \subseteq C^\beta$$

hence proved

Theorem 1.8: Any cube p is a prime of F iff $\Theta(p)$ is a maximum point of χ^F

Proof: If x and z are cubes of a function

$$(x \supseteq z) \Rightarrow (\Theta(x) \geq \Theta(z)) \tag{44}$$

$$(p \in Prime(F)) \Leftrightarrow \overline{\exists z\,(z \subseteq F)\ (z \supseteq x)} \tag{45}$$

From lemma 1.1 we have the following.

$$\overline{\exists z\,(z \subseteq F)\ (z \supseteq x)} \Leftrightarrow \overline{\exists z\,(\Theta(z) \geq \Theta(x))}\,. \tag{46}$$

The maximal points are those points such that $\overline{\exists z\,(\Theta(z) \geq \Theta(x))}$.

Hence the primes of the function are the represented by the maximal points in the extended space.

Theorem 1.9: Let G be any function, G monotonically decreasing in x, then

$$Max(G_x) \subseteq Max(G_{\bar{x}}) \tag{47}$$

Proof:

$$x \cdot A \in G \Rightarrow \bar{x} \cdot A \in G \tag{48}$$

where A is a cube. Thus

$$p \in G_x \Rightarrow p \in G_{\bar{x}} \tag{49}$$

where p is a cube. This implies

$$Max(G_x) \subseteq Max(G_{\bar{x}}) \tag{50}$$

Lemma1.2: Let G be any function, G monotonically decreasing, the maximum points of G are given by

$$Max(G) = x \cdot Max(G_x) + \bar{x} \cdot Max(G_{\bar{x}})\,\overline{G}_x \tag{51}$$

Proof:

I)

$$Max(G) \supseteq x \cdot Max(G_x) + \bar{x} \cdot Max(G_{\bar{x}})\, \overline{G}_x \tag{52}$$

a) $(p \in x \cdot Max(G_x)) \Rightarrow either\,(p \in Max(G))\; p = xs$
or $\exists q\,((q = xr), r > s, q \in G_x)$
$\Rightarrow (r \in G_x), r > s \quad \Rightarrow s \notin Max(G_x)$
a contradiction.
b)$p \in x \cdot \overline{G}_x \cdot Max(G_{\bar{x}}) \Rightarrow either\,(p \in Max(G))\; p = \bar{x}r$
or $\exists q\,(q > p)$

1) $either\,(q = \bar{x}s),\ (s > r), (s \in G_{\bar{x}})$
this contradicts the assumption that $r \in Max(G_{\bar{x}})$

2) $(q = xs),\ (s \geq r)$
$(s \in G_x) \Rightarrow (s \in G_{\bar{x}})$ as$G_x \subset G_{\bar{x}}$
$(s > r) \Rightarrow (r \notin Max(G_{\bar{x}}))$ a contradiction
$(s = r) \Rightarrow (r \in G_x)$ a contradiction

$$\Rightarrow Max(G) \supseteq x \cdot Max(G_x) + \bar{x} \cdot Max(G_{\bar{x}})\, \overline{G}_x \tag{53}$$

II)

$$Max(G) \subseteq x \cdot Max(G_x) + \bar{x} \cdot Max(G_{\bar{x}})\, \overline{G}_x \tag{54}$$

a) $p \in Max(G) \quad \Rightarrow either\,(p = xs)$
$\Rightarrow either\,(s \in Max(G_x))\, or\,(\exists r\,(r \in G_x)\;(r > s))$
$\Rightarrow xr > p \quad \Rightarrow p \notin Max(G_x)$
a contradiction
b) $(p = \bar{x}s) \Rightarrow (r \in G_{\bar{x}})$
if $(r \in G_x) \Rightarrow (xe \in G), (xr > p)$ we get a contradiction: $(p \notin Max(G))$
if $(r \notin Max(G_{\bar{x}})) \Rightarrow (\exists s\,(s > r), (s \in G_x)) \Rightarrow (\bar{x}s > p), (\bar{x}s \in G)$
we get a contradiction $(p \notin Max(G))$

$$\Rightarrow Max(G) \subseteq x \cdot Max(G_x) + \bar{x} \cdot Max(G_{\bar{x}})\, \overline{G}_x \tag{55}$$

from Equations 54 and 55 the lemma is proved.

lemma 1.3: The maximum points of G are also given by

$$Max(G) = x \cdot Max(G_x) + \bar{x} \cdot Max(G_{\bar{x}} \cdot \overline{G}_x) \tag{56}$$

Proof:

I) $Max(G_{\bar{x}} \cdot \overline{G}_x) \subseteq Max(G_{\bar{x}}) \cdot \overline{G}_x$

if $\exists p\,(\bar{x}p \in G)\ (p \in Max(G_{\bar{x}} \cdot \overline{G}_x)), (p \notin Max(G_{\bar{x}}) \cdot \overline{G}_x)$

then $\overline{(\exists q\,(q = \bar{x}t)\ (t \in G_{\bar{x}} \cdot \overline{G}_x), (t > p))}$ and $\exists q\,(q = \bar{x}t)\ (t \in G_{\bar{x}})\ (t \notin \overline{G}_x), (t > p)$

this is a contradiction.

II) $Max(G_{\bar{x}} \cdot \overline{G}_x) \supseteq Max(G_{\bar{x}}) \cdot \overline{G}_x$

if $\exists p\,(\bar{x}p \in G)\ (p \notin Max(G_{\bar{x}} \cdot \overline{G}_x)), (p \in Max(G_{\bar{x}}) \cdot \overline{G}_x)$

then since $G_x \subseteq G_{\bar{x}}$ and $Max(G_{\bar{x}}) \subseteq G_{\bar{x}}$

we have the statement $\exists q\,(q = \bar{x}r)\ (r \in Max(G_{\bar{x}} \cdot \overline{G}_x)), (r > p)$

$\Rightarrow (r \in G_x) \Rightarrow (r \notin \overline{G}_x) \Rightarrow (r \notin Max(G_{\bar{x}}) \cdot \overline{G}_x)$

a contradiction.

using lemma1.2 and the above, lemma 1.3 is proved.

The primes are calculated by considering F+D, namely the onset plus the don't care set of the function.

$$\pi_{F+D}(x) = Max(\chi^{F+D}) \tag{57}$$

Forming & Solving the Covering Problem

The covering table is a table representing the problem of finding the minimal prime irredundant cover of the function. The rows of this table correspond to the minterms of the onset of the function to be minimized and the columns of the table correspond to the primes of the onset plus the don't care set of the function. The entries of the table are 1 if a minterm is contained in a prime and 0 otherwise. We are looking for a minimum column cover for this table.

Having calculated the primes and the minterms of the function, we now formulate the covering problem as follows.

1. Form conditions for minterm dominance. $\alpha\,(x, z)$
2. Remove dominated minterms.
3. Form conditions for prime dominance $\beta\,(x, z)$
4. Remove dominated primes

5. Repeat steps 1 - 4 until no change is observed.
6. Solve the reduced covering problem.

It must be specified that in the case of completely specified functions, at the start of this process no prime will dominate another. However after the first round of removal of dominated minterms some primes begin to dominate each other.

Minterm Dominance ($\alpha(x, z)$)

A minterm x is said to dominate another minterm z, $(\alpha(x, z) = 1)$, if every prime covering x also covers z and there exists at least one prime which covers z but does not cover x. In such a case a cover involving x automatically covers z. Figure 3 illustrates this case. When x and z are covered by the same set of primes, we have the condition for *co-dominance*. If x and z co-dominate we may pick just one of the two as a representative term. We formulate the dominance condition as the converse of the condition for non-dominance, i.e the converse of the condition that there exists a prime covering x but not covering z.

$$\eta(x, z) = \mu(x) \cdot \mu(z) \overline{\exists u (\pi(u) \cdot \overline{u \supseteq z} \cdot u \supseteq x)} \tag{58}$$

Hence we have the complete condition for minterm dominance as

$$\alpha(x, z) = \eta(x, z) \cdot (\overline{\eta(z, x)} + x \neg z) \tag{59}$$

where $x \neg z$ is the tie-breaker whose purpose is to arbitrarily choose one representative term when 2 terms are *co-dominators*. *Co-domination* occurs when two minterms are covered by exactly the same set of primes, as shown in Figure 4. We remove all minterms which are dominated at any stage of the reduction. Alternately we keep all the minterms which are not dominated at any stage of the reduction. In order to remove dominated minterms we use the condition:

$$\mu_n(x) = \overline{\exists z \cdot \alpha(z, x)} \mu_{n-1}(x) \tag{60}$$

FIGURE 3. minterm x dominates minterm z

X		1				1
Z	1	1			1	1

Equation 60 is equivalent to step (3) of the Quine-McCluskey procedure. This is the step which removes all dominated rows (i.e. minterms).

FIGURE 4. co-dominators

X	1	1			1	1
Z	1	1			1	1

Prime Dominance ($\beta(x, z)$)

In order for a prime x to dominate another prime z, all minterms contained in z must be contained in x, but not vice versa. Alternately we may formulate this as the converse of the condition that there exists a minterm contained in z but not in x, in such a case x does not dominate z. Thus a cover involving x would automatically cover z and hence we may remove z from our covering problem. The condition is

$$\Upsilon(x, z) = \pi(x)\,\pi(z)\,\overline{\exists u\,(\mu(u) \cdot \overline{x \supseteq u} \cdot z \supseteq u)} \tag{61}$$

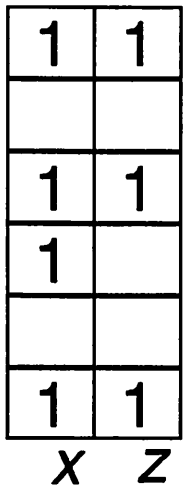
Similar to minterm dominance the complete formulation follows as:

$$\beta(x, z) = \Upsilon(x, z) \cdot (\overline{\Upsilon(z, x)} + x \neg z) \tag{62}$$

Figure 5. shows a case of *prime dominance*

The tie-breaker is identical to the previous case. Co-domination occurs when two primes cover exactly the same set of minterms at some stage of the reduction of the covering problem. We need to pick only one of the two co-dominators as a representative term. The tie-breaker is the condition for arbitrarily choosing one of two co-dominators, as shown in Figure 6.

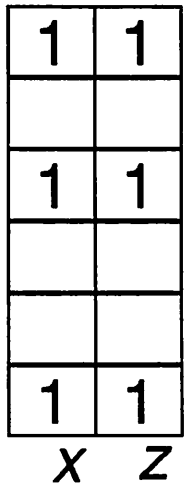
FIGURE 5. prime x dominates prime z



We remove all primes which are dominated by some other prime. Alternately we keep all primes which are not dominated. In order to remove all dominated primes we use the following expression.

$$\pi_n(x) = \exists z \overline{\beta(z, x)} \cdot \pi_{n-1} x \tag{63}$$

FIGURE 6. Prime co-dominators



This is equivalent to step3 of the Quine-McCluskey algorithm. For a completely specified function before the first pass of reduction, neither prime dominance nor co-dominance occur, however after removal of dominated rows or columns these conditions may come into play.

After the iterative removal of dominated primes and minterms, we are left with a reduced covering problem, which needs to be solved.

The result of this step are a set of rows and columns that are not dominated by any other row or column. These rows and columns form the *cyclic core* of the problem.

Handling Multiple Output Functions

In order to handle multiple output functions, we need to add additional bits of data corresponding to the output part. We need one additional bit of data for each multiple output. Thus in the extended space a function of m inputs and n outputs is represented by $2m+n$ bits. The occurrence of the j^{th} output bit implies that the j^{th} output part exists for the given input. This leads to complete representation in the extended space as shown in Equation 64.

$$\chi^F = \prod_j (X_{oj} \Rightarrow \chi^{F_j}) \tag{64}$$

where F_j is the j^{th} output and X_{oj} is the variable in the extended space which represents it. The above equation becomes

$$\chi^F = \prod_j (\bar{X}_{oj} + \chi^{F_j}) \tag{65}$$

The *vertex space and the null cube* need to modified in order to take into account this output part. A point is a member of the null space iff it is a null in its input or it is a null in its output part.It is an output null iff all the output variables are turned off, i.e. they are all in their complemented phase. This would imply that no output part (function) is present. Hence the *null cube* is given by

$$\phi_{final} = \phi_{output} + \phi_{input} \tag{66}$$

where the *null cube* for the input part is as previously calculated and the *null cube* for the output part is given by

$$\phi = \prod_j \bar{X}_{oj} \tag{67}$$

The new *vertex function* now includes both an input part and an output part. The point is a point in the vertex function if it is a point in the input vertex and the output vertex. It is a point in the output *vertex function* iff exactly one output variable is turned on. This is given by

$$\upsilon_{final} = \upsilon_{output} \cdot \upsilon_{input} \tag{68}$$

where the input vertex space is as calculated and the output vertex space is given by

$$\upsilon = \sum_j X_{oj} \prod_{i \neq j} \bar{X}_{oi} \tag{69}$$

The remaining computations remain identical.

BDD Representation and Implementation of functions

All the functions discussed in this paper are handled as BDD's. The BDD for the onset plus the don't care set and for the function alone are used as input. The characteristic BDD in the extended space is created by writing a recursive routine which evaluates the BDD in terms of the BDD node and the BDD's for the characteristic for the left branch and the right branch at each BDD node. A similar technique is used for prime computations. Thus at each stage the BDD of the results is the merged result of the BDD's for the left and right branches.

According to Bryant's analysis [4], the size of the BDD of the function $f_1 \cdot (op) \cdot f_2$ is bounded by $|f_1| \cdot |f_2|$ where $|\ |$ represents the size of the BDD. The operator could be any logical operator. It can be shown by analysis that the sizes of the null space and vertex space BDD's are essentially linear in the number of variables, thus if the extended space representation is of manageable size, it follows that so are the prime and minterm BDD's.

In order to implement the mapping into the extended space, the original BDD is traversed recursively, at each level the BDD of the extended function is written as a combination of the BDD's of the extended representation of the left branch and the right branch. This is a recursive formulation for the extended BDD in the form

$$extend(f) = (\bar{X}_{i0} + extend(f_{\bar{x}_i}))\ (\bar{X}_{i1} + extend(f_{x_i}))$$

This recursion terminates when the remaining BDD is either a "0" BDD or a "1" BDD. The pseudo-code for the calculation of the extended BDD is as follows:

```
extend(f)
if (look_up(f,value)) return value
else if (terminal_value(f,value)) return value
else
    Y_i= top_variable(f)
```

f_nex = extend(left_branch(f))
f_ex = extend(right_branch(f))
return $((\bar{x}_{i0}+f\_ex)(\bar{x}_{i1}+f\_nex))$

In order to make this calculation more efficient a *memoization* is used. This is the look_up routine used in the code. At each node first a check is made as to whether the node has been traversed; in that event it would have been stored in a look_up table and a lookup of the table yields the answer. In the other case it is computed and the computed BDD is inserted into the table using the BDD node as the key to insert the BDD. This method of hashing to avoid further computation is also used for the prime computation routine. Each node needs to be computed just once. *This technique relies heavily on the fact that a BDD is a canonical form and as a result each node in the BDD is unique[4].*

In order to do prime computation we use the equation

$$Max(G) = x \cdot Max(G_x) + \bar{x} \cdot Max(G_{\bar{x}} \cdot \overline{G_x}) \tag{70}$$

This formulation is recursive and based on computing the result for the left and right branches of the BDD first. Again a hash table is used to hash the value of the result BDD with the node as key.

As a result the pseudo-code for this computation becomes:

Max(G)
if look_up(G,value) return value
else if (terminal_value(f,value)) return value
else
x_{ij} *= top_variable(G)*
max_nx = Max((left_branch(G)) and not(right_branch(G))
max_x = Max(right_branch(G))
return(bdd_ite(x_{ij}*, max_x, max_nx))*

We traverse the BDD node by node. At each point we first check if the max point for a node has already been calculated, in such an event we merely perform a lookup of the hash table. If however it has not been computed it is computed using the above equation and then the value is stored in the hash table. This ensures that we compute the "Max" BDD at each node exactly once.

The Quine-McCluskey algorithm uses the BDD AND,OR and SMOOTH operators to implement. It is a straightforward implementation which essentially uses Equation 58-63 to form and segregate the minterm and prime dominators by using BDD AND for all logical ANDs, BDD OR for all logical ORs and BDD-SMOOTH for all the existential quantifiers, in the aforementioned equations. Thus the pseudo-code for the algorithm becomes:

Quine-McCluskey-reduction(primes,minterm)
While further reduction possible
minterms =bdd_and(minterms,bdd_not(minterms_dominated(minterm,primes)))

```
primes = bdd_and(primes,bdd_not(primes_dominated(minterm,primes)))

primes_dominated(minterms,primes)
gamma_1=bdd_not(bdd_and_smooth_with_u_vars(minterms_in_u_vars,u_vars_not_in_x_vars,u_vars_in_z_vars))
gamma_x_dom_z = bdd_and(primes_in_x_vars,primes_in_z_vars,gamma_1)
gamma_z_dom_x = bdd_swap_x_vars_z_vars(gamma_x_dom_z)
beta = bdd_and(gamma_dom_z, bdd_or(bdd_not(gamma_z_dom_x), x_vars_tie_z_vars))
return(beta)

minterms_dominated(minterms,primes)
eta_1=bdd_not(bdd_and_smooth_with_u_vars(minterms_in_u_vars,u_vars_in_x_vars,u_vars_not_in_z_vars))
eta_x_dom_z = bdd_and(minterms_in_x_vars,minterms_in_z_vars,eta_1)
eta_z_dom_x = bdd_swap_x_vars_z_vars(eta_x_dom_z)
alpha = bdd_and(eta_dom_z, bdd_or(bdd_not(eta_z_dom_x), x_vars_tie_z_vars))
return(alpha)
```

Ordering heuristics

Malik's "level" heuristics were originally used to order the variables to build the BDD's in the original space. In the extended space the new input variables are ordered according to the order of the corresponding variables in the original space. The *x,u and z* variables are interleaved. The output variables were ordered first. Thus the ordering in the extended space puts the output variables first and then the input variables. All *x,u and z* variables are interleaved.

After further experimentation we found that a much better ordering was achieved by ordering the support of each output part according to the aforementioned level heuristics and ordering each support set by its size. The output variables were ordered after their supports. This is essentially an application of [9] for ordering combinatorial circuits. All *x,u and z* variables were again interleaved.

Results

The Following table gives the results of the above method. There are many aspects of the problem which require further development. We need to find a means of removing the quantifiers from the expressions for prime and minterm dominance as these are the bottlenecks of the problem.

We found that the method worked well in some examples, giving an answer equal to the minimum obtained from Espresso-exact, however in the larger examples it failed at the quantifiers in Equations 58-63. Currently we are working

on an expression for domination relationships which eliminates the need for quantifiers. However as a result of our previous efforts, we have been able to solve a few of the 20 hard espresso problems; namely the examples *misj and misg*. We also able to build the prime and minterm BDD's for a large fraction of the remainder, namely *ex4, ex1010, ibm, jbp, misg, misj, mish, shift,soar.pla, ti,ts10,x2dn and xparc*. We are confident that further development will solve all of the 20 hard problems.

The results table for the "Benchmark" examples are given in the appendix.However table 1 & 2 show the behavior on a few example circuits.We observed that the problem size is very much a function of the ordering technique used and change in our ordering strategy can cause a phenomenal change in the size of the problem.In addition we need to formulate an implicit means to solve the reduced covering problem obtained.

TABLE 1. Results on the 20 hard espresso problems

Name	Input/ Output	Nodes in extended BDD (F)	Size of Prime BDD	Size of Minterm BDD	Primes	Minterms	Time to compute (in sec.)
ex4☛	128/28	43881	4238	3992	1.8348E14	computed	61.5
ex1010☛[a]	10/10	12662	40970	2655	25888	1471	4540.6
ibm☛	48/17	9507	16427	4975	1047948736	1.5523729E15	134.2
jbp☛	36/57	309019	80699	26461	2496809	8.0095268E11	2755.3
misg☛	56/23	350	770	889	6499491840	1.054609E18	3.4
mish☛	94/43	14109	503	605	139103	2.561545E11	1.3
misj☛	35/14	364	8784	6597	1.1243753E15	4.1494202E29	49.1
shift☛	19/16	7117	22831	6814	165133	4194304	383.2
soar.pla☛	83/94	45302	38797	7898	3.3047729E14	1.7458651E26	822.3
ti☛	47/72	143009	69678	28078	836287	4.136440E14	1923.3
ts10☛	22/16	88803	52143	52251	524280	4194304	1084.9
x2dn☛	82/56	18908	18393	7563	1.1488762E16	8.849739E25	194.1
xparc☛	41/73	232552	55839	11665	15039	1.0865220E13	1384.8

a. ☛ refers to one of espresso's 20 hard problems

TABLE 2. Results including reduction on some sample examples

Name	Input/ Output	Nodes in extended BDD	Size of Prime BDD	Size of Minter m BDD	Number of Primes	Number of Minterms	Primes After Reducing	Minterms after Reducing	Primes from espresso	Total Time (sec)
clpl	11/5	41	70	89	143	6713	20	20	20	1.9
cordic	23/2	336	310	224	1754	8634368	1712	1712	914	22.1
e64	65/65	321	511	577	65	36893488E20	65	65	65	24.0
misex1	8/7	136	207	162	28	548	12	12	12	12.0
misg☛	56/23	350	770	889	6499491840	1.054609E18	69	69	☛	102.1

TABLE 2. Results including reduction on some sample examples

Name	Input/ Output	Nodes in extended BDD	Size of Prime BDD	Size of Minter m BDD	Number of Primes	Number of Minterms	Primes After Reducing	Minterms after Reducing	Primes from espresso	Total Time (sec)
misj☛	35/14	364	503	605	139103	2.561545E11	35	35	☛	39.2
newapla	12/10	264	312	239	113	10421	17	17	17	34.9
newcpla2	7/10	301	282	197	38	282	19	19	19	28.3
rd84	8/4	247	228	125	633	411	255	255	255	6.8
t3	12/8	594	429	282	233	167920	33	33	33	86.5
vg2	25/8	1110	572	477	1188	61570752	110	110	110	187.9

References

[1] J.Madre, O. Coudert, Implicit and Incremental computation of primes and essential primes, DAC 1991.

[2] R Brayton, G. D. Hachtel, C. T. Mcmullen, A.L. Sangiovanni-Vincentelli, *Logic minimization algorithms for VLSI synthesis*, Kluwer Press 1984.

[3] O. Coudert, J. C. Madre, Bill Lin, *Symbolic Prime computation of multiple output boolean functions*, Bull research 1990.

[4] R. E. Bryant, *Graph based algorithms for boolean manipulations*, IEEE transactions on computers, Vol 35 1986.

[5] Karl S. Brace, Richard L. Rudell, Randall E. Bryant, *Efficient implementation of a Bdd package*, IEEE design automation conference 1989

[6] R. Rudell, *Logic synthesis for VLSI design*, E. R. L., College of Engineering, U. C. Berkeley 1989

[7] E. J. McCluskey, Minimization of Boolean functions, Bell Syst. Tech. Jour., Vol 35 1956

[8] K. Keutzer, D. Richards, *Computational complexity of Logic synthesis and optimization*, Proc INLS 1989

[9] H. Touati, H. Savoj, B. Lin, *Implicit state enumeration of finite state machines using BDD's*, ICCAD 1990

Appendix

Table of Results

The following table gives the calculated number of primes and minterms, primes and minterms after reduction and the number of primes in the minimum by espresso.

TABLE 3. Nodes in extended space representation

Name	Inputs	Outputs	Nodes in extended BDD (F+D)	Nodes in extended BDD (F)
al2	16	47	1681	1681
alcom	15	38	802	802
alu1	12	8	543	543
alu2	10	8	3356	5834
alu3	10	8	4414	7422
amd	14	24	5963	5963
apla	10	12	2271	1765
b10	15	11	39516	34136
b11	8	31	808	857
b12	15	9	830	830
b2	16	17	22125	22125
b3	32	20	42174	32075
b4	33	23	93489	62278
b7	8	31	808	857
b9	16	5	2250	2250
bc0	26	11	1923822	1923822
bca	26	46	23508	23118
bcb	26	39	19870	16645
bcc	26	45	15904	16562
bcd	26	38	7156	7762
br1	12	8	738	738
br2	12	8	558	558
chkn	29	7	46466	46466
clpl	11	5	41	41
cps	24	109	56447	56447
clip	9	5	4435	4435
con1	7	2	79	79
cordic	23	2	336	336
dc1	4	7	129	129
dc2	8	7	658	658
dekoder	4	7	91	125
dk17	10	11	1005	813
dk27	9	9	300	246
dk48	15	17	1303	1172

TABLE 3. Nodes in extended space representation

Name	Inputs	Outputs	Nodes in extended BDD (F+D)	Nodes in extended BDD (F)
duke2	22	29	7880	7880
e64	65	65	321	321
ex1010☛[a]	10	10	62822	12662
ex4☛	128	28	43881	43881
ex5	8	63	85985	85985
ex7	16	5	2250	2250
exp	8	18	3275	2794
exps	8	38	11570	8985
gary	15	11	16590	16590
ibm☛	48	17	9507	9507
in0	15	11	30138	30138
in1	16	17	22125	22125
in2	19	10	14810	14810
in3	35	29	15976	15976
in4	32	20	33414	33414
in5	24	14	11441	11441
in6	33	23	11446	11446
in7	26	10	6045	6045
inc	7	9	1375	1269
intb	15	7	56570	56570
jbp☛	36	57	309019	309019
lin.rom	7	36	16153	16153
luc	8	27	2034	2034
m1	6	12	622	622
m2	8	16	4064	4064
m3	8	16	3715	3715
m4	8	16	3319	3319
mark1	20	31	1902	1223
max1024	10	6	12254	12254
max128	7	24	3707	3707
max46	9	1	433	433
max512	9	6	2144	2144
misex1	8	7	136	136
misex2	25	18	869	869
misex3	14	14	92952	92952
misg☛	56	23	350	350
mish☛	94	43	14109	14109
misj☛	35	14	364	364
mp2d	14	14	360	360
newapla	12	10	264	264
newapla1	12	7	237	237

TABLE 3. Nodes in extended space representation

Name	Inputs	Outputs	Nodes in extended BDD (F+D)	Nodes in extended BDD (F)
newapla2	6	7	99	99
newbyte	5	8	85	85
newcond	11	2	299	299
newcpla1	9	16	1245	1245
newcpla2	7	10	301	301
newcwp	4	5	54	54
newill	8	1	60	60
newtag	8	1	28	28
newtpla	15	5	339	339
newtpla1	10	2	135	135
newtpla2	10	4	306	306
newxcpla1	9	23	2192	2192
p82	5	14	608	608
pope.rom	6	48	11291	11291
prom1	9	40	101324	101324
prom2	9	21	53547	53547
rd53	5	3	80	80
rd73	7	3	161	161
rd84	8	4	247	247
risc	8	31	941	941
ryy6	16	1	99	99
sex	9	14	560	560
opa	17	69	11477	11477
shift☛	19	16	7117	7117
soar.pla☛	83	94	45302	45302
spla	16	23	53285	53364
sqn	7	3	423	423
t1	21	23	13283	13283
t2	17	16	1433	1344
t3	12	8	594	594
t4	12	8	1767	1184
ti☛	47	72	143009	143009
table3	14	14	42637	42637
table5	17	15	73511	73511
tms	8	16	893	893
ts10☛	22	16	88803	88803
vg2	25	8	1110	1110
vtx1	27	6	4619	4619
wim	4	7	103	134
x1dn	27	6	4619	4619
xor5	5	1	31	31
x2dn☛	82	56	18908	18908

TABLE 3. Nodes in extended space representation

Name	Inputs	Outputs	Nodes in extended BDD (F+D)	Nodes in extended BDD (F)
x9dn	27	7	3090	3090
xparc☛	41	73	232552	232552
5xp1	7	10	381	381
9sym	9	1	178	178
z9sym	9	1	178	178

a. ☛ refers to one of espresso's 20 hard problems

TABLE 4. Number of Primes and Minterms

Name	Primes	Minterms	Time to compute
5xp1	390	576	1.1
9sym	1680	420	0.7
Z9sym	1680	420	1.0
al2	9179	191296	5.7
alcom	4657	88064	2.8
alu1	780	15872	1.2
alu2	434	7422	8.3
alu3	540	3903	10.3
alu4	7145	62256	162.4
amd	457	35072	21.8
apex1	6750	1.6482007E14	1704.2
apex2	13403	1.6481762E11	747.2
apex3	2700	5.8194951E16	4041.1
apex4	2336	2770	106.7
apla	201	157	6.3
b10	938	72912	60.8
b11	44	836	3.7
b12	1490	163072	2.7
b2	928	328488	97.5
b3	3056	1.3076E10	194.8
b4	6455	4.9942E10	202.8
b7	44	836	3.7
b9	3002	133704	5.1
bc0	6596	284933120	2331.2
bca	305	2778112	117.7
bcb	255	2417664	85.7
bcc	237	2477056	88.2
bcd	172	1699840	41.4
br1	29	114	1.5
br2	27	125	1.3

TABLE 4. Number of Primes and Minterms

Name	Primes	Minterms	Time to compute
bw	108	281	7.6
chkn	671	788036864	41.7
con1	24	156	0.2
cordic	1754	8634368	3.5
clpl	143	6713	0.2
cps	2487	124362704	287.4
dc1	22	47	0.3
dc2	173	442	1.3
dekoder	26	49	0.5
dk17	111	61	3.6
dk27	82	20	1.5
dk48	157	42	9.9
duke2	1044	8464768	20.9
e64	65	36893488E20	11.3
ex1010 ☛	25888	1471	4540.6
ex4 ☛	1.8348E14	computed	61.5
ex5	2532	7620	1460.9
ex7	3002	133704	4.7
exp	238	297	13.0
exps	852	1623	65.5
gary	706	84196	28.8
ibm ☛	1047948736	1.5523729E15	134.2
in0	706	84196	36.0
in2	666	686336	21.1
in3	1114	1.7485E11	83.9
in4	3076	1.3295E10	139.0
in5	1067	24912896	40.6
in6	6174	4.9950E10	31.0
in7	2112	220769280	15.4
inc	124	281	3.2
intb	6522	101720	131.2
jbp ☛	2496809	8.0095268E11	2755.3
lin.rom	1087	2306	84.7
luc	190	2198	7.7
m1	59	218	1.3
m2	243	831	7.1
m3	344	1105	9.7
m4	670	2134	18.6
mark1	208	2098128	32.1
max1024	1278	3232	27.8
max128	469	1616	16
max46	49	62	0.7

TABLE 4. Number of Primes and Minterms

Name	Primes	Minterms	Time to compute
max512	535	1616	11.3
misex1	28	548	0.5
misex2	42	37257216	1.8
misex3	6731	23196	456.7
misg ☛	6499491840	1.054609E18	3.4
misj ☛	139103	2.561545E11	1.3
mish ☛	1.1243753E15	4.1494202E29	49.1
mp2d	469	118544	2.0
newapla	113	10421	0.7
newapla1	31	380	0.3
newapla2	7	7	0.2
newbyte	8	8	0.2
newcond	72	704	0.6
newcpla1	170	1317	2.5
newcpla2	38	282	0.7
newcwp	23	42	0.2
newill	11	142	0.1
newtag	8	234	0.1
newtpla	40	4484	0.6
newtpla1	6	12	0.2
newtpla2	23	608	0.4
newxcpla1	191	3506	3.6
opa	477	732072	56.8
p82	48	81	1.0
pope.rom	593	1614	61.1
prom1	9326	8306	1892.4
prom2	2635	3027	276.4
rd53	51	42	0.2
rd73	211	192	0.7
rd84	633	411	1.3
risc	46	844	2.6
ryy6	112	19710	0.3
sao2	184	747	1.2
seq	7457	9.8390465E12	983.8
sex	99	1848	1.3
shift ☛	165133	4194304	383.2
soar.pla ☛	3.3047729E14	1.7458651E26	822.3
sqn	75	144	0.9
spla	4972	122736	680.4
square5	71	85	0.7
t1	15135	13956096	56.1
t2	233	167920	8.8

TABLE 4. Number of Primes and Minterms

Name	Primes	Minterms	Time to compute
t3	42	4096	1.2
t4	174	982	15.2
t481	481	42016	2.2
table3	539	11467	50.8
table5	462	119523	77.3
ti ☛	836287	4.136440E14	1923.3
tms	162	790	3.3
ts10 ☛	524280	4194304	1084.9
vg2	1188	61570752	2.5
vtx1	1220	133035072	6.5
wim	25	51	0.5
x1dn	1220	133035072	6.6
x2dn ☛	1.1488762E16	8.849739E25	194.1
x9dn	1272	133041984	6.9
xor5	16	16	0.1
xparc ☛	15039	1.0865220E13	1384.8

TABLE 5. Primes and Minterms after Reduction

Name	Primes After Reduction	Minterms after Reduction	Primes from espresso	Time (in sec)
al2	66	66	66	141.4
alcom	40	40	40	62.4
alu1	19	19	19	94.8
b11	27	27	27	144.2
b7	27	27	27	147.1
br1	19	19	19	87.2
br2	13	13	13	58.4
clpl	20	20	20	1.9
con1	9	9	9	4.0
cordic	1712	1712	914	22.1
dc1	9	9	9	5.9
dc2	39	39	39	204.6
dekoder	12	12	9	5.7
dk27	14	14	10	48.8
e64	65	65	65	24.0
inc	29	29	29	304.3
m1	19	19	19	65.6
misex1	12	12	12	12.0
misex2	28	28	28	192.7
max46	46	46	46	25.2

TABLE 5. Primes and Minterms after Reduction

Name	Primes After Reduction	Minterms after Reduction	Primes from espresso	Time (in sec)
misg ☛	69	69	☛	102.1
misj ☛	35	35	☛	39.2
newapla	17	17	17	34.9
newapla1	10	10	10	4.1
newapla2	7	7	7	1.1
newbyte	8	8	8	1.0
newcond	31	31	31	49.3
newcpla1	40	40	38	4389.8
newcpla2	19	19	19	28.3
newcwp	11	11	11	2.6
newill	11	11	8	1.0
newtag	8	8	8	0.5
newtpla	23	23	23	18.1
newtpla1	4	4	4	1.2
newtpla2	9	9	9	12.5
p82	21	21	21	38.7
rd53	31	31	31	1.2
rd73	127	127	127	3.9
rd84	255	255	255	6.8
risc	28	28	28	226
ryy6	112	112	112	1.8
sao2	58	58	58	88.8
sex	21	21	21	73.2
sqn	38	38	38	35.7
squar5	35	35	25	94.4
t3	33	33	33	86.5
vg2	110	110	110	187.9
wim	12	12	9	5.5
xor5	16	16	16	0.2
5xp1	146	156	63	322
9sym	1680	420	84	2.6
Z9sym	1680	420	84	2.8

The total number of primes and minterms for all examples is given by the following table.

TABLE 6. Size of Prime and Minterm BDD's

Name	Size of Prime BDD	Size of Minterm BDD
5xp1	464	169
9sym	162	96
Z9sym	162	96
al2	990	936
alcom	635	571
alu1	742	509
alu2	1689	1279
alu3	2231	1602
alu4	12059	3174
amd	4125	1899
apex1	67731	19265
apex2	15858	8594
apex3	126934	24568
apex4	28307	8873
apla	11865	495
b10	3234	1702
b11	744	572
b12	958	512
b2	10905	2242
b3	12318	4282
b4	11562	4578
b7	744	572
b9	1716	754
bc0	28750	9473
bca	5060	3953
bcb	16645	3930
bcc	4072	3025
bcd	2310	2050
br1	473	414
br2	367	316
bw	1968	595
chkn	4195	1863
con1	120	86
cordic	310	224
clpl	70	89
cps	33255	9725
dc1	145	88
dc2	499	345
dekoder	145	86
dk17	760	317

TABLE 6. Size of Prime and Minterm BDD's

Name	Size of Prime BDD	Size of Minterm BDD
dk27	390	178
dk48	1278	611
duke2	4667	2552
e64	511	577
ex1010 ☛	40970	2655
ex4 ☛	4238	3992
ex5	70225	3397
ex7	1716	754
exep	6878	10113
exp	1794	707
exps	6933	2814
gary	4714	2120
ibm ☛	16427	4975
in0	4504	2062
in1	10905	2242
in2	3736	2010
in3	10729	3799
in4	12609	4974
in5	6691	2764
in6	6720	2515
in7	4125	1438
inc	721	298
intb	11423	3089
jbp ☛	80699	26461
lin.rom	11033	1966
luc	2163	817
m1	550	260
m2	2115	791
m3	2784	946
m4	4249	1339
mark1	2597	811
max1024	3465	1091
max128	3733	1014
max46	215	186
max512	2026	961
misex1	207	162
misex2	631	645
misex3	21291	10755
misg ☛	770	889
misj ☛	503	605
mish ☛	8784	6597
mp2d	600	318

TABLE 6. Size of Prime and Minterm BDD's

Name	Size of Prime BDD	Size of Minterm BDD
newapla	312	239
newapla1	141	129
newapla2	96	96
newbyte	82	82
newcond	228	166
newcpla1	787	464
newcpla2	282	197
newcwp	71	60
newill	58	55
newtag	36	44
newtpla	249	213
newtpla1	92	90
newtpla2	153	118
newxcpla1	1155	494
opa	9512	4171
p82	436	292
pope.rom	10630	1798
prom1	49464	10177
prom2	23370	5311
rd53	74	55
rd73	151	96
rd84	228	125
risc	806	635
ryy6	80	95
sao2	407	243
seq	50715	9296
sex	623	404
shift ☛	22831	6814
soar.pla ☛	38797	7898
sqn	290	207
spla	37353	24299
square5	341	214
t1	9126	2626
t2	1435	980
t3	429	282
t4	959	525
t481	518	293
table3	4003	2458
table5	5337	2832
ti ☛	69678	28078
tms	1065	417
ts10 ☛	52143	52251

TABLE 6. Size of Prime and Minterm BDD's

Name	Size of Prime BDD	Size of Minterm BDD
vg2	572	477
vtx1	1462	1062
wim	154	88
x1dn	1462	1062
x2dn ☛	18393	7563
x9dn	1564	1308
xor5	25	25
xparc ☛	55839	11665