Symbolic Prime Generation for Multiple-Valued Functions

Bill Lin1,* Olivier Coudert2 Jean Christophe Madre2

1Department of EECS, University of California, Berkeley, CA 94720
2Bull Corporate Research Center, Rue Jean Jaurès, 78340 Les Clayes-sous-bois, France

Abstract

The minimization of multiple-valued functions has many important applications in logic synthesis. In particular, the problems of two-level multiple-output minimization, symbolic input encoding, and state assignment rely on efficient solutions to the multiple-valued minimization problem. Previously, exact solution to this problem can only be afforded for moderate size problems. A fundamental bottleneck in existing state-of-the-art exact minimizers is the inability to efficiently generate and represent all prime implicants for large functions. This paper presents new techniques based on the implicit representation and generation of primes and essential primes that can tackle multiple-valued functions with sets of primes several orders of magnitude larger than existing methods. Functions with over $10^{10}$ primes have been successfully generated using our proposed method.

1 Introduction

Two-level logic minimization is a key problem that arises in many contexts in the domain of logic synthesis. For over 20 years, the subject has been under intensive investigation. Initial research was directed towards developing techniques for minimizing binary-valued logic functions [1]. These techniques have been generalized for the minimization of multiple-valued (MV) functions [6,5], in which variables can assume more than two discrete values. More specifically, a multiple-valued function $f : D \times B^m$ is a function mapping the set $D$ to the set $B^m$. The output set $B^m$ is the $m$ dimensional Boolean space and the input set $D$ is the Cartesian product of $n$ sets $D_1 \times \cdots \times D_n$, where $D_i$ is a set of $P_i$ values, $D_i = \{0,1,\ldots,P_i - 1\}$, and $P_i$ is a positive integer. Binary-valued input variables are a special case of multiple-valued variables. The significance of the multiple-valued minimization problem is in its wide applications in such areas as PLA optimization, multi-level logic synthesis symbolic input encoding, and state assignment. Multiple-valued minimization can be used even in the context of minimizing multiple-output binary-valued functions. Sasao [6] showed that the minimization of multiple-output functions can be reduced to an equivalent single-output minimization problem with an added multiple-valued variable. Thus, binary-valued, multiple-valued, and multiple-output functions can be treated uniformly.

Heuristic minimization techniques have been developed for minimizing both binary-valued and multi-valued functions [1,5]. These techniques have been successful in tackling relatively large size problems. Exact minimization techniques, on the other hand, have only been affordable in relatively moderate size problems [5]. A fundamental bottleneck in existing state-of-the-art minimizers is their inability to efficiently generate and represent all prime implicants for large functions. This limitation is due to the fact that a function can have in the worst case exponential number of prime implicants with respect to the number of variables. Thus, exact minimization techniques that require the explicit generation and representation of primes are inherently limited to functions with relatively limited number of variables and primes [1,5].

In this paper, we present new techniques based on the implicit representation and generation of primes for multiple-valued functions. Solving this problem is fundamental in developing more powerful two-level minimization algorithms. Our proposed techniques can generate and represent all primes for functions with prime sets several orders of magnitude larger than existing methods. The key idea that makes this computation possible is the symbolic representation of multiple-valued cubes in a characteristic function form called the characteristic-cube function. This symbolic representation can be efficiently denoted using a binary decision diagrams (BDD's) [2], which is known to be a very compact representation for Boolean formulae. Since there is no direct correspondence between the number of elements in a characteristic function and the size of the BDD representation that denotes it, very large sets of primes may be captured symbolically using the characteristic-cube function representation. Using this representation, we have developed a symbolic prime generation procedure based on implicit set operations. The proposed method requires only basic Boolean operators that are available in any BDD package. For a 32-bit adder, we are able to symbolically generate the complete set of over $10^{10}$ primes in only 8.5 seconds of CPU time. The characteristic-cube function that denotes this set has less than 8500 BDD nodes.

The rest of this paper is organized as follows. In Section 2, basic definitions and terminology are provided. In Section 3, we describe our symbolic representation for multiple-valued cubes and covers. In our approach, both binary-valued and multiple-valued variables can be treated uniformly. In Section 4, we present a simple but very efficient symbolic prime generation method based on the characteristic-cube representation for both completely specified and incompletely specified multiple-valued functions. In Section 5, the problem of extracting essential primes is considered. In Section 6, strategies for handling multiple-output functions are described. In Section 7, experimental results are presented on a range of benchmarks. Finally, conclusions are drawn in Section 8.

2 Definitions and Notations

We use the standard definitions and notations widely used in the areas of logic synthesis and verification. Only a subset of these will be summarized for the sake of brevity.

2.1 Multiple-Valued Logic

A multiple-valued function $f$ is a mapping $f : D \rightarrow B^m$. $D = D_1 \times \cdots \times D_n$ is called the domain of $f$, where $D_i =$...
A finite set of values $D$, of variable $X$, can assume. A binary-valued variable is a special case where $P_2 = 2$. Let $X_i$ be a variable taking a value from the set $D_i$. Let $S_i \subseteq D_i$ be a subset of $D_i$. Then $X_i^{S_i}$ represents the characteristic function $X_i^{S_i} = \begin{cases} 0 & \text{if } X_i \notin S_i \\ 1 & \text{if } X_i \in S_i \end{cases}$ $X_i^{S_i}$ is called a literal of variable $X_i$. A product term $S$ is a Boolean product (AND) of literals of all the variables of $D$. Thus, $S$ represents a subset of $D$. If $|S| = 1$ for all $X_i$, then the product term is also a minterm. If a product term evaluates to 1 for a given minterm, the product term is said to contain the minterm.

2.2 BDD's and Characteristic Functions

A binary decision diagram (BDD) [2] is a very compact canonical graph representation of Boolean functions. The reader is referred to [2] for details. BDD's hold key properties that make them a better representation for reasoning about Boolean functions than previously used representations, e.g. the sum-of-products representation. In addition to the canonical property, BDD's are in most cases much more compact than any sum-of-products that denotes the characteristic function. BDD's can also be used to represent finite sets efficiently. Any subset $A$ of $B^n$ can be denoted by a unique Boolean function $x_A : B^n \rightarrow B$. This function is called the characteristic function of $A$ and is defined in the following way: $x_A(x) = 1$ if and only if $x \in A$. Characteristic function is a very interesting implicit representation of Boolean sets because there is a direct correspondence between the set operators and the logical operators. Since a characteristic function is a Boolean function, it can be represented with a binary decision diagram and basic set operations can be performed directly on the graph representation.

2.3 Boolean Operations

In addition to the basic propositional logic operations, the existential operator (3) and the universal operator (V) are required for the implicit manipulation of sets. These Boolean quantifiers can also be computed directly on a BDD representation as follows. The existential quantification of a set of Boolean variables $X = \{x_1, x_2, \ldots, x_n\}$ with respect to the Boolean formula $f$ can be evaluated as

$$\exists X(f) = \exists x_1(\exists x_2(\ldots \exists x_n(\exists x_k(f))))$$

$$\exists x_k(f) = f_{x_k} + f_{\overline{x_k}}$$

where $f_{x_k}$ denotes the cofactor of formula $f$ with respect to a literal $x_k(\overline{x_k})$. Likewise, the universal quantifier can be evaluated as

$$\forall X(f) = \forall x_1(\forall x_2(\ldots \forall x_n(\forall x_k(f))))$$

$$\forall x_k(f) = f_{\overline{x_k}} + f_{x_k}$$

In this paper, we will use the notations $f \Rightarrow g$ to designate a logical implication from $f$ to $g$ (evaluated as $f + g$), $f \equiv g$ to designate an XOR operation, and $f \rightarrow g$ to designate a sharp product operation (evaluated as $f \cdot g$). These alternative notations are used to improve the readability of the manuscript.

3 Characteristic-Cube Function

3.1 Representation of Multiple-Valued Cubes

A multiple-valued function $f : D \rightarrow B$ can be represented efficiently as a characteristic function in BDD form by first encoding the multi-valued variables into binary vectors. This problem was addressed in [7]. Product terms and primes, however, have been represented explicitly using two-level cover concepts. This is a fundamental limitation of previous prime generation algorithms. Here, we propose to alleviate this bottleneck by using a symbolic representation based on binary decision diagrams to implicitly represent and manipulate multi-valued product terms and primes. Given a function $f : D \rightarrow B$, where $D = D_1 \times \ldots \times D_n$, and each $D_i$ is some finite set of $P_i$ ($P_i \geq 2$) values $\{0, \ldots, P_i - 1\}$, there can be $\prod_{i=1}^{n} P_i^2$ possible product terms. This is because a literal $X_i^{S_i}$ of the $i$-th variable can be any subset of $D_i$. In order to represent in an efficient way any possible set of product terms with a characteristic function in BDD form, an efficient encoding of all possible product terms is required. For this purpose, many encoding schemes are possible. Effectively, we need some function $\zeta : B^n \rightarrow P(D_i)$, where $P(D_i)$ is the powerset of $D_i$, to encode each variable $X_i$. Clearly, at least $P_i$ unique binary variables are required to encode $P(D_i)$. Here, we present an efficient symbolic representation based on the positional cube notation.

Definition 3.1 Let $X_i^{S_1}, X_2^{S_2}, \ldots, X_n^{S_n}$ be a product term. It can be represented in positional cube notation form as $c_{1_1}c_{2_1} \cdots c_{1_{P_1-1}}c_{2_{P_2-1}} \cdots c_{n_{P_n-1}}$ where $c_{ij} = 0$ if $j \notin S_i$ and $c_{ij} = 1$ if $j \in S_i$.

Definition 3.2 Let $C = \{T_1, \ldots, T_m\}$ be a set of product terms. A characteristic-cube function is simply a characteristic function

$$\chi_{\text{cube}} : B^n \rightarrow B$$

defined as follows: (1) $C = \sum_{i=1}^{m} P_i$.(2) $\chi_{\text{cube}}(c) = \begin{cases} 1 & \text{if and only if } \exists \exists' \in C \text{ such that} \\ \forall x_{i}, y_{i} = \{0, \ldots, P_{i} - 1\}, \ c_{ij} = \langle j \in S_{i} \rangle, \\ 0 & \text{otherwise} \end{cases}$

where $c = c_{1_1}c_{2_1} \cdots c_{1_{P_1-1}} \cdots c_{n_{P_n-1}}$ and $T = X_1^{S_1}X_2^{S_2} \cdots X_n^{S_n}$. Each component $c_{1_1}c_{2_1} \cdots c_{1_{P_1-1}}$ represents a set of values contained in the $i$-th literal of $c$, and consists of $P_i$ binary bits. Each bit is called a part. We denote $I(c)$ as the set of minterms of $D$ contained in $c$.

The key to the characteristic-cube form is the symbolic representation of product terms with binary decision diagrams. There is no direct correspondence between the number of product terms in the set and the size of the BDD that denotes the characteristic-cube function. Depending on the structural regularity of the set under consideration, very large sets of product terms may be symbolically represented in a compact form.
3.2 All-Zero Code Simplification

In the characteristic-cube representation, each literal \( X^S_i \) is represented by its corresponding component \( c_{i_1}c_{i_2}\cdots c_{i_{P-1}} \). With \( P_i \) bits, all possible subsets \( S_i \subseteq D_i \) of \( P_i \) values can be represented. However, in the two-level minimization problem (and most other problems), we do not need to consider the case where a literal contains no values (i.e., \( X^S_i \) where \( S_i = \{\} \)) since it has no real meaning. Hence, we do not need to consider product terms with any empty literals either. The collection of these "non-real" product terms can be used implicitly as "don't care" conditions to simplify the BDD representation for representing multiple-valued product terms. More specifically, the possible don't care conditions can be succinctly captured in the expression

\[
\sum_{i=1}^{n} \prod_{j=0}^{P_i-1} c_{ij}
\]

The term \( \prod_{j=0}^{P_i-1} c_{ij} \) expresses the condition for an empty literal and the outer summation is used to check if any literal violates this condition.

3.3 Relations and Properties

Characteristic-cube representation has a number of key properties that follow directly from the fact that it is a characteristic function representation. Therefore, set operations corresponds directly to logical operations. The characteristic-cube representation also has the interesting property that many useful relations can be easily defined and computed. We develop in this section several of these relations that will be used for the symbolic prime generation procedure. The equivalence of two product terms can be determined simply by checking if their characteristic-cube representations are identical.

**Definition 3.3** Let \( f : D \rightarrow B \) be a multiple-valued function, where \( D = D_1 \times \cdots \times D_n \) and \( D_i = \{0, \ldots, P_i - 1\} \). The equality relation of \( f \) is a characteristic function \( x_{equal} : B^2 \times B^2 \rightarrow B \), where \( L = \sum_{i=1}^{n} P_i \), such that \( x_{equal}(c, \hat{c}) = 1 \) if and only if the multiple-valued product terms \( c \) and \( \hat{c} \) are identical (contain exactly the same minterms).

**Theorem 3.1** Let \( f : D \rightarrow B \) be a multiple-valued function, where \( D = D_1 \times \cdots \times D_n \), and \( D_i = \{0, \ldots, P_i - 1\} \). The expression

\[
x_{equal}(c, \hat{c}) = \prod_{i=1}^{n} \prod_{j=0}^{P_i-1} c_{ij} \equiv c'_{ij}
\]

computes the equality relation.

**Proof:** Two product terms \( c \) and \( \hat{c} \) are identical if and only if they are products of the same literals. The expression \( \prod_{j=0}^{P_i-1} c_{ij} \equiv c'_{ij} \) ensures that the \( i \)-th literal of both product terms contain exactly the same subset of values from \( D_i \). The outer product \( \prod_{i=1}^{n} \) ensures that this holds for all variables.

A relation expressing the containment of two product terms can also be easily computed. A product term \( S = X_{i_1}^S X_{i_2}^S \cdots X_{i_{P-1}}^S \) contains another product term \( T = X_{i_1}^T X_{i_2}^T \cdots X_{i_{P-1}}^T \) if and only if \( S_i \supseteq T_i \), for all \( i \). The pairwise containment relation can be defined as follows.

**Definition 3.4** The containment relation of \( f \) is a characteristic function \( x_{contain} : B^2 \times B^2 \rightarrow B \), \( L = \sum_{i=1}^{n} P_i \), such that \( x_{contain}(c, \hat{c}) = 1 \) if and only if the multiple-valued product term \( c \) contains \( \hat{c} \).

**Theorem 3.2** The expression

\[
x_{contain}(c, \hat{c}) = \prod_{i=1}^{n} \prod_{j=0}^{P_i-1} c_{ij} \Rightarrow c'_{ij}
\]

computes the containment relation.

**Proof:** \( \prod_{j=0}^{P_i-1} c_{ij} \Rightarrow c'_{ij} \) ensures that the \( i \)-th literal of the product term \( c \) contains all the values contained in the same literal of the product term \( \hat{c} \). The outer product \( \prod_{i=1}^{n} \) ensures that this holds for all variables.

In the above expression, \( x_{contain}(c, \hat{c}) = 1 \) if and only if \( c \supseteq \hat{c} \).

**Definition 3.5** The strict containment relation of \( f \) is a characteristic function \( x_{strict-contain} : B^2 \times B^2 \rightarrow B \), \( L = \sum_{i=1}^{n} P_i \), such that \( x_{strict-contain}(c, \hat{c}) = 1 \) if and only if the multiple-valued product term \( c \) strictly contains \( \hat{c} \).

The characteristic function representing pairwise strict containment (\( c \supset \hat{c} \)) can be computed as follows.

**Theorem 3.3** The expression

\[
x_{strict-contain}(c, \hat{c}) = x_{contain}(c, \hat{c}) - x_{equal}(c, \hat{c})
\]

computes the strict containment relation.

**Proof:** Follows from the definitions.

4 Symbolic Prime Generation

In this section, techniques are given for implicit generation and representation of implicants and prime implicants. These techniques are simple but very efficient. They require only basic Boolean operations available in any BDD package. Given a function \( f : D \rightarrow B \), \( D = D_1 \times \cdots \times D_n \), the set of possible implicants and prime implicants can be efficiently generated using the characteristic-cube representation and the global relations defined in the previous section. We first consider the case for completely specified functions. Then we will consider the case for incompletely specified functions.

4.1 Completely Specified Functions

We first need to establish the relationship between the characteristic-cube representation and the multiple-valued function.

**Definition 4.1** Let \( f : D \rightarrow B \) be a multiple-valued function, where \( D = D_1 \times \cdots \times D_n \), and \( D_i = \{0, \ldots, P_i - 1\} \). The mapping relation for the domain \( D = D_1 \times \cdots \times D_n \) is the characteristic-cube function

\[
x_{mapping}(c, \hat{x}) = \prod_{i=1}^{n} \prod_{j=0}^{P_i-1} x_{i}^{(j)} \Rightarrow c_{ij}
\]

where \( x_{i}^{(j)} \) corresponds to the \( j \)-th value of the \( i \)-th variable of \( D \).

This relation provides the correspondence between a product term \( c \) and the minterms that it contains.
Theorem 4.1 Let \( f: D \rightarrow B \) be a multiple-valued function, where \( D = D_1 \times \ldots \times D_n \) and \( D_i = \{0, \ldots, P_i - 1\} \). 
\[
\chi_{\text{mapping}}(e, x) = 1 \text{ if and only if } x \in I(e).
\]

Proof: If the \( i \)-th variable of the minterm \( x \) has the value \( X(i) \), then the characteristic-cube representation of \( e \) should have a 1 in the \( j \)-th part of the \( i \)-th component (i.e. \( c_{ij} = 1 \)). This is precisely given by the expression \( X(i) \Rightarrow c_{ij} \).

Using the mapping relation, we can next derive a characteristic-cube function corresponding to the set of all possible implicants of \( f \). This can be simply expressed as follows.

Theorem 4.2 Let \( f: D \rightarrow B \) be a completely specified multiple-valued function, where \( D = D_1 \times \ldots \times D_n \), and \( D_i = \{0, \ldots, P_i - 1\} \), and let \( \chi_{\text{mapping}}(e, x) \) be the mapping relation for the domain \( D \). The characteristic-cube function for the set of product terms that form the set of all possible implicants of \( f \) is given by the expression
\[
\chi_{\text{implicants}}(e) = \forall x (\chi_{\text{mapping}}(e, x) \Rightarrow f(x))
\]

Proof: A product term \( c \) is an implicant of \( f \) if and only if all the minterms that it covers (i.e. \( f(e) \)) belong to the \( \text{ON-set of } f \). That is, \( \forall x \in f(e), f(x) = 1 \). This is precisely given by the expression \( \forall x (\chi_{\text{mapping}}(e, x) \Rightarrow f(x)) \).

The computation only makes use of standard BDD operations described in Section 2.

From the set of implicants \( f \), we can extract out symbolically the subset of implicants that are primes of \( f \) using the global relations developed in the previous section. Precisely, we remove from the characteristic function \( \chi_{\text{implicants}}(c) \) those implicants \( c \) that are fully contained (single-cube containment) by some other implicant \( c \in \chi_{\text{implicants}} \). The remaining set of implicants are guaranteed to be primes. This is achieved as follows.

Theorem 4.3 Let \( \chi_{\text{prime}}(c) \) be the complete set of implicants for the multiple-valued function \( f \) defined above. Then the characteristic-cube function for the set of primes is given by the expression
\[
\chi_{\text{prime}}(c) = \chi_{\text{implicants}}(c) - (\exists e (\chi_{\text{strict-contains}}(e, c) \chi_{\text{implicants}}(e)))
\]

Proof: \( (\exists e (\chi_{\text{strict-contains}}(e, c) \chi_{\text{implicants}}(e))) \) computes the set of product terms where each product term \( c \) in the set is fully contained by some implicant \( e \) in \( \chi_{\text{implicants}}(c) \). These product terms can be removed from \( \chi_{\text{implicants}}(c) \), as indicated by the sharp operation.

4.2 Incompletely Specified Functions

We now consider the case of incompletely specified function. We will assume an incompletely specified function \( F \) is given by a couple \((f, C)\), where \( f: D \rightarrow B \) is a function and \( C: D \rightarrow B \) is the care set. Minterms \( x \in D \) outside the care set \( C \) are considered don't care points. Any function \( f: D \rightarrow B \) such that \( f(x) = f(x) \) when \( C(x) = 1 \) is a possible implementation of \( F \).

In the case of an incompletely specified function, the set of all possible implicants is computed differently. A product term \( c \) is an implicant of \( F \) if it does not contain any minterm outside of \( f + C \), and it contains at least one minterm inside of \( C \). The computation of all implicants of an incompletely specified function is stated by the following theorem.

Theorem 4.4 Let \( F = (f, C) \) be an incompletely specified multiple-valued function, where \( f \) is a multiple-valued function, and \( C \) is the care set. Let \( \chi_{\text{mapping}}(e, x) \) be the mapping relation. Then, the characteristic-cube function for the set of all possible implicants is given by the expression
\[
\chi_{\text{implicants}}(c) = (\exists x (\chi_{\text{mapping}}(c, x) C(x))) (\forall x \chi_{\text{C(x)}} (\Rightarrow (\chi_{\text{mapping}}(c, x) \Rightarrow f(x))))
\]

Proof: The first component \((\exists x (\chi_{\text{mapping}}(c, x) C(x)))\) ensures that the cube \( c \) covers at least one point in the care set. The second component expresses the condition that every minterm in the care set covered by \( c \) must also belong to the \( \text{ON-set of } c \). The set of primes of \( F \) can then be extracted from the characteristic-cube function \( \chi_{\text{prime}}(c) \) by applying Theorem 4.3 above.

5 Extraction of Essential Primes

We now consider the problem of computing essential primes. This can be achieved by extracting symbolically from the set of primes those that are essential. Recall that a prime is deemed essential if it covers at least one minterm in the \( \text{ON-set} \) that is not covered by any other prime. The basic strategy here is to compute implicitly for each prime \( c \) those minterms that can still be covered if it is not considered.

Theorem 5.1 Let \( F = (f, C) \) be an incompletely specified multiple-valued function, where \( f \) is a multiple-valued function, and \( C \) is the care set. Let \( \chi_{\text{prime}}(c) \) be the set of prime implicants for \( F \). The characteristic-cube function for the set of essential prime implicants is given by the expressions
\[
\chi_{\text{still-cover}}(c, x) = \exists c (\chi_{\text{prime}}(c) \chi_{\text{equivalent}}(c, c))
\]

Proof: The characteristic function \( \chi_{\text{still-cover}}(c, x) \) represents for each possible \( c \) (not necessarily prime), the set of \( x \)'s still covered by \( \chi_{\text{prime}}(c) - c \). The expression \( \exists c (\chi_{\text{C(x)}}(f(x) \Rightarrow \chi_{\text{still-cover}}(c, x))) \) imposes the condition that every care set minterm in the \( \text{ON-set of } f \) must still be covered. If there exists one \( x \) not covered, then \( c \) is essential.

Note that we don't need to further check if \( c \) is a prime because by definition the set of primes can cover \( f \).

The above computation again requires only standard Boolean operations.

6 Multiple-Output Functions

In the previous sections, single-output multi-valued functions were considered. As shown in [6], a \( m \)-output multi-valued function can be transformed into a \( n + l \)-input single-output function where the \( n + l \)-th variable \( (X_{n+l}) \) can take on \( m \) values. That is, given
\[
f: D \rightarrow B^m,
\]
where \( D = D_1 \times \ldots \times D_n \), we can transform it into another function of the form
\[
f: D \times D_{n+l} \rightarrow B.
\]
deriving the mapping relation \( X_{\text{mapping}} \) and the set of all possible implicates \( X_{\text{implicants}} \) directly from the multiple-output function form. This avoids computing the characteristic function for representing the outputs. Each \( m \) unique variables \( \{ c_{n+1}, \ldots, c_{m+1} \} \) are needed. There is also the one-to-one correspondence between \( c_{n+1} \) and \( y_j \). The computation of the mapping relation \( X_{\text{mapping}} \) is modified as

\[
X_{\text{mapping}}(c, x, y) = \left( \prod_{i=1}^{n} \prod_{j=0}^{P_i-1} x_i(j) \Rightarrow c_{i j} \right) \left( \prod_{j=0}^{m-1} c_{n+j} \Rightarrow y_j \right) \tag{10}
\]

The characteristic functions \( X_{\text{equal}}, X_{\text{contain}}, X_{\text{strict contain}}, \) and \( X_{\text{prime}} \) are computed the same way as in previous sections since these characteristic functions do not depend on the variables in \( x \) or \( y \). The set of all possible implicants needed for prime computation needs to be modified as

\[
X_{\text{implicants}}(c) = \forall x, y
X_{\text{mapping}}(c, x, y) = \left( \prod_{j=0}^{m-1} f(j)(x) \equiv y_j \right) \tag{11}
\]

Incompletely specified multiple-output functions can be handled in a similar fashion. Also, the computation for detecting essential primes can be extended as well.

7 Implementation and Results

We have implemented the techniques described in this paper and have tested them on a range of large benchmarks. The experimental results are provided in Table 1. The CPU times are reported in seconds on a DECstation 5000 machine. In our experiments, the characteristic-cube representation was used to represent and generate primes symbolically. We used the all-zero code concept described in Section 3.2 to simplify the BDD sizes and improve the computation times. The experimental results show that the techniques presented in this paper can indeed be applied to functions with very large number of primes.

The first set of benchmarks is the MCNC FSM benchmark set. A total of 43 examples were considered. For these state machine examples, the state variable must be treated as a multiple-valued variable. The total number of primes and CPU time for all the examples combined are reported under the columns labeled \( \# \) primes and time, respectively. The examples \( ax4, lba, \) and \( misg \) are large PLA examples from the ESPRESSO benchmark set. For each of these examples, the techniques of Section 6 was applied to treat a multiple-output function as an equivalent single-output multiple-valued function. The number of primes can be quite significant for these examples when multiple-output primes are considered. Again, the number of primes and CPU times are reported appropriately. The third set of examples is a collection of multi-level logic benchmarks. The examples cbp32 and cbp16 are 32-bit and 16-bit adders, respectively. The example \( cim \) is a memory controller. The example \( s420 \) is from the ISCAS sequential test set. The examples \( m10 \) and \( m20 \) are min max circuits.

As can be seen from the table, large problem instances with prime sets well beyond the size of traditional prime generation methods can be efficiently handled using our proposed techniques with only modest CPU times. More extensive experimental results can be found in [4].

8 Conclusions

In this paper, a novel approach for generating primes and essential primes implicitly for multiple-valued functions has been presented. The approach is based on a symbolic representation of multiple-valued cubes called the characteristic-cube function that can efficiently denote very large sets of prime implicates with compact data structures like BDD's. The symbolic generation algorithms make use of only simple logic operators that are available in any BDD package. Alternative techniques based on a different symbolic representation have also been developed for Boolean functions [3]. However, the framework developed in this paper is much more general in the sense that binary-valued, multiple-valued, and multiple-output functions can all be treated uniformly.

References


<table>
<thead>
<tr>
<th>name</th>
<th>#files</th>
<th>#primes</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcnc.fsm</td>
<td>43</td>
<td>8.34e+03</td>
<td>34.76</td>
</tr>
<tr>
<td>ax4</td>
<td>128/28</td>
<td>2.04e+09</td>
<td>16.37</td>
</tr>
<tr>
<td>lba</td>
<td>47/17</td>
<td>1.05e+09</td>
<td>29.95</td>
</tr>
<tr>
<td>misg</td>
<td>56/23</td>
<td>2.20e+09</td>
<td>10.95</td>
</tr>
<tr>
<td>cbp32</td>
<td>65/52</td>
<td>4.29e+10</td>
<td>8.37</td>
</tr>
<tr>
<td>cbp16</td>
<td>33/17</td>
<td>6.55e+05</td>
<td>1.95</td>
</tr>
<tr>
<td>cim</td>
<td>46/54</td>
<td>1.51e+04</td>
<td>42.24</td>
</tr>
<tr>
<td>s420</td>
<td>35/18</td>
<td>6.56e+05</td>
<td>15.70</td>
</tr>
<tr>
<td>m10</td>
<td>45/30</td>
<td>5.14e+03</td>
<td>3.34</td>
</tr>
<tr>
<td>m20</td>
<td>83/60</td>
<td>5.24e+06</td>
<td>14.80</td>
</tr>
</tbody>
</table>

Table 1: Experimental results.