An Exact Logic Minimizer Using Implicit Binary Decision Diagram Based Methods

Gitanjali M. Swamy

Masters Thesis

December 1993

Department of Electrical Engineering and Computer Science

University of California, Berkeley

Abstract

We present an exact method for minimizing logic functions using BDD's. In this method the function is mapped to an extended space which gives it special properties that can 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.

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 Co-NP-hard and hence no polynomial time algorithm is known to exist [3].

One particular physical implementation of the logic function is the Programmable Logic Array or PLA. One of the major advantages of a PLA is the fact that the symbolic representation of the function can be mapped to the physical representation in a very straightforward way. Thus there is a high correlation between the size of the function representation and the size of the PLA used to represent the function. If the logic function is represented in the so called *Sum of Products (SOP)* form, it may be translated directly on to the PLA. The Sum of Products representation consists of a logical OR of a series of product terms. Each product term consists of a logical AND of boolean variables or their complements, which belong to the input of the function. The Physical Area of the PLA is a direct function of the size of the *Sum of products* representation chosen. Rows in the PLA corresponds to products in the *SOP* representation and Columns to individual variables in the input. By minimizing the number rows and columns of the PLA the physical area is minimized. In order to do so, the product terms in the SOP representation are made as "large" as possible. This means that the products are explicitly dependant on as few variables as possible. These "maximal" product terms are called *Primes*. Our objective is to get a representation for the function as a minimum sized set of these maximal product terms. It also important to note that products chosen to represent the function must completely specify the original function. In order to do so the *SOP* must contain a basic set of terms which must be in the function representation. Each product may actually contain more than one of these terms. These terms are specified as products which have every variable of the input present exactly once as either complemented or non-complemented form. Such products are called vertices and vertices which are used to specify the function are called *Minterms*. In order to check if a SOP representation is complete, we need to check that every one of these minterms is contained within our representation.

Since the *SOP* product representation involves the logical OR of a set of logical ANDS, there are only two levels in the hierarchy of the logic. Thus this representation is called a *two-level logic representation*. The PLA area minimization problem becomes the *two-level logic minimization* problem. There are many algorithms to perform this minimization. The Quine-McCluskey procedure[6] is an exact algorithm used for solving the problem of two level logic minimization; i.e. it minimizes the number of rows used in a PLA implementation of the function. It important to keep in mind that PLA implementations are just one of a large class of problems for which two level logic minimization is used. This problem has a large spectrum of applications ranging from state-encoding to general logic optimization. Though PLA's provide a convenient framework to understand the problem; they are not even the most important application.

In brief the basic Quine-McCluskey tabular minimization proceeds as follows:

1) It finds all the Prime implicants (potential rows of PLA) of the function
2) It constructs the Prime-implicant table, which relates Primes to Minterms (vertices which must be included in the function representation).
3) It determines the rows and columns of the table, whose information is entirely contained within some other row or column and deletes them.
4) It repeats the previous step until no more reduction is possible.

The remainder at this stage is called the *cyclic core* of the problem. Finally the Quine-Mccluskey procedure finds the minimum column cover for the cyclic core of the problem using branch and bound algorithms. This procedure may be carried out *Explicitly* or *Implicitly. Explicit* cases include those implementations where each *Prime or Minterm* is handled individually. The converse holds true for the *Implicit* implementations, where groups of Primes and Minterms are handled together as a single entity.

Though this algorithm may be used effectively for many examples, it fails in its explicit form for some large examples with an exponential number of primes. For example the espresso-exact algorithm[1] fails for the *mish*[8] example; even though the actual cover is quite small in comparison. An effective solution to this problem lies in the *Signature Cube* methods, developed by Sanghavi et al.[13], which does not enumerate all the primes and minterms of the function. However this method has the intrinsic disadvantage that it cannot list all the primes and minterms of the function.

Recent work [5, 6], provided us with a new implicit approach to this problem. In those papers, O. Coudert & J.C. Madre have developed a new method of representing primes of Boolean functions. Through their techniques they have been able to arrive at a representation of the primes of the largest and most difficult of the public benchmark functions. We extend [9,10] the techniques of [5, 6] to exact minimization of boolean functions. The method we propose here relies on the following statement: *Any precise set (e.g. Boolean function) can be phrased as a propositional sentence over the appropriate boolean space.* Thus the primes, minterms, as well as terms in the Quine-McCluskey reduction may be formulated as propositional sentences.

Briefly the primes and the minterms required and the covering table are represented implicitly, and step (3) of the Quine-McCluskey procedure is represented as operations over this implicit representation. We then arrive at the cyclic core 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 can solve some of those problems which have not yet been solved by explicit methods[1,8], by applying branch and bound techniques to this remaining problem.

A BDD [2] or a binary decision diagram is a DAG (Directed Acyclic Graph) data structure (defined in the next section). Operations on this type of data structures are a function of the number of nodes in the DAG, whereas the number of terms it represent are dependent on the number of paths through the DAG. 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[2]. The BDD[2] 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. BDD's provide an efficient means of representing groups, rather than individual terms. 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 combinational function.

Given any logical proposition(statement) over a finite boolean space, one can find the solution set to the proposition by a series of BDD operations on the proposition; in fact there is a direct correlation between logical operations in the proposition and BDD operations. It turns out that the BDD representations of formidable propositions (i.e. those with a very large explicit representation) 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 this thesis 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.

Definitions

Boolean space: A boolean space B^n or $\{0, 1\}^n$ is a space of n variables which may only take the values 0 and 1. The rules of boolean logic within this space.

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 is 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 1 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.

Multiple Output function: A multiple output function is a function which has more than one output variable. These functions may be reduced to single output functions by adding a new input variable for each multiple output and creating a new single output function using these variables and the original input variables (the exact construction is given in the section on multiple output functions). The new input variables are called *multi-output* variables.

Vertex: A vertex is a single point in the subspace corresponding to the function input and multi-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 0 to value 1 causes the function value, if it changes, to go from value 1 to value 0.

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, i.e. C_i is a subset of *{0,1}*, e.g. the cube $\{0, 1\} \times \{0\} \times \{1\}$ over B^3 is written as $\overline{X}_2 X_3$. 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.

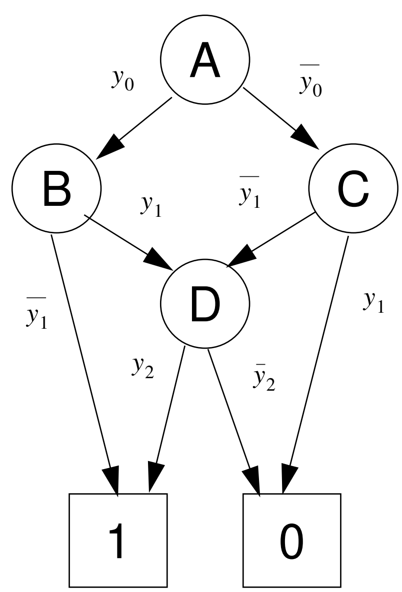
Implicant: An implicant of an incompletely specified function *(f,d,r)* is a cube *C*, such that no vertex contained in this cube belongs to the offset *r*.

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*, i.e. as $f = \bar{x}f_{\bar{x}} + xf_x$. This leads to the concept of a Shannon cofactoring 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, eventually the remaining function is either 0 or 1. This terminates the tree and we get an expanded function. If we represent each level of 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.

Figure 1. BDD for function F



Binary Decision Diagram: [2] 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 $value(v) \in \{0, 1\}$. 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 which the function is expanded at the given level.

This representation is a canonical form. The root node corresponds to the variable with respect to which we cofactor and one branch corresponds to the BDD for the cofactor of the function with respect to $\bar{x}$ and the other 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 $F = y_0 \cdot \overline{y_1} + y_0 \cdot y_2 + \overline{y_1} \cdot y_2$

Covering Table: The covering table M_f of function *f*, is used to solve the problem of finding the smallest prime irredundant cover of the function. The rows of this table correspond to the minterms and the columns 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. A *minimum prime cover* corresponds to minimum column cover for this table.

Row Dominance or Minterm Dominance: A row (minterm) is said to dominate another row (minterm) if and only if 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). *Strict Minterm Dominance* is said to occur when, in addition, there exists a prime containing the second minterm which does not contain the first.

Column Dominance or Prime Dominance: A column (prime) is said to dominate another column (prime) if and only if 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). *Strict Column Dominance* is said to occur when, in addition, there exists a minterm (row entry =1) which is contained in the first prime (column) but not in the second prime (column).

Cyclic Core: The primes and minterms remaining in the covering table after all *dominated* primes and minterms are removed, form the cyclic core of the original problem. There are no dominated primes or minterms remaining in the *cyclic core*. In order to get the final solution to the problem, *branch and bound* methods have to applied to this cyclic core.

Quantification: There are two *quantifiers*, $\exists x$ and $\forall x$. The first is the *Existential Quantifier*. If there exists a vertex *x* such that some logic function $f(x)$ is 1, this is shown as $\exists x f(x) = 1$. The second quantifier is the *Universal Quantifier*. If for all cubes x some condition $f(x)$ is 1, this is written as $\forall x f(x) = 1$. The relation between these operators is $\forall x F(x) \Leftrightarrow \overline{\exists x (\bar{F}(x))}$ and $\exists x F(x) \Leftrightarrow \overline{\forall x \bar{F}(x)}$.

Support: The variable support of a function *F or supp(F)* is the set of variables on whom the function explicitly depends, i.e., changing the value of any variable (i.e. its complement too) within *supp(F)*, while keeping all other variables constant, changes the value of the function *F.* For example if *F= a* where the input space consists of variables *{a,b}*, then *supp(F) = {a}*.

Propositional Formulae: A propositional formula is a valid boolean formula composed of quantifiers, boolean operands and variables. This formula can be used to represent a set variables satisfying a logical proposition or statement.

In addition we will also define a set of *temporary variables z* and *u,* which will be used during computation. The purpose of these variables is temporary storage during intermediate computations.

The Quine-McCluskey procedure

The Quine-McCluskey tabular minimization procedure follows the given 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 its column prime, otherwise the entry is 0. Our problem is to find a *minimum column cover* for all the rows. The *essential primes* are represented by those columns such that at least one of their 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 step 3 until no further 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. In general *branch and bound algorithms* are used for this step.

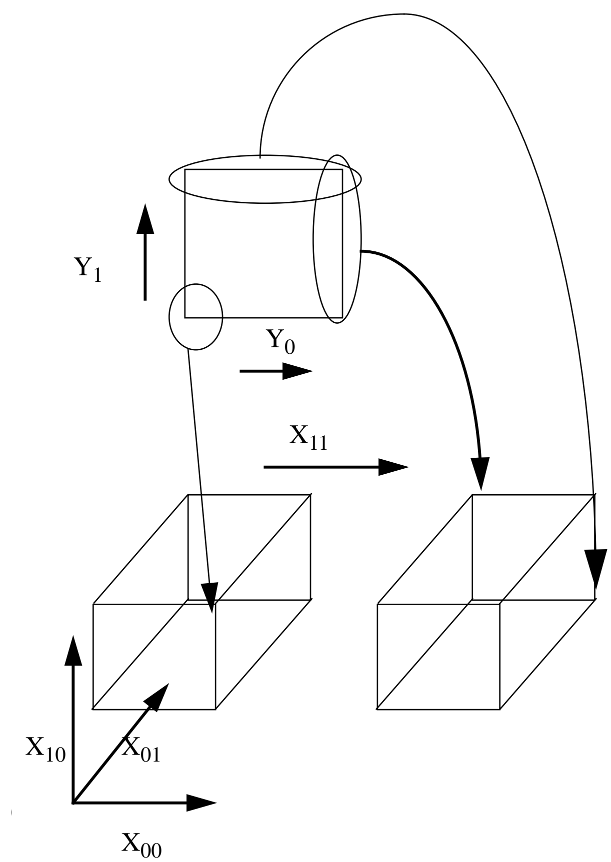
In practice the main bottleneck in *Espresso-Exact* [8] is the problem of prime explosion. The number of primes for n input variables can potentially be as large as $3^n/n$ and hence for larger examples the number of primes may become too large to enumerate, even when the size of the cyclic core is small. For example, the Espresso-exact algorithm fails on the example circuit “mish” [8] because it has 10^{14} primes but a cyclic core with just 82 primes[9,10].

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 propositional formulae are sets of *points* in a space but the Quine-McCluskey algorithm functions over cubes, which are sets of *collections of points.* The first step 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.

Figure 2. The Coded Cube Space



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 possible subsets of {0,1}, it follows there are 4^n vertices of any extended space; i.e. the extended space of B^n is B^{2n}. 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 $1 \le i \le n$, $j \in \{0, 1\}$ and *(z,u)* represent temporary variables. Assuming all variables are binary valued in the original space, we choose the following extended space, using *2n* variables:

<u>Definition</u> $\Theta(C)$: 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*
$\Theta(C) = (X_{10} \cdot X_{11} \cdot X_{20} \cdot \ldots \cdot X_{n1})$ *, where* $X_{ij} = 1$ *if* $j \in C_i$ *and* $X_{ij} = 0$ *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.

<u>Theorem</u> 1.1: The mapping Θ is 1-1.

<u>Proof</u>: *By contradiction.*

Assume the converse.

Assume there exists a cube $C_1 \times C_2 \times \ldots \times C_n$ which maps to at least 2 points *a* and *b*.
Consider some variable X_{ij} in which *a,b* differ.
$X_{ij} = 0$ for *a*.
$X_{ij} = 1$ for *b*.

This is a contradiction since

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

and

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

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

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 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.

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

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

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.

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

Proof: *By construction from the definition of* $\chi^F(x)$

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

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

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

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

$\Leftrightarrow (x \in \chi^F) \bullet and \bullet (x \in \chi^G)$

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

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.

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

Proof: *By construction from the definition of* $\chi^F(x)$.

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

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

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

Every prime of *F+G* is either a prime of *F* or a prime of *G*. If some cube is formed by combining cubes of *F* and *G*, then it follows that there is a prime covering this cube, which is formed by combining cubes in *F* and *G* but is neither in *F* nor in *G*. This is a contradiction. Hence

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

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

Key Theorem of implicant characteristic

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

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

Proof: *By case analysis on the implicants of function F.*

The above is equivalent to

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

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

$\chi^0 = 0$

and

$\chi^1 = 1$

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

$F = Y_i$ 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.

$F = \bar{Y}_i$ 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.

F=0 and *F=1* are covered by the base case.

The function F can be written as

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

which factors as

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

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

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

Using the fact that every prime of (Y_i *or* $F_{\bar{Y}i}$) is either a prime of Y_i *or a prime of* $F_{\bar{Y}_i}$ and a similar result for ($\bar{Y}_i$ *or* F_{Yi})(excluding the null cube) and theorem 1.3 we have the following.

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

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

<u>Lemma </u>1.1: Let C^α, C^β be cubes of B^n, then

$$C^\alpha \subseteq C^\beta \Leftrightarrow \Theta(C^\alpha) \le \Theta(C^\beta)$$

where $a \le b$ if and only if there is no variable X_{ij}, such that $b_{X_{ij}} = 0$ and $a_{X_{ij}} \neq 0$

<u>Proof</u>: *By assuming one side of the given implication and proving the other for both directions of the given implication*

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

$$C^{\alpha} \subseteq C^{\beta}$$

$$\Rightarrow (\forall p)\, \overline{\exists k\,(k \notin C_p^{\beta})\,(k \in C_p^{\alpha})}$$

$$\Rightarrow \forall X_{ij}\,(((\Theta(C^{\alpha}))_{X_{ij}}) \Rightarrow ((\Theta(C^{\alpha}))_{X_{ij}}))$$

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

Conversely $\Theta(C^{\alpha}) \leq \Theta(C^{\beta})$

$$\Rightarrow \forall X_{ij}\,(((\Theta(C^{\alpha}))_{X_{ij}}) \Rightarrow ((\Theta(C^{\alpha}))_{X_{ij}}))$$

$$\Rightarrow (\forall p)\, \overline{\exists k\,(k \notin C_p^{\beta})\,(k \in C_p^{\alpha})}$$

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

Technical Functions over the CCS

The Null cube function and the Vertex function together, form the set of technical functions. Both these functions do not depend on the original function in any way, but are merely properties of the extended space. The technical functions are functions of the Coded Cube Space alone.

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

<u>Theorem</u> 1.5: The null cube set is given by

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

<u>Proof</u>:

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

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

$(0 \notin C_i)\ and\ (1 \notin C_i) \Rightarrow C_i = \phi$

$C_i = \phi \Rightarrow C = \phi$

Conversely

$C = \phi \Rightarrow \exists i C_i = \phi$

$(C_i = \phi) \Rightarrow ((0 \notin C_i)\ and\ (1 \notin C_i))$

$(1 \notin C_i) \Rightarrow (X_{i1} = 0)$ and $(0 \notin C_i) \Rightarrow (X_{i0} = 0)$

Adding the null cube to an expression does not add any vertices to the function. 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.

The Unateness Theorem

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

Proof: *By using lemma 1.1.*

Consider any cube of the form $\bar{X}_{i0} \cdot X_{i1} \cdot A$, $X_{i0}, X_{i1} \notin Supp(A)$

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

In addition $\bar{X}_{i0} \cdot \bar{X}_{i1} A \subseteq \bar{X}_{i0} \cdot \bar{X}_{i1}$. Thus we have

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

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

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

Similarly $\bar{X}_{i1} A$ is a cover for a cube of the form $X_{i0} \cdot \bar{X}_{i1} A$

Consider any cube of the form

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

This 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.

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

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

occur. Hence if the null cubes are 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 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 only complemented variables* [1]. This implies that the function is ***monotonically decreasing*** in the extended space.

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

The vertex function is the extended space representation of all the vertices of the space B^n.

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

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

Proof: *By construction using the observation that* $|C_i| = 1$ *for all vertex points.*

The vertices of a function are those cubes C such that $|C| = 1$.

$|C| = 1 \quad \Leftrightarrow \quad \forall i \; |C_i| = 1$.

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

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

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 $X_{i0} \cdot \bar{X}_{i1}$. Hence $\upsilon(X)$ 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

$$\chi^F(X) = 1$$

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

$$\upsilon(X) = 1.$$

This results in the following expression for the minterms of the original function, in the extended space.

$$\mu(X) = \chi^F(X)\upsilon(X)$$

Primes ($\pi(x)$)

In order to calculate the primes of the function, in the extended space, theorems 1.8 and 1.9 are required. We also need to understand the concept of a *maximal point*.

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

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

Let *Max(F)* denote the *maximal points* of function F.

<u>Theorem</u> 1.8: Any cube p is a prime of F iff $\Theta(p)$ is a maximal point of χ^F

<u>Proof</u>: *By construction using the definition of* $\Theta(p)$

If x and z are cubes of a function

$(x \supseteq z) \Rightarrow (\Theta(x) \geq \Theta(z))$

$(x \in Prime(F)) \Leftrightarrow \overline{\exists z (z \subseteq F) \ (z \neq x) \ (z \supseteq x)}$

From lemma 1.1 we have the following.

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

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.

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

<u>Theorem</u> 1.9: Let *G* be any function, *G* monotonically decreasing in *x*, then

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

<u>Proo</u>f:

G is monotonically decreasing in x.

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

where A is a cube, $x \notin Supp(A)$. Thus

$A \in G_x \Rightarrow A \in G_{\bar{x}}$

This implies

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

<u>Lemma</u>1.2: Let *G* be any function, *G* monotonically decreasing, the maximum points of *G* are given by

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

<u>Proof</u>: *By contradiction.*

I)

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

a) $(p \in x \cdot Max(G_x)) \Rightarrow either(p \in Max(G))\ p = xs$ where $x \notin Supp(s)$ Or $\exists q((q = xr), r > s, q \in G_x)$

where $x \notin Supp(r)$

if $(p \notin Max(G))$ then $\exists q((q = xr), r > s, q \in G_x)$

$\Rightarrow (r \in G_x), r > s$

$\Rightarrow s \notin Max(G_x)$

$p \notin x \cdot Max(G_x)$

a contradiction.

b) $p \in x \cdot \overline{G}_x \cdot Max(G_{\bar{x}}) \Rightarrow either(p \in Max(G))\ p = \bar{x}r$ where $x \notin Supp(r)$

Or $\exists q(q > p)$ such that

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

2) Or $(q = xs), (s \geq r)$

$(s \in G_x) \Rightarrow (s \in G_{\bar{x}})$ as $G_x \subset G_{\bar{x}}$

Hence if $(s > r) \Rightarrow (r \notin Max(G_{\bar{x}}))$ a contradiction

Or if $(s = r) \Rightarrow (r \in G_x)$ a contradiction

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

II)

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

$p \in Max(G) \quad \Rightarrow either(p = xs)\ or\ (p = \bar{x}s)$ where $x \notin Supp(s)$

a) $(p = xs) \quad \Rightarrow either(s \in Max(G_x))\ or\ (\exists r(r \in G_x)\ (r > s))$ where $x \notin Supp(r)$

if $s \notin Max(G_x)$ then $(\exists r(r \in G_x)\ (r > s))$

$\Rightarrow xr > p \quad \Rightarrow p \notin Max(G_x)$

a contradiction

b) Or $(p = \bar{x}s) \Rightarrow (r \in G_{\bar{x}})$

if $(r \in G_x) \Rightarrow (xr \in G), (xr > p) \quad \Rightarrow (p \notin Max(G))$ a contradiction:

if $(r \notin Max(G_{\bar{x}})) \Rightarrow (\exists s(s > r), (s \in G_x)) \Rightarrow (\bar{x}s > p), (\bar{x}s \in G) \quad \Rightarrow (p \notin Max(G))$

a contradiction

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

hence proved.

<u>Lemma</u> 1.3: The maximum points of G are also given by

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

Proof: *By using lemma1.2*

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

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

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

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 of F+D, namely the onset plus the don't care set of the function are computed in order to form the covering table.

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

Forming & Solving the Covering Problem

The Covering table is a table used to solve the problem of finding the *minimal prime irredundant cover* of a 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* of 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.

It is important to note that since we are in the extended space, when we talk about cube containment, we mean cube containment in the original space. However cube containment in the original space translates to the notion of maximality in the extended space, as shown in Lemma1.1. Thus the containment operator $\subseteq$ is replaced by the maximality operator $\leq$, in the extended space, whenever cube containment is required in the original space. Recall, that $a \leq b$ if and only if there is no variable X_{ij}, such that $b_{X_{ij}} = 0$ and $a_{X_{ij}} \neq 0$.

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

A minterm x is said to strictly 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-dominanc*e. If x and z co-dominate we may pick just one of the two as a representative term.

Figure 3. Minterm x dominates Minterm z

X		1				1
Z	1	1			1	1

Figure 4. Minterm Co-Dominators

X	1	1			1	1
Z	1	1			1	1

We formulate the general dominance (not strict) condition as the converse of the condition for non-dominance. Minterm x cannot dominate minterm z if there exists a prime covering x but not covering z.

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

If x dominates z, $(\eta(x, z) = 1)$ and z does not dominate x, $(\eta(z, x) = 0)$ then we can remove z from the covering table; since a cover for x automatically implies a cover for z. Recall that this is due to the fact that all primes covering x automatically cover z but all primes covering z do not necessarily cover x. This is the condition for *strict minterm dominance*. This condition, however, does not include all minterms we want to retain, specifically the set satisfying the condition that x and z are covered by exactly the same set of primes from the covering table; i.e. under the condition for *co-dominance*. Under co-dominance $(\eta(x, z) = 1)$ and $(\eta(z, x) = 1)$ are satisfied, and we need to choose one representative between x and z, to keep in the covering table. This leads to the following expression for strict minterm domination and minterm co-domination representation.

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

where $x \neg z$ is the tie-breaker whose purpose is to arbitrarily choose one representative term when two terms are *Co-dominators*. We need to retain the strict dominators as well as representatives for every pair of co-dominators in the covering table. Minterms z which satisfy the condition that there exists some other minterm x such that the following condition $(\alpha(x, z) = 1)$ is met, must be removed form the covering table. *Minterm co-domination* occurs when two minterms are covered by exactly the same set of primes, as shown in Figure 3.

We remove all minterms which are strictly dominated or are not co-domination representatives at any stage of the reduction. Alternately we keep all the minterms which are not strictly dominated and are co-domination representatives at any stage of the reduction. Minterms are retained as follows.

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

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

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

In order for a prime x to strictly 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.

We obtain the following expression for general prime dominance (not strict) in a manner very similar to minterm dominance.

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

If all the minterms covered by prime x are covered by prime z, $(\Upsilon(x, z) = 1)$ and all the minterms covered by prime z are not necessarily covered by prime x, $(\Upsilon(z, x) = 0)$ then we could safely remove z from the covering table, as a cover for x would automatically imply a cover for z. This is the condition for *strict prime dominance*. However there is a set of primes we need to include in the covering table, in addition to strict dominators. When x and z cover exactly the same set of minterms in the covering table, the condition of *prime co-dominance* is met and we need to pick one representative for every pair of *co-dominators*. Primes x and z co-dominate if and only if $(\Upsilon(x, z) = 1)$ and

$(\Upsilon(z, x) = 1)$. This leads to the following expression for strict dominance as well as co-dominance representation.

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

Figure 5. Prime x dominates Prime z

1	1
1	1
1	
1	1
X	Z

Figure 6. Prime co-dominators

1	1
1	1
1	1
X	Z

The tie-breaker is the condition for arbitrarily choosing one of two Co-dominators, as shown in Figure 6. A case of *strict prime domination* is shown in Figure 5. We choose to retain those primes x which satisfy the condition that there does not exist another prime z, which satisfies the condition $(\beta(x, z) = 1)$.

We remove all primes which are dominated (strictly or co-dominated) by some other prime. Alternately we keep all primes which are not dominated. Primes are retained for the next stage of reduction using the following expression.

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

This is equivalent to step 3 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. This is because if two primes of a completely specified function contain the same set of minterms of the onset, then they must be identical as they cover exactly the same set of points (There are no don't care points to account for the difference).

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.

A Quantifier Free Formulation for solving the covering problem

The previous formulation for Prime and Minterm Dominance has two major disadvantages. Apart from the disadvantage that it requires an intermediate formula to be built in thrice the number of original variables (instead of x alone, x,u and z are used), quantifiers are used in the formulations and these tend to increase computations immensely. We propose a new method to compute this relations using certain special properties of our extended space. This new method relies on traversing the extended space representation of the prime and minterm BDD's in parallel and computing dominating and co-dominating terms based on this traversal.

The new formulation is based on the fact that if the representation of any cube of the original space covers another cube of the original space, then the representation of the former cube in the extended space cannot have any variable x in its complemented phase, if the latter cube has the corresponding variable x in its non-complemented phase. This follows from Lemma 1.1.

Thus if (the original space representation of) a prime π (in the extended space) contains a minterm μ it follows that if μ has any variable x as its non-complemented literal then π must also have the variable x in its non-complemented phase. In addition if π has any variable x as its complemented literal then μ must also have the variable x in its complemented phase.

It is important to keep in mind that, when we refer to containment, we imply containment in the original space. However, as explained in the previous section, containment in the original space translates to the maximality operator in the extended space (Lemma 1.1). Thus all arguments will be made for representations of primes and minterms in the extended space but the relationship will be referred to as containment, even though it is strictly maximality.

Minterm Dominance

The formula for computing the reduced cover is based on traversing and comparing two minterm BDD's in parallel. It is essentially a means of comparing each minterm with every other minterm in the set, in order to evaluate whether it may be dominated or may co-dominate the other.

Consider two minterms (μ, μ'). If μ' dominates μ, then every prime covering μ' also covers μ. This emerges from the fact that in such a situation a cover for μ' is automatically a cover for μ. Now consider any variable $x_i \ni (\bar{x}_i \supseteq \mu')$, $(x_i \supseteq \mu)$. Since $\bar{x}_i \supseteq \pi) \Rightarrow \overline{(\pi \geq \mu}$ and that contradicts the assumption that μ dominates μ', since we have found a prime covering μ' but not covering μ. It must follow that for all primes π such that $(\pi \geq \mu')$, $(x_i \supseteq \pi)$. The procedures are based on this insight.

The reasoning given above helps give a means to compute the set of dominated minterms α'.

In order to compute the subset of a set of minterms that are covered by a given set of primes, the *MCover* relation is used. The following theorem indicates how to compute this function.

Lemma1.4: The subset of a set of minterms μ that are covered by a given set of primes π is given by

$$MCover(\mu, \pi) = \bar{x} \cdot MCover(\mu_{\bar{x}}, \pi_{\bar{x}} + \pi_x) + x \cdot MCover(\mu_x, \pi_x)$$

Termination Conditions:

1. if $(\mu = 1, \pi = 1)$ then return 1
2. else return 0

Proof: [1] *By Induction on the covering property*

The set of minterms from the set μ which are covered by the set of primes π can be computed as follows.

1. If any variable *x =1* (i.e.$\mu_x \neq 0$) in any minterm belonging to the set, then this minterm can only be covered by cubes which have *x= 1* and cannot be covered by cubes which have *x=0* (From the containment argument on the previous page using Lemma 1.1). Hence minterms with *x=1* can only be contained within primes with *x =1* (i.e. $\pi_x \neq 0$). This gives the second term in the expansion of MCover.

2. If any variable *x=0* in any minterm belonging to the set, then this minterm can only be covered by cubes which have either *x=0 or x =1*. Hence minterms with *x=0* can be covered by primes with either *x=0 or x=1*. This gives the first term in the expansion of MCover.

This formulation is required for the computation of the minterms that are strictly dominated and co-dominated using theorems 1.10 and 1.11.

Theorem 1.10: The set of minterms of a given covering table $C(\mu, \pi)$, which are dominated by some other minterm of the table are given by $\alpha'(\mu, \mu, \pi, 0)$ where the relation α' can be computed by

$$\alpha'(\mu, \mu', \pi, \lambda) =$$

$$\bar{x} \cdot \alpha'(\mu_{\bar{x}}, \mu'_{\bar{x}}, \pi_{\bar{x}} + \pi_x, \lambda_1) + \bar{x} \cdot \alpha'(\mu_{\bar{x}}, \mu'_x, \pi_x, \lambda_2)$$

$$+ x \cdot \alpha'(\mu_x, \mu'_{\bar{x}} \cdot \overline{MCover(\mu'_{\bar{x}}, \pi_{\bar{x}})}, \pi_x, \lambda_3) + x \cdot \alpha'(\mu_x, \mu'_x, \pi_x, \lambda_4)$$

where the λ terms are integers that represent checks to ensure that we are not checking any minterm against itself. If $\lambda = 0$ then $\lambda_1 = \lambda_4 = 0$ and $\lambda_2 = \lambda_3 = 1$, otherwise if $\lambda \neq 0$ then $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = \lambda$.

Termination Conditions:

1. If $\mu = 1, \mu' = 1, \pi = 1, \lambda = 1$ then return 1, else
2. return 0 if $\mu = 1, \mu' = 1, \pi = 1, \lambda \neq 1$
3. return 0 if any bdd tree is a zero(0) BDD

Proof: [1]*By Induction on the dominance property.*

Our formula for dominated minterms minterms is a function of three arguments (μ, μ', π). μ is the set of minterms, μ' is the set of minterms that dominate them and π is the set of primes such that they cover every minterm in the dominating set. Loosely, π is the set of primes that have not been shown to fail to cover the dominating set. The formula for dominated minterms is in the form of a comparison function between two minterms, μ and μ'. Ideally we would like to be able to conclude whether μ is dominated by μ' based on whether their cofactors with respect to some variable in their support dominate each other. There are four such pairs of cofactors which may be compared. These are as follows:

1. (μ_x, μ'_x) : In this case, no statement can be made at this level of the recursion. All the terms in (μ_x, μ'_x) need to be examined further. In addition only primes within π_x potentially cover minterms within μ'_x and hence π_x represents the set of primes which require to be examined at a later stage.

2. $(\mu_{\bar{x}}, \mu'_{\bar{x}})$: No conclusive statement can be made at this level of the recursion. All minterms in $(\mu_{\bar{x}}, \mu'_{\bar{x}})$ need to be examined further. The set of primes which potentially cover $\mu'_{\bar{x}}$ include all primes such that $x \supseteq \pi$ or $\bar{x} \supseteq \pi$. Hence $\pi_x + \pi_{\bar{x}}$ is the set of primes which need to be examined further.

3. $(\mu_{\bar{x}}, \mu'_x)$: In this case too, no statement as to whether μ' dominates μ can as yet be made. All minterms in $(\mu_{\bar{x}}, \mu'_x)$ need to be further examined. The set of primes which may potentially cover μ'_x include all primes such that $x \supseteq \pi$ and hence π_x is the set of primes which need to be examined further.

4. $(\mu_x, \mu'_{\bar{x}})$: In this case it can be definitely concluded that if there exists a prime π such that $\pi_{\bar{x}} \geq \mu'_{\bar{x}}$, then since we know that this prime can never cover μ_x, the minterms in $\mu'_{\bar{x}}$ can never dominate minterms in μ_x. At this stage only those minterms in $\mu'_{\bar{x}}$ which are not covered by any prime in $\pi_{\bar{x}}$ need to be examined further. It can be concluded from the previous statement that π_x represents the set of primes which require to be examined at a later stage.

If $\mu = 1, \mu' = 1, \pi = 1, \lambda = 1$ then we can safely conclude that minterm μ' dominates μ because all primes covering μ' (namely π) cover μ and $\mu \neq \mu'$. However if any one of the (μ, μ', π) BDD's are a Zero(0) BDD we can safely conclude that minterm μ' does not dominate μ

In a similar manner we get the formulation for Co-dominators δ.

<u>Theorem</u> 1.11: The Co-dominating minterms of a given covering table $C(\mu, \pi)$ are given by $\delta(\mu, \mu, \pi, 0)$ where the relation δ can be computed recursively as

$$\delta(\mu, \mu', \pi, \lambda) =$$

$$x \cdot \delta(\mu_x, \mu'_x, \pi_x, \lambda_1) + \bar{x} \cdot \delta(\mu_{\bar{x}}, \mu'_{\bar{x}}, \pi_{\bar{x}} + \pi_x, \lambda_4)$$

$$+ \bar{x} \cdot \delta(\mu_{\bar{x}} \cdot \overline{MCover(\mu_{\bar{x}}, \pi_{\bar{x}})}, \mu'_x, \pi_x, \lambda_2) + x \cdot \delta(\mu_x, \mu'_{\bar{x}} \cdot \overline{MCover(\mu'_{\bar{x}}, \pi_{\bar{x}})}, \pi_x, \lambda_4)$$

where the λ terms represent checks to ensure that we are not checking any minterm against itself. If $\lambda = 0$ then $\lambda_1 = \lambda_4 = 0$ and $\lambda_2 = \lambda_3 = 1$, otherwise if $\lambda \neq 0$ then $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = \lambda$.

The termination conditions in this recursion are identical to the recursion for α'

<u>Proof</u>: [1]*By Induction on the dominance property.*

As in the previous case, there are four cofactors pairs to check.

1. (μ_x, μ'_x) : In this case, no statement can be made at this level of the recursion. All the terms in (μ_x, μ'_x) need to be examined further to check for co-dominance. In addition only primes within π_x potentially cover minterms within μ'_x and $\mu_{\bar{x}}$, hence π_x represents the set of primes which require to be examined at a later stage.

2. $(\mu_{\bar{x}}, \mu'_{\bar{x}})$: No conclusive statement can be made at this level of the recursion. All minterms in $(\mu_{\bar{x}}, \mu'_{\bar{x}})$ need to be examined further. The set of primes which potentially cover $\mu'_{\bar{x}}$ and $\mu_{\bar{x}}$ include all primes such that $x \supseteq \pi$ or $\bar{x} \supseteq \pi$. $\pi_x + \pi_{\bar{x}}$ is the set of primes which need to be examined further.

3. $(\mu_{\bar{x}}, \mu'_x)$: In this case it can be definitely concluded that if there exists a prime π such that $\pi_{\bar{x}} \geq \mu_{\bar{x}}$, then since this prime may never cover any minterm in the minterms in μ'_x, $\mu_{\bar{x}}$ can never Co-dominate minterms in μ'_x. At this stage only those minterms in $\mu_{\bar{x}}$ which are not covered by any prime in $\pi_{\bar{x}}$ need to be examined further. From this we may conclude that the set of primes which may potentially cover μ'_x and $\mu_{\bar{x}}$ include all primes such that $x \supseteq \pi$ and hence π_x is the set of primes which need to be examined further.

4. $(\mu_x, \mu'_{\bar{x}})$: In this case it can be definitely concluded that if there exists a prime π such that $\pi_{\bar{x}} \geq \mu'_{\bar{x}}$, then the minterms in $\mu'_{\bar{x}}$ can never Co-dominate minterms in μ_x. At this stage only those minterms in $\mu'_{\bar{x}}$ which are not covered by any prime in $\pi_{\bar{x}}$ need to be examined further. It can be concluded from the previous statement that π_x represents the set of primes which require to be examined at a later stage.

We do not need to add all the Co-dominating minterms of a table; it suffices to have only one representative Co-dominator of each pair of Co-dominators. One must note that if minterm *a* Co-dominates another minterm *b*, minterm *b* Co-dominates minterm *c* and as a result *a* Co-dominates *c*, then any one of the 3 minterms suffice to represent the set. However since Co-dominance is a pairwise relationship, it is not possible to choose from each pair *(a,b), (b,c), (a,c)*, a minterm to add. Instead however it is always possible to choose a minterm to delete from each pair, based on some tie-breaking criteria; for example choose to remove the minterm with the first complemented variable as per the BDD variable ordering. Thus we could choose minterm *b* from *(a,b)*, minterm *c* from *(b,c)* and it must follow that we choose minterm *c* from *(a,c)* to delete from the set of all Co-dominators. This leaves minterm *a* as the representative Co-dominator.

In order to compute this set of Co-dominators to delete we modify the algorithm for the computation of Co-dominators in order to take into account the *tie-breaker*. To compute the Co-dominators_to_delete δ' we easily incorporate the following changes in theorem. We change the choice of λ as follows: If $\lambda = 0$ then $\lambda_1 = \lambda_4 = 0$ and $\lambda_2 = 1, \lambda_3 = -1$, otherwise if $\lambda \neq 0$ then $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = \lambda$. This results in the choice of one Co-dominator from each pair.

The final set of minterms to retain at each pass of the Quine-McCluskey algorithm include the *strict dominators plus some Co-dominators*. In order to obtain the *strict dominators* we need to compute the set of minterms which are dominated by some other minterm of the set and subtract these from the set of all minterms in the covering table. We also need to add some subset of the Co-dominators to this set of strict dominators to get a reduced cover for the function. The final expression for minterms to keep at each pass of the reduction algorithm is obtained as follows.

$$\mu_{n+1} = \mu_n - \alpha'(\mu_n, \mu_n, \pi_n, 0) + \delta(\mu_n, \mu_n, \pi_n, 0) - \delta'(\mu_n, \mu_n, \pi_n, 0)$$

Prime-dominance

The conditions for Prime dominance can be derived in a similar manner to Minterm dominance. Consider two primes (π, π'). If π is dominated by π', then every minterm covered by π is also covered by π'. Now consider any variable $x_i \ni (\bar{x}_i \supseteq \pi')$, $(x_i \supseteq \pi)$. Since $x_i \supseteq \mu) \Rightarrow \overline{(\pi' \geq \mu}$ and that contradicts our assumption that π is dominated by π',it must follow that for all $(\mu \geq \pi)$, $(\bar{x}_i \supseteq \mu)$. The conditions for prime dominance can be reasoned in a manner analogous to the conditions for minterm dominance.

In order to compute the subset of primes which cover a given set of minterms the following lemma is used. This lemma is used by theorems 1.12 and 1.13 to compute the set of Primes which are dominated and Co-dominated.

Lemma1.5: The subset of a set of primes π that cover a given set of minterms μ is given by

$$PCover(\mu, \pi) = \bar{x} \cdot PCover(\mu_{\bar{x}}, \pi_{\bar{x}}) + x \cdot PCover(\mu_{\bar{x}} + \mu_x, \pi_x)$$

1. if $(\mu = 1, \pi = 1)$ then return 1
2. else return 0

Proof: [1]*By Induction on the covering property in the extended space.*

1. If some prime within the set of primes π has variable *x=0* (i.e. $\pi_{\bar{x}} \neq 0$), then this prime may only cover those minterms which have *x=0*. It can never contain a minterm with *x=1*. Thus primes with

x=0 may potentially cover only minterms with variable *x=0*. This results in the first term in the expansion *PCover.*

2. If some prime has variable *x=1* (i.e $\pi_x \neq 0$), then this prime may cover minterms with either *x=0 or x=1*. This returns the second term in the expansion of PCover.

Theorem 1.12: The set of Primes of a given covering table $C(\mu, \pi)$ that are Dominated by some other prime in the set, are given by $\beta'(\pi, \pi, \mu, 0)$

$\beta'(\pi, \pi', \mu, \lambda) =$

$$\bar{x} \cdot \beta'(\pi_{\bar{x}}, \pi'_{\bar{x}}, \mu_{\bar{x}}, \lambda_1) + \bar{x} \cdot \beta'(\pi_{\bar{x}}, \pi'_x, \mu_x, \lambda_2)$$

$$+ (x \cdot \beta'(\pi_x \cdot \overline{PCover(\mu_x, \pi_x)}, \pi'_{\bar{x}}, \mu_x, \lambda_3) + x \cdot \beta'(\pi_x, \pi'_x, \mu_{\bar{x}} + \mu_x, \lambda_4))$$

where the λ terms represent checks to ensure that we are not checking any prime against itself. If $\lambda = 0$ then $\lambda_1 = \lambda_4 = 0$ and $\lambda_2 = \lambda_3 = 1$, otherwise if $\lambda \neq 0$ then $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = \lambda$.

The termination conditions and the computation of the λ variables are identical to the case of recursion α'

Proof: [1] *By Induction on the Dominance property.*

Similarly we can compute the Co-dominators, Δ as follows.

Theorem 1.13: The set of Prime Co-dominators of a given covering table $C(\mu, \pi)$ are given by $\Delta(\pi, \pi, \mu, 0)$ where

$\Delta(\pi, \pi', \mu, \lambda) =$

$$\bar{x} \cdot \Delta(\pi_{\bar{x}}, \pi'_{\bar{x}}, \mu_{\bar{x}}, \lambda_1) + \bar{x} \cdot \Delta(\pi_{\bar{x}}, \pi'_x \cdot \overline{PCover(\mu_x, \pi'_x)}, \mu_x, \lambda_2)$$

$$+ (x \cdot \Delta(\pi_x, \pi'_x, \mu_{\bar{x}} + \mu_x, \lambda_4) + x \cdot \Delta(\pi_x \cdot \overline{PCover(\mu_x, \pi_x)}, \pi'_{\bar{x}}, \mu_x, \lambda_3))$$

The termination conditions and the computation of the λ variables are identical to the case of recursion α'

Proof: [1] *By Induction on the Dominance property.*

As before λ terms represent checks to ensure that we are not checking any prime against itself. The termination conditions and the computation of the λ variables are identical to the case of recursion α'

The computation of the prime Co-dominators to delete follows along the lines of the minterm analyses. Thus in order to compute the prime Co-dominators to delete $\Delta'(\pi, \pi, \mu, 0)$ we need only modify the computation of λ as follows: If $\lambda = 0$ then $\lambda_1 = \lambda_4 = 0$ and $\lambda_2 = 1, \lambda_3 = -1$, otherwise if $\lambda \neq 0$ then $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = \lambda$.

The set of primes remaining after each pass of the reduction are given as

$$\pi_{n+1} = \pi_n - \beta'(\pi_n, \pi_n, \mu_n, 0) + \Delta(\pi_n, \pi_n, \mu_n, 0) - \Delta'(\pi_n, \pi_n, \mu_n, 0)$$

Non of the above formulations require quantifier operations. As in the case of most bdd traversal based computations hashing is necessary in order to make this method efficient.

Partially Quantifier free formulation for the Dominance Relations

The conditions for Prime and Minterm Dominance can also be made partially quantifier free. The advantage of this approach is that it avoids the excessively large number of recursions that the approach in the previous section entails. On the other hand it bypasses the need to build the intermediate BDD's in the equations for dominance (section 8), which tend to blow up in size.

Consider two minterms (μ, μ'). If μ dominates μ', then every prime covering μ also covers μ'. Now consider any variable $x_i \ni (\bar{x}_i \supseteq \mu)$, $(x_i \supseteq \mu')$. Since $(\bar{x}_i \supseteq \pi) \Rightarrow \overline{(\pi \geq \mu')}$ and that contradicts the assumption that μ' is dominated by μ, it must follow that for all primes π such that $(\pi \geq \mu)$, $(x_i \supseteq \pi)$.

Theorem 1.14: The condition for minterm dominance $\eta(x, z)$ is given by $\eta(\mu(x), \mu'(x), \pi(x))$ where

$\eta(\mu, \mu', \pi) =$

$$\bar{x}\bar{z} \cdot \eta(\mu_{\bar{x}}, \mu'_{\bar{x}}, \pi_{\bar{x}} + \pi_x) + \bar{x}z \cdot \eta(\mu_{\bar{x}} \cdot \overline{Cover(\mu_{\bar{x}}, \pi_{\bar{x}})}, \mu'_x, \pi_x) + x\bar{z} \cdot \eta(\mu_x, \mu'_{\bar{x}}, \pi_x) + xz \cdot \eta(\mu_x, \mu'_x, \pi_x)$$

The termination conditions in this recursion are the following

If $= 1, \mu' = 1, \pi =$ then return 1, else

return 0 if any bdd tree is a zero(0) BDD

Proof: *By Induction on the dominance property in the extended space.*

Our formula for minterm dominance is a function of three arguments (μ, μ', π). μ is the set of minterms,μ' is the set of minterms that dominate them and π is the set of primes such that they cover every minterm in the dominating set. Loosely, π is the set of primes that have not been shown to fail to cover the dominating set. The formula for dominated minterms is in the form of a comparison function between two minterms, μ and μ'. Ideally we would like to be able to conclude whether μ is dominates μ' based on whether their cofactors with respect to some variable in their support dominate each other. There are four such pairs of cofactors which may be compared. These are as follows:

1. (μ_x, μ'_x) : In this case, no statement can be made at this level of the recursion. All the terms in (μ_x, μ'_x) need to be examined further. In addition only primes within π_x potentially cover minterms within μ_x and hence π_x represents the set of primes which require to be examined at a later stage.

2. $(\mu_{\bar{x}}, \mu'_{\bar{x}})$: No conclusive statement can be made at this level of the recursion. All minterms in $(\mu_{\bar{x}}, \mu'_{\bar{x}})$ need to be examined further. The set of primes which potentially cover $\mu_{\bar{x}}$ include all primes such that $x \supseteq \pi$ or $\bar{x} \supseteq \pi$. Hence $\pi_x + \pi_{\bar{x}}$ is the set of primes which need to be examined further.

3. $(\mu_{\bar{x}}, \mu'_x)$: In this case it can be definitely concluded that if there exists a prime π such that $\pi_{\bar{x}} \geq \mu_{\bar{x}}$, then since we know that this prime can never cover μ'_x, hence the minterms in $\mu_{\bar{x}}$ can never dominate minterms in μ'_x. At this stage only those minterms in $\mu_{\bar{x}}$ which are not covered by any prime in $\pi_{\bar{x}}$ need to be examined further. It can be concluded from the previous statement that π_x represents the set of primes which require to be examined at a later stage.

4. $(\mu_x, \mu'_{\bar{x}})$: In this case too, no statement as to whether μ' dominates μ can as yet be made. All minterms in $(\mu_{\bar{x}}, \mu'_x)$ need to be further examined. The set of primes which may potentially cover μ_x include all primes such that $x \supseteq \pi$ and hence π_x is the set of primes which need to be examined further.

If $\mu = 1, \mu' = 1, \pi = 1, \lambda = 1$ then we can safely conclude that minterm μ' dominates μ because all primes covering μ' (namely π) cover μ and $\mu \neq \mu'$. However if any one of the (μ, μ', π) BDD's are a Zero(0) BDD we can safely conclude that minterm μ' does not dominate μ

Consider two primes (π,π'). If π dominates π', then every minterm covered by π' is also covered by π. Now consider any variable $x_i \ni (\bar{x}_i \supseteq \pi)$, $(x_i \supseteq \pi')$. Since $(x_i \supseteq \mu) \Rightarrow \overline{(\pi \geq \mu)}$ and that contradicts our assumption that π dominates π',it must follow that for all $(\mu \geq \pi')$, $(\bar{x}_i \supseteq \mu)$.

Theorem 1.15: The condition for prime dominance $(x, z$ can also be formulated as

$\Upsilon(\pi(x), \pi'(x), \mu(x))$ **where**

$\Upsilon(\pi, \pi', \mu) =$

$$\bar{x}\bar{z} \cdot \Upsilon(\pi_{\bar{x}}, \pi'_{\bar{z}}, \mu_{\bar{x}}) + \bar{x}z \cdot \Upsilon(\pi_{\bar{x}}, \pi'_{z} \cdot \overline{Cover(\mu_x, \pi'_x)}, \mu_{\bar{x}}) + x\bar{z} \cdot \Upsilon(\pi_x, \pi'_{\bar{z}}, \mu_{\bar{x}}) + xz \cdot \Upsilon(\pi_x, \pi'_z, \mu_{\bar{x}} + \mu_x)$$

The termination conditions in this recursion are the following

If $\pi = 1, \pi' = 1$ then return 1, else

return 0 if any bdd tree (π, π') is a zero(0) BDD

<u>Proof:</u> [1] By Induction on the dominance property

Our formula for prime dominance is a function of three arguments (π, π', μ). π is the set of primes, π' is the set of primes that are dominated by them and μ is the set of minterms such that they are covered by every prime in the dominating set. The formula for prime dominance is in the form of a comparison function between primes, π and π'. Ideally we would like to be able to conclude whether π is dominates π' based on whether their cofactors with respect to some variable in their support dominate each other. There are four such pairs of cofactors which may be compared. These are as follows:

1. (π_x, π'_x) : In this case, no statement can be made at this level of the recursion. All the terms in (π_x, π'_x) need to be examined further. The set of minterms which are potentially covered by π include all minterms such that $x \supseteq \mu$ or $\bar{x} \supseteq \mu$. Hence $\mu_x + \mu_{\bar{x}}$ is the set of minterms which need to be examined further

2. $(\pi_{\bar{x}}, \pi'_{\bar{x}})$: No conclusive statement can be made at this level of the recursion. All primes in $(\pi_{\bar{x}}, \pi'_{\bar{x}})$ need to be examined further. In addition the set of minterms which are potentially covered by π include minterms such that $\bar{x} \supseteq \mu$; i.e. we examine $\mu_{\bar{x}}$

3. $(\pi_{\bar{x}}, \pi'_x)$: In this case, if there exists a minterm μ such that $\bar{x} \supseteq \mu$, then we definitely conclude that π does not dominate π'. Thus we only examine further those primes of π' which do not satisfy this condition. We examine minterms such that $\bar{x} \supseteq \mu$, i.e. minterms $\mu_{\bar{x}}$.

4. $(\pi_x, \pi'_{\bar{x}})$: In this case we can make no conclusive statement at this stage and hence need to examine all primes at a successive level in the recursion. All minterms covered by π are included in minterms such that $\bar{x} \supseteq \mu$, i.e. we examine minterms in the set $\mu_{\bar{x}}$.

If $\pi = 1,\ \ \pi' = 1, \lambda = 1$ then we can safely conclude that prime π dominates π' because there does not exist a minterm covered by π' but not by π. However if any one of the π or π'BDD's are a Zero(0) BDD we can safely conclude that prime π does not dominate π'.

Having computed these we can substitute the expressions back in the equations for Quine-McCluskey reduction in order to solve the covering problem. The method remains the essentially same.

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.

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

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

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

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

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

$$\phi_{output} = \prod_j \bar{X}_{oj}$$

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

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

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

$$\upsilon_{output} = \sum_j X_{oj} \prod_{i \neq j} \bar{X}_{oi}$$

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 / max points 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. 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 make this calculation more efficient a *memoization* is used. 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[2].*

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 do prime computation we use the equation

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

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 expressions given in section 7. 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)

The Quantifier free dominance relations are computed using the formulae given in the relevant sections. This formulation is recursive and based on computing the result for the left and right branches of the input BDD's first. Again a hash table is used to hash the value of the result BDD with the node as key.

The pseudo code for Minterms dominated at a single pass of the Quine-McCluskey algorithm is given by the following. The other terms can similarly derived.

Minterms_dominated μ, μ', π, λ*()*

if look_up((μ, μ', π, λ*),value) return value*

else if (terminal_value((μ, μ', π, λ*),value)) return value*

else

  x_{ij} *= top_variable((*μ, μ', π*))*

  md_nx = bdd_or(Minterms_dominated ($\mu_{\bar{x}}, \mu'_{\bar{x}}, \pi_{\bar{x}} + \pi_x, \lambda_1$*),Minterms_dominated (*$\mu_{\bar{x}}, \mu'_x, \pi_x, \lambda_2$*))*

md_x = bdd_or(Minterms_dominated ($\mu_x, \mu'_{\bar{x}} \cdot \overline{MCover(\mu'_{\bar{x}}, \pi_{\bar{x}})}, \pi_x, \lambda_3$*),Minterms_dominated (*$\mu_x, \mu'_x, \pi_x, \lambda_4$*))*

  return(bdd_ite(x_{ij}*, md_x, md_nx))*

We traverse the BDD's for primes and minterms in parallel node by node. At each point we first check if the dominance relation for a set of nodes 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.

Results

We ran our program on 115 examples that Espresso could solve and 18 examples Espresso could neither compute the primes for, nor solve. These examples were taken from the 134 PLA test benchmark examples reported in [8]. We have reported all the results for future reference. All experiments were run on a DEC 5000 workstation. The results are summarized in the conclusions section.

Using the implicit method described in this thesis, we have been able to solve 5 of the 18 hard Espresso problems tried; namely the examples *misj, misg, mish, ibm and ex4*. We were 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, x7dn, and xparc*. We are also able to form the cyclic core of most of the 134 PLA test benchmark circuits. Using a branch and bound procedure to solve the remaining problem creates no problem as the cyclic core remaining at this stage is in general extremely compact.

Table1. reports the number of inputs and outputs, the number of primes and minterms before reduction, the number of primes and minterms after reduction, the number of primes in the minimum cover generated by *Espresso* and finally the cpu-time used for this computation, using the two best ordering heuristics we implemented. 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. The partial quantifier dominance technique is used for computing the results in table1.

TABLE 1. Primes and Minterms before and after reduction

Name	Inputs	Outputs	Primes	Minterms	Primes After Reduction (Cyclic Core)	Minterms after Reduction	Primes Espresso Minimum	Time (in sec) using old Order	Time using Aziz's Order
al2	16	47	9179	191296	66	66	66	22.3	33.5
alcom	15	38	4657	88064	40	40	40	11.3	10.7
alu1	12	8	780	15872	19	19	19	16.8	64.4
alu2	10	8	434	3249	71	71	68	238	161
alu3	10	8	540	3903	64	64	64	273.7	299
amd	14	24	457	35072	81	79	66	152	184.6
apla	10	12	201	157	25	25	25	17.1	17.7
b10	15	11	938	72912	103	103	100	2365	2669
b11	8	31	44	836	27	27	27	11.1	17.9
b12	15	9	1490	163072	52	59	41	82.0	198.4

Name	Inputs	Outputs	Primes	Minterms	Primes After Reduction (Cyclic Core)	Minterms after Reduction	Primes Espresso Minimum	Time (in sec) using old Order	Time using Aziz's Order
b2	16	17	928	328488	104	104	104	1415.4	*
b4	32	20	6455	49942364160	54	54	54	3558.7	*
b7	8	31	44	836	27	27	27	11.6	16.8
b9	16	5	3002	133704	119	119	119	328.3	361.9
bca	26	46	305	69837824	180	67059892	180	350.0	258.8
bcb	26	39	255	69524480	155	67106972	155	140.7	146.5
bcc	26	45	237	203738112	143	20126216	137	214.0	205.3
bcd	26	38	172	202973184	117	201273456	117	145.5	109.6
br1	12	8	29	114	19	19	19	4.9	2.9
br2	12	8	27	125	13	13	13	2.8	2.7
chkn	29	7	671	788036864	143	143	140	489.6	556.9
clpl	11	5	143	6713	20	20	20	0.7	2.2
dc1	4	7	22	47	9	9	9	1.2	1.2
dc2	8	7	173	443	39	39	39	14.3	16.1
dekoder	4	7	26	49	12	12	9	1.5	1.6
dk17	10	11	111	61	18	18	18	9.9	9.1
dk27	9	9	82	20	14	14	10	3.0	3.4
dk48	15	17	157	42	21	21	21	22.2	23.1
ex4☛	128	28	1.83487E14	computed	289	289	☛	570.2	640.1
ex7	16	5	3002	133704	119	119	119	328.3	331.1
exep	29	63	219	211536128	119	119	108	1294.5	228.9
exp	8	18	238	297	56	56	56	54.3	55.3
exps	8	38	852	1847	209	426	132	1261.3	1241.2
gary	15	11	706	84196	110	110	110	478.3	484.7
ibm ☛	48	17	1057948736	1.5523729E15	173	173	☛	3291.3	*
in0	15	11	706	84196	110	110	107	563.7	554.1
in1	16	17	928	328488	104	104	104	1379.8	4500.8
in2	19	10	666	686336	155	159	134	299.4	499.5
in3	35	29	1114	1.748525E11	74	74	74		8099.9
in5	24	14	1067	24912896	62	62	62	1235.2	2246.0
in6	33	23	6174	4.9950752E10	54	54	54	734.6	1432.7
in7	26	10	2112	220769280	54	54	54	464.9	158.4
inc	7	9	124	281	29	29	29	17.5	17.1
lin.rom	7	36	1087	2306	594	560	128	2464.9	3420
luc	8	27	190	2198	26	26	26	61.5	73.4
m1	6	12	59	218	19	19	19	6.5	6.1
m2	8	16	243	831	47	47	47	70.5	71.5

Name	Inputs	Outputs	Primes	Minterms	Primes After Reduction (Cyclic Core)	Minterms after Reduction	Primes Espresso Minimum	Time (in sec) using old Order	Time using Aziz's Order
m3	8	16	344	1105	85	82	62	148.6	122.1
m4	8	16	670	2134	246	240	101	413.6	446.6
mark1	20	31	208	7340912	126	1646	19		5514.4
max1024	10	6	1278	3232	954	966	261	1046.5	1179.0
max128	7	24	469	1616	131	126	78	336.5	450.5
max46	9	1	49	62	46	46	46	1.8	1.9
max512	9	6	535	1616	297	304	133	315.5	279.6
misg ☛	56	23	6499491840	1.054609E18	69	69	☛	66.8	164.2
mish ☛	94	43	1.124375E15	4.4149420E29	82	82	☛	912.1	986.0
misj ☛	35	14	139103	2.561545E11	35	35	☛	13.7	14.4
mp2d	14	14	469	118544	180	290	30	74.2	96.2
newapla	12	10	113	10421	17	17	17	6.1	12.5
newapla1	12	7	31	380	10	10	10	1.1	1.1
newapla2	6	7	7	7	7	7	7	0.2	0.2
newbyte	5	8	8	8	8	8	8	0.2	0.2
newcond	11	2	72	704	31	31	31	7.4	7.4
newcpla1	9	16	170	1317	40	40	38	26.6	25.3
newcpla2	7	10	38	282	19	19	19	4.0	4.2
newcwp	4	5	23	42	11	11	11	0.7	0.8
newill	9	1	11	284	11	11	8	0.5	0.5
newtag	8	1	8	234	8	8	8	0.2	0.3
newtpla	15	5	40	4484	23	23	23	2.8	3.4
newtpla1	10	2	6	12	4	4	4	0.5	0.3
newtpla2	10	4	23	608	9	9	9	1.5	1.4
nexcpla1	9	23	191	3506	39	39	39	29.1	40.5
opa	17	69	477	732072	150	150	77	791.4	1048
p82	5	14	48	81	21	21	21	5.7	6.0
pope.rom	6	48	593	1614	130	121	59	1202	1232
rd53	5	3	51	42	31	31	31	0.5	0.5
rd73	7	3	211	192	127	127	127	1.8	1.6
rd84	8	4	633	411	255	255	255	2.6	2.6
risc	8	31	46	844	28	28	28	10.8	11.1
ryy6	16	1	112	39420	112	112	112	0.5	0.5
sex	9	14	99	1848	21	21	21	13.1	18.6
sqn	7	3	75	144	38	38	38	6.6	7.1
t1	21	23	15135	13956096	226	261	100	2073.4	2073.4
t2	17	16	233	167920	52	52	52	52.4	95.5

Name	Inputs	Outputs	Primes	Minterms	Primes After Reduction (Cyclic Core)	Minterms after Reduction	Primes Espresso Minimum	Time (in sec) using old Order	Time using Aziz's Order
t3	12	8	42	4096	33	33	33	8.1	7.5
t4	12	8	174	982	16	16	16	24.2	30.5
tms	8	16	162	790	34	34	30	29.5	28.4
vg2	25	8	1188	61570752	110	110	110	55.8	239.2
vtx1	27	6	1220	133035072	110	110	110	227	172.9
wim	4	7	25	51	12	12	9	1.3	2.0
x1dn	27	6	1220	133035072	110	110	110	229.4	285.5
x6dn	39	5	916	667727953920	84	84	81	2260.4	2349.3
x9dn	27	7	1272	133041984	120	120	120	193.4	371.6

☚ refers to one of espresso's 20 hard problems.

b. "computed" implies the minterms could not be counted as they exceeded the float size limitation.

As reported earlier the problem size is very much a function of the ordering technique used. The next table (table 2.) reports the sizes of the BDD's for the extended space representation, the Prime BDD and the minterm BDD, as well as the number of inputs and Outputs. For any BDD based method it is these numbers which are most representative of the complexity of the problem.

Malik's "level" heuristics [14] 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 [11] for ordering combinatorial circuits. All *x,u and z* variables were again interleaved. The Last ordering technique tried out was an adaptation of the technique described in [12]. This method is based on choosing an ordering such that the partition induced reduces communication complexity and is referred to as Aziz's Ordering[12].

TABLE 2. Sizes of Extended Space, Prime and Minterm BDD's

Name	Inputs	Outputs	Nodes in extended BDD (F+D)	Nodes in extended BDD (F)	Size of Prime Bdd	Size of Min-term Bdd
al2	16	47	1432	1432	919	877
alcom	15	38	733	733	619	568
alu1	12	8	543	543	742	509
alu2	10	8	3345	6035	1761	1291
alu3	10	8	4480	6745	2057	1544
amd	14	24	4272	4272	3288	1171
apla	10	12	791	548	645	311
b10	15	11	25678	18093	3693	1589
b11	8	31	663	705	616	489
b12	15	9	777	777	969	513
b2	16	17	12416	12416	11050	2256
b3	32	20	31459	25476	9890	3415
b4	33	23	64474	39404	10601	39404
b7	8	31	663	705	616	489
b9	16	5	2369	2369	1756	796
bc0	26	11	55242	55242	22388	2712
bca	26	46	17276	17893	5541	4128
bcb	26	39	9698	9597	3572	2802
bcc	26	45	10282	9114	4348	3202
bcd	26	38	11680	9277	3192	2684
br1	12	8	632	632	398	339
br2	12	8	269	269	271	242
chkn	29	7	6866	6966	2862	1583
clpl	11	5	41	41	70	89
cps	24	109	47524	47524	31111	9530
dc1	4	7	119	119	130	82
dc2	8	7	536	536	484	348
dekoder	4	7	90	118	145	86
dk17	10	11	594	500	527	238
dk27	9	9	217	226	304	156
dk48	15	17	766	785	821	446
ex1010☛[a]	10	10	63490	12693	40559	2620
ex4☛	128	28	3518	3518	2772	2495
ex5	8	63	61216	61216	60118	2847
ex7	16	5	2369	2369	1756	796
exep	29	63	153884	153884	6016	11128

Name	Inputs	Outputs	Nodes in extended BDD (F+D)	Nodes in extended BDD (F)	Size of Prime Bdd	Size of Min-term Bdd
exp	8	18	2416	1753	1594	677
exps	8	38	12220	9870	7278	2887
gary	15	11	7503	7503	3840	1647
ibm☚	48	17	8101	8101	16053	4937
in0	15	11	4690	4690	4304	1711
in1	16	17	12416	12416	11050	2256
in2	19	10	4225	4225	3698	1912
in3	35	29	12133	12133	10441	3699
in4	32	20	21071	21071	9763	3580
in5	24	14	11121	11121	6772	2905
in6	33	23	6687	6687	4237	1675
in7	26	10	6217	6217	4096	1402
inc	7	9	657	553	675	277
intb	15	7	50404	50404	10845	3042
jbp☚	36	57	706942	706942	78955	26237
lin.rom	7	36	15894	15894	11070	1987
luc	8	27	1920	1920	2045	782
m1	6	12	380	380	339	196
m2	8	16	1257	1257	1430	585
m3	8	16	2456	2456	1694	686
m4	8	16	3419	3419	4139	1181
mark1	20	31	1441	984	1872	775
max1024	10	6	4430	4430	3484	1324
max128	7	24	3549	3549	3796	1034
max46	9	1	398	398	202	199
max512	9	6	2448	2448	2090	840
misg☚	56	23	432	432	784	930
mish☚	94	43	13749	13749	8808	6577
misj☚	35	14	364	364	503	605
mp2d	14	14	441	441	610	326
newapla	12	10	243	243	302	235
newapla1	12	7	98	98	103	109
newapla2	6	7	63	63	60	60
newbyte	5	8	79	79	70	70
newcond	11	2	348	348	246	188
newcpla1	9	16	1198	1198	777	462
newcpla2	7	10	301	301	282	197
newcwp	4	5	51	51	66	56
newill	8	1	58	58	56	55

Name	Inputs	Outputs	Nodes in extended BDD (F+D)	Nodes in extended BDD (F)	Size of Prime Bdd	Size of Min-term Bdd
newtag	8	1	23	23	35	45
newtpla	15	5	342	342	262	210
newtpla1	10	2	126	126	94	80
newtpla2	10	4	306	306	153	118
newxcpla1	9	23	989	989	883	407
opa	17	69	10784	10784	9285	4129
p82	5	14	639	639	438	292
pope.rom	6	48	6890	6890	7551	1573
prom1	9	40	113079	113079	49680	10192
prom2	9	21	33929	33929	20309	4869
rd53	5	3	80	80	74	55
rd73	7	3	161	161	151	96
rd84	8	4	247	247	228	125
risc	8	31	723	723	646	506
ryy6	16	1	58	58	66	80
sex	9	14	560	560	623	404
shift☜	19	16	7548	7548	22740	6808
signet☜	39	8	933361	933361	228543	57093
soar.pla☜	83	94	59478	59478	37854	7702
spla	16	23	38136	38345	19494	11319
sqn	7	3	386	386	297	195
t1	21	23	16037	16037	11352	2641
t2	17	16	1647	1307	1419	969
t3	12	8	655	655	430	280
t4	12	8	598	699	740	386
ti☜	47	72	194965	194965	79477	24612
tms	8	16	860	860	1083	407
ts10☜	22	16	18757	18757	46031	46139
vg2	25	8	1079	1079	582	484
vtx1	27	6	4098	4098	1398	883
wim	4	7	83	92	129	80
x1dn	27	6	4098	4098	1398	883
x2dn☜	82	56	23889	23889	18843	7885
x6dn	39	5	2184	2184	5219	2717
x7dn☜	66	15	1084457	1084457		
x9dn	27	7	4214	4214	1410	888
xparc☜	41	73	475670	475670	133084	36437

a. ☛ refers to one of espresso's 20 hard problems.
b. "computed" implies the minterms could not be counted as they exceeded the float size limitation.

For completeness we also report the next table. This table (table 3.)gives the number of Primes and Minterm computed for each example, as well as the time taken for their computation.

TABLE 3. Primes and Minterms

Name	Primes	Minterms	Time to compute
al2	9179	191296	2.7
alcom	4657	88064	1.6
alu1	780	15872	0.6
alu2	434	7422	3.5
alu3	540	3903	3.7
amd	457	35072	5.1
apla	201	157	1.8
b10	938	72912	20.5
b11	44	836	1.9
b12	1490	163072	0.9
b2	928	328488	18.3
b3	3056	1.3076E10	54.3
b4	6455	4.9942E10	78.7
b7	44	836	1.9
b9	3002	133704	1.8
bc0	6596	284933120	112.6
bca	305	2778112	61.3
bcb	255	2417664	31.9
bcc	237	2477056	88.2
bcd	172	1699840	31.1
br1	29	114	0.6
br2	27	125	0.4
bw	108	281	7.6
chkn	671	788036864	5.3
clpl	143	6713	0.1
cps	2487	124362704	69.8
dc1	22	47	0.1
dc2	173	442	0.5
dekoder	26	49	0.3
dk17	111	61	1.5

Name	Primes	Minterms	Time to compute
dk27	82	20	0.8
dk48	157	42	5.0
duke2	1044	8464768	20.9
e64	65	36893488E20	11.3
ex1010 ☚	25888	1471	207.0
ex4 ☚	1.8348E14	computed	10.3
ex5	2532	7620	108.0
ex7	3002	133704	1.8
exep	219	211536128	180.4
exp	238	297	5.5
exps	852	1623	32.6
gary	706	84196	5.9
ibm ☚	1047948736	1.5523729E15	17.5
in0	706	84196	5.5
in1	928	328488	18.2
in2	666	686336	4.9
in3	1114	1.7485E11	23.2
in4	3076	1.3295E10	28.6
in5	1067	24912896	11.9
in6	6174	4.9950E10	7.0
in7	2112	220769280	5.1
inc	124	281	1.1
intb	6522	101720	28.7
jbp ☚	2496809	8.0095268E11	888
lin.rom	1087	2306	21.2
luc	190	2198	2.5
m1	59	218	0.4
m2	243	831	1.5
m3	344	1105	2.3
m4	670	2134	4.4
mark1	208	2098128	15.8
max1024	1278	3232	5.1
max128	469	1616	4.1
max46	49	62	0.4
max512	535	1616	2.3
misg ☚	6499491840	1.054609E18	1.9
misj ☚	139103	2.561545E11	0.8
mish ☚	1.1243753E15	4.1494202E29	17.5
mp2d	469	118544	1.3
newapla	113	10421	0.3

Name	Primes	Minterms	Time to compute
newapla1	31	380	0.2
newapla2	7	7	0.1
newbyte	8	8	0.1
newcond	72	704	0.3
newcpla1	170	1317	1.0
newcpla2	38	282	0.3
newcwp	23	42	0.1
newill	11	142	0.1
newtag	8	234	0.0
newtpla	40	4484	0.3
newtpla1	6	12	0.1
newtpla2	23	608	0.3
newxcpla1	191	3506	1.2
opa	477	732072	23.4
p82	48	81	0.5
pope.rom	593	1614	12.3
prom1	9326	8306	170.6
prom2	2635	3027	39.1
rd53	51	42	0.2
rd73	211	192	0.7
rd84	633	411	1.3
risc	46	844	1.2
ryy6	112	19710	0.1
sao2	184	747	1.2
seq	7457	9.8390465E12	983.8
sex	99	1848	0.6
signet ☜	78735	1.83009529E12	301.1
shift ☜	165133	4194304	35.4
soar.pla ☜	3.3047729E14	1.7458651E26	239.6
sqn	75	144	0.3
spla	4972	122736	101.0
square5	71	85	0.7
t1	15135	13956096	23.8
t2	233	167920	4.5
t3	42	4096	0.5
t4	174	982	7.6
t481	481	42016	2.2
table3	539	11467	50.8
table5	462	119523	77.3
ti ☜	836287	4.136440E14	455.4

Name	Primes	Minterms	Time to compute
tms	162	790	1.2
ts10 ☛	524280	4194304	49.1
vg2	1188	61570752	1.3
vtx1	1220	133035072	3.2
wim	25	51	0.3
x1dn	1220	133035072	3.3
x2dn ☛	1.1488762E16	8.849739E25	53.2
x6dn	916	6.6772795E11	6.5
x9dn	1272	133041984	3.8
xor5	16	16	0.1
xparc ☛	15039	1.0865220E13	584.3

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

b. "computed" implies the minterms could not be counted as they exceeded the float size limitation.

Our method compares very favorably to Espresso [1]; we are able to solve problems which Espresso cannot. These results have been reported in the results table. In addition we are also able to solve nearly all problems Espresso can solve, some of them with times which are faster than Espresso's reported times.

In general the Signature Cube methods [13] tend to be faster than our methods as they do not compute all the primes and minterms. There are, however examples where our methods tend to be faster. In addition signature cube methods can never list all the primes of a function, whereas our application can do so. It is important to note that Espresso-Signature solves many examples that we cannot. Table 2. shows a comparison of times reported by these programs and ours, on a few representative examples. Time-outs were set to 5000 cpu seconds for all runs of our program.

TABLE 4. Comparison of the times on Our program, Espresso and Espresso-Signature

Name	Time (in sec) using old Order	Time using Aziz's Order	Time using Espresso	Time using Espresso Signature
al2	22.3	33.5	349	324.8
alcom	11.3	10.7	97	160.09
b12	82.0	198.4	32	8.16
b9	328.3	361.9	27	7.93
bca	350.0	258.8	267	308.62

TABLE 4. Comparison of the times on Our program, Espresso and Espresso-Signature

Name	Time (in sec) using old Order	Time using Aziz's Order	Time using Espresso	Time using Espresso Signature
bcb	140.7	146.5	79	69.07
bcc	214.0	205.3	168	123
bcd	145.5	109.6	56	49.2
ex4☚	570.2	640.1	☚	163.2
ex7	328.3	331.1	31	7.88
ibm ☚	3291.3	*	☚	1.6
misg ☚	66.8	164.2	☚	13.9
mish ☚	912.1	986.0	☚	49.2
misj ☚	13.7	14.4	☚	2.1
mp2d	74.2	96.2	16	27

☚ refers to one of espresso's 20 hard problems.

.

We found that the quantifier method worked well in some examples, giving an answer equal to the minimum obtained from Espresso-exact, however in some of larger examples it failed at the quantifiers. The quantifier free expressions had very poor results in terms of cpu. time required. Their major drawback seems to be the presence of too many recursive calls. The best results were given by the partial quantifier methods. Table 5. reports the number of primes and minterms before and after reduction. It also reports the number of recursive calls to the dominance functions in the quantifier free method. Recall that the quantifier free methods are formulated as recursive BDD functions. In addition it reports the total number of applications (passes) of the QM procedure and the times used by the quantifier free and quantifier only methods for the corresponding examples. The results for the partial quantifier method have already been reported in table 1. Recall that we had stated that the best results were given by the partial-quantifier technique.

TABLE 5. Comparison of Full Quantifier and Quantifier free forms

Name	Primes	Minterms	Primes after Reduction	Minterms after Reduction	Max number of recursions of fns/pass	Total number of passes	Time using Quantifier free formulation (in sec)	Time using formulation with quantifier (in sec)
br1	29	114	19	19	12079	3	8765	87.2
br2	27	125	13	13	13825	5	12220.	58.4
con1	24	156	9	9	13462	4	1344.	4.0
dc1	22	47	13	13	1723	5	500	5.9
dekoder	26	49	12	12	2488	4	631	5.7

Conclusions

In conclusion we have designed and implemented a method for implicit bdd based two level logic minimization. We have called this program *Implicit_mini.* This program (*Implicit_mini*) was run on 115 of the 117 PLA test examples *Espresso* could solve. We were able to build the primes and minterms for all 115 examples and form the cyclic core (and solve) 93 of these. In addition we ran *implicit_mini* on 18 examples that *Espresso* could neither form the primes for, nor solve and were able to compute the primes (and minterms) for 14 of these and the cyclic core for 5 of these examples. As a result, we have been able to solve 5 of the 18 hard *Espresso* problems tried; namely the examples *misj, misg, mish, ibm and ex4* and build the prime and minterm BDD's for *ex1010, ibm, jbp, misg, misj, mish, shift,soar.pla, ti,ts10, x2dn, x7dn, and xparc*. These are referenced in the tables by the ☚ symbol.

We have, as a course of our research experimented with various bdd-ordering heuristics and evaluated their performance. We have arrived at a set of efficient heuristics, which give the best results (small BDD sizes) for combinational multiple output functions. We have experimented with removal of quantifiers from logical formalisms as part of this research. This removal of quantifiers has been shown to have very impressive results on the speed of BDD computations. We have also derived totally quantifier free expressions for dominance relations. Though this result is not important in itself, it may provide a starting point for the development of quantifier free methods for BDD's.

It is also important to keep in mind while comparing our results with those of *Espresso* that there are different criterion for evaluating the complexity of an example when solved by implicit vs. explicit techniques. Those problems that Espresso finds difficult do not necessarily form large BDD's and conversely problems that form large BDD's may be solved by Espresso with ease. In a sense implicit and explicit methods are symbiotic and neither is complete in itself.

A similar implicit technique was developed and implemented by O. Coudert and J. Madre at Bull research. The essential difference in the two methods lies in their choice of extended space representation and the various bdd minimization methods employed by them. For the former, it can be shown that there is a direct one to one correspondence between the extended space of [6] and our extended space. However the use of suppressed-zero BDD's, which is a technique for reducing BDD sizes, in [6], might account for the differences in our results. In general the BDD's generated by the Bull methods are consistently smaller than our BDD's for the same example.

However our method are very easily extendable to multi-valued minimization. Our extended space is identical to the so called "positional" space used for multi-valued function representation. This makes its easy to apply to vast collection of already available methods which use the positional space representation.

In conclusion we have presented *Implicit_mini*; an efficient BDD based two-level logic minimizer.

References

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

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

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

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

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

[6] J. Madre, O. Coudert, *Implicit and Incremental computation of primes and essential primes*, IEEE Design automation conference 1991.

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

[8] R. Rudell, *Logic synthesis for VLSI design,* ERL Tech Report UCB/ERL M89/49, U. C. Berkeley 1989

[9] G. Swamy, R.K. Brayton, P. Mcgeer, *A Fully implicit Quine-McCluskey procedure using BDD's*, ERL Tech report UCB/ERL M92/127, U.C. Berkeley 1992.

[10] G. Swamy, R.K. Brayton, P. Mcgeer, *A Fully implicit Quine-McCluskey procedure using BDD's*, International Workshop on Logic Synthesis, 1993.

[11] H. Touati, H. Savoj, B. Lin, *Implicit state enumeration of finite state machines using BDD's*, IEEE International Conference on Computer Aided Design 1990.

[12] Adnan Aziz, Serdar Tasiran, Robert K. Brayton, *BDD Variable Ordering for Interacting FSM's*, ERL Tech Report UCB/ERL M93/71,U.C. Berkeley 1993.

[13] Jagesh Sanghavi, A.L. Sangiovanni-Vincentelli, R Brayton, Espresso- Signature: *A new exact minimizer for logic functions*, IEEE Design automation conference 1993.

[14] S. Malik, A.R. Wang, R.K. Brayton and A. Sangiovanni-Vincentelli, *Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment*, IEEE International Conference on Computer Aided Design 1988.

Acknowledgments

To begin with I'd like to thank my advisor Professor Robert K. Brayton for being so patient and understanding and for giving me direction, guidance and ideas when I asked for them. I'd like to give an equally large thank you to Dr. Patrick McGeer for his mentorship, guidance and for all the ideas. I'd like to thank Professor Sangiovanni for his insightful comments on my draft and for patiently correcting my report. A very big thank you to all the very kind members of the CAD group who patiently clarified my doubts and taught me so much: Adnan Aziz, Alex Saldanha, Ellen Sentovich, K.J. Singh, Narendra Shenoy, Rajeev Murgai, Tom Shiple, Tim Kam, Vigyan Singhal and Yosinori Watanabe. I'd also like to thank Alberto Reyes, Carol Wawrukiewicz, Chris Lennard, Desmond Kirkpatrick,.Eric Felt, Jagesh Sanghavi, Heather Harkness and Lisa Guerra for the interesting discussions and for weathering the prelims together. A special thank you to my office-mate Wendell Baker for not only clarifying the engineering questions but also for his wonderfully sardonic and witty comments on life. To the long suffering Flora, Elise, Heather (gradeecs!) and Genevieve, thank you, for fixing everything when I walked in distraught about some administrative detail. To Brad, thank you, for fixing all the real and imaginary complaints I reported about the system. Finally I'd like to thank Sheila Humphreys for giving me all the encouragement I needed to get through my first year in Grad school.

There are many more thanks I'd like to add but I should stop here so that I have something new to add in my doctoral thesis.