

A New Method to Compute Prime and Essential Prime Implicants of Boolean Functions

O. Coudert and J. C. Madre

Bull Corporate Research Center
Rue Jean Jaurès
78340 Les Clayes-sous-bois, FRANCE
coudert@crg.bull.fr, madre@crg.bull.fr

Abstract

Computing the prime implicants and the essential prime implicants of Boolean functions is a problem that has applications in several areas of computer science, for instance it is critical in circuit synthesis and optimization, but also in automated reasoning. In this paper we propose a new method to compute *implicitly* the sets of prime implicants and of essential prime implicants of incompletely specified Boolean functions. This method allows us to handle functions that have sets of prime implicants and of essential prime implicants several orders of magnitude larger than those of the functions that could be handled by existing methods. The key idea that makes these computations possible is to manipulate sets of prime and of essential prime implicants denoted by their *meta-products*, which are characteristic functions. These functions are represented with binary decision diagrams that are a very compact canonical graph representation of Boolean functions and that support very efficiently all the operations needed here.

1 Introduction

In the worst case, the number of prime implicants and essential prime implicants of a Boolean function is exponential with respect to the number of variables of this function [1]. Thus all techniques that perform a prime by prime computation are by nature limited to functions with relatively limited numbers of variables and of prime implicants. For instance the Espresso minimizer [1, 9] is practically limited to functions with less than 20,000 primes, and the reasoning maintenance system described in [7], which is based on a similar technology, has the same limitations.

We have introduced recently [3] a technique for verifying sequential circuits that is not limited by the number of states of these circuits. The key idea that makes this verification technique possible is to represent subsets of $\{0, 1\}^N$ with their characteristic functions. These Boolean functions are denoted by binary decision diagrams (BDD) that are a very compact

graph representation of propositional formulas. This representation allows us to operate on these sets without considering their elements individually. In this paper we show that this idea can be used with very good results to implement an *implicit* prime and essential prime implicant computation procedure.

This paper is divided in 7 parts. Section 2 presents the problem that will be treated here and the elementary concepts that will be used to solve it. Section 3 introduces the canonical representation, which we call *meta-product*, of sets of products. Section 4 explains how meta-products can be represented with binary decision diagrams and gives the functions *ProductCost* and *LiteralCost* that allow us to evaluate in linear time on meta-products the standard cost functions associated with sum of products. Section 5 shows how meta-products are used to compute implicitly the sets of prime and of essential prime implicants of Boolean functions with care sets. Section 6 gives experimental results. Finally Section 7 discusses this method and the experimental results obtained with it.

2 Definitions

We consider here propositional formulas built out of the finite set of propositional variables $\{x_1, \dots, x_n\}$. A *literal* is a propositional formula of the form (x_k) or (\bar{x}_k) . A *product* is a formula of the form $(l_1 \dots l_k)$, $k \leq n$, whose literals are built out of distinct variables. When $k = n$, such a product is called a *minterm*. The product p *contains* the product p' if and only if the formula $(p' \Rightarrow p)$ is a tautology. The product p *strictly contains* the product p' if p contains p' , and p' does not contain p .

A *sum of products* is a formula of the form $(p_1 + \dots + p_m)$, where p_1, \dots, p_m are distinct products. Any propositional formula f can be rewritten into a formula expressed as a sum of products. However this rewriting process is not canonical, which means that there exist different formulas written as sums of products that are equivalent. There are two standard cost functions for formulas expressed as sums of products [6, 9]. The first cost function of the formula $f = (p_1 + \dots + p_m)$, noted *ProductCost*(f), is the number of products in this sum, and the second cost function, noted *LiteralCost*(f), is defined as $LiteralCost(f) = \sum_{k=1}^{k=m} LiteralCost(p_k)$, where *LiteralCost*(p_k) is the number of literals that occur in the product p_k .

Any Boolean function from the set $\{0, 1\}^n$ into the set $\{0, 1\}$ can be represented by a propositional formula built out of the finite set of propositional variables $\{x_1, \dots, x_n\}$. Conversely any propositional formula built out of the finite set of propositional variables $\{x_1, \dots, x_n\}$ denotes a unique Boolean function from the set $\{0, 1\}^n$ into the set $\{0, 1\}$. In the sequel we will use the same symbol to represent a propositional formula and the Boolean function it denotes.

Any subset S of $\{0, 1\}^n$ can be represented by a unique Boolean function

χ_S from the set $\{0, 1\}^n$ into $\{0, 1\}$, called its *characteristic function*, that is defined in the following way:

$$\chi_S(x) = 1 \quad \text{if and only if} \quad x \in S.$$

Conversely, any Boolean function f from $\{0, 1\}^n$ into $\{0, 1\}$ denotes a unique subset S_f of $\{0, 1\}^n$ made of all the elements x of $\{0, 1\}^n$ such that: $f(x) = 1$. For instance the characteristic function of the subset $\{[01010], [11010], [01011], [11011]\}$ of $\{0, 1\}^5$ can be represented by the product $(x_2\overline{x_3}x_4)$. Characteristic functions are a very interesting implicit representation of Boolean sets because there is a direct correspondence between the set operators and the logical operators, for instance union corresponds with disjunction, intersection with conjunction, and complementation with negation.

A Boolean function f_C with a care set C is denoted by a couple (C, f) , where C is a subset of $\{0, 1\}^n$ and f is a Boolean function from $\{0, 1\}^n$ into $\{0, 1\}$, and is defined by the following equation:

$$\begin{aligned} f_C(x) &= f(x) & \text{if } x \in C, \\ f_C(x) &= * & \text{if } x \notin C, \end{aligned}$$

where the symbol “*” can be either 0 or 1. A product p built out of the variables $\{x_1, \dots, x_n\}$ is a *cube* of the function f_C if and only if the formula $(\chi_C \wedge p)$ is not an antilogy, and the formula $(\chi_C \Rightarrow (p \Rightarrow f))$ is a tautology. This expresses that the intersection between the set S_p denoted by p and the care set C is not empty, and that for any element of this intersection, the function f evaluates to 1. The product p is a *prime implicant* of f_C if and only if it is a cube of f_C , and there is no cube of f_C different from p that contains p . Finally, the product p is an *essential prime implicant* of f_C if and only if it is a prime implicant of f_C and the Boolean function denoted by the sum of all the primes of f_C that are different from p is not equal to f on the care set C .

Prime implicants and essential prime implicants play a very important part in the minimal cover problem [9]. The set of prime implicants of the Boolean function f_C is the set of products that can be used to build a minimal cover of f_C . Amongst all these prime implicants the essential ones are critical since they are prime implicants that occur in any minimal cover of f_C . In particular the cost of any minimal cover of f_C is at least equal to the cost associated with its essential prime implicants.

In the case where the care set of the function f_C is equal to the complete input set $\{0, 1\}^n$, the function f_C becomes a standard Boolean function. For this reason, we will in the following consider only functions with a care set, Boolean functions being just a particular case of such functions.

In this paper we will also use *quantified propositional formulas* that are formulas of the form $(Q_1x_1 \dots Q_kx_k f)$ where Q_1, \dots, Q_k are either

the universal or the existential quantifier, x_1, \dots, x_k are propositional variables, and f is a propositional formula [3]. For instance the formula $(\forall x_1 \exists x_2 (x_1 \vee (\neg x_2 \wedge x_3)))$ is such a quantified propositional formula. Any quantified propositional formula can be rewritten into a propositional formula using the following rewriting rules:

$$\begin{aligned} (\forall x f) &\rightarrow f[x \leftarrow 0] \wedge f[x \leftarrow 1], \\ (\exists x f) &\rightarrow f[x \leftarrow 0] \vee f[x \leftarrow 1], \end{aligned}$$

if the variable x is free in the formula f . For instance, the quantifier free form of the quantified formula given above is (x_3) . Adding quantifiers to Propositional Logic does not increase its expressive power, but it allows us to write very concisely expressions whose quantifier-free forms have exponential sizes.

3 Meta-Products

There are 3^n elements in the set P_n of products that can be built out of the set of variables $\{x_1, \dots, x_n\}$. Indeed there is a one to one mapping between this set and the set of strings $\{x_1, \bar{x}_1, \varepsilon\} \times \dots \times \{x_n, \bar{x}_n, \varepsilon\}$, where ε is the empty string. In order to represent any set of products with a Boolean function it is thus sufficient to use $\lceil \log_2(3^n) \rceil$ Boolean variables. However, though this number is theoretically sufficient, it is not the most interesting from the computing point of view. In this section we propose a representation of subsets of P_n that uses more Boolean variables (i.e. $2n$) but that has very good properties regarding the implicit prime implicant and the essential prime implicant computation technique presented in this paper.

3.1 Definition

In order to represent sets of products built out of the set of variables $\{x_1, \dots, x_n\}$, we introduce 2 finite sets of variables $\{o_1, \dots, o_n\}$ and $\{s_1, \dots, s_n\}$. We will call the variables o_1, \dots, o_n the *occurrence* variables and the variables s_1, \dots, s_n the *sign* variables. We define the mapping σ from the set $\{0, 1\}^n \times \{0, 1\}^n$ into the set P_n in the following way:

$$\sigma(o, s) = l_1 \dots l_n,$$

where l_k is the empty string ε if $o_k = 0$, l_k is the string \bar{x}_k if $o_k = 1$ and $s_k = 0$, and finally l_k is the string x_k if $o_k = 1$ and $s_k = 1$. For instance, with $n = 4$, we have $\sigma([0111], [1101]) = (x_2 \bar{x}_3 x_4)$.

The mapping σ defined above can not be a one to one mapping from the set $\{0, 1\}^n \times \{0, 1\}^n$ into the set P_n because it maps 4^n elements onto 3^n elements. There are two ways to get a canonical representation of the elements of P_n using the sets of variables $\{o_1, \dots, o_n\}$ and $\{s_1, \dots, s_n\}$. The

first way is to choose, amongst all the couples that denote the same product, a particular one that will be taken as reference. The second way, which leads to a simpler and better implementation, is to consider the canonical representation of a product to be the set of all the couples that denote this product. For instance, for $n = 4$, the canonical set representation of the product $(x_2\bar{x}_3x_4)$ is the set $\{([0111], [0101]), ([0111], [1101])\}$. Since any subset of $\{0, 1\}^n \times \{0, 1\}^n$ has a unique characteristic function from the set $\{0, 1\}^n \times \{0, 1\}^n$ into the set $\{0, 1\}$, we get the following canonical functional representation of sets of products.

Definition 3.1 *Let $P = \{p_1, \dots, p_m\}$ be a subset of products of P_n . Its meta-product \mathcal{P} is the characteristic function of the following subset of $\{0, 1\}^n \times \{0, 1\}^n$:*

$$\left(\bigcup_{k=1}^{k=m} \sigma^{-1}(p_k) \right)$$

3.2 Properties

Meta-products have immediate properties that are direct consequences of the fact that they are characteristic functions. For any meta-products \mathcal{P} and \mathcal{P}' , the function $(\mathcal{P} \vee \mathcal{P}')$ is the meta-product of the union of the sets of products P and P' denoted by \mathcal{P} and \mathcal{P}' respectively ; $(\mathcal{P} \wedge \mathcal{P}')$ is the meta-product of the intersection of these sets ; $(\neg\mathcal{P})$ is the meta-product of $(P_n \setminus P)$; the set P is included in the set P' if and only if the function $(\mathcal{P} \Rightarrow \mathcal{P}')$ is a tautology.

The product p associated with the couple (o, s) denotes a unique subset of the set $\{0, 1\}^n$, whose characteristic function, noted $\chi(o, s)$, is defined by the equation:

$$\chi(o, s) = \lambda x. \left(\bigwedge_{k=1}^{k=n} (o_k \Rightarrow (x_k \Leftrightarrow s_k)) \right)$$

Two couples (o, s) and (o', s') of $\{0, 1\}^n \times \{0, 1\}^n$ denote the same product if and only if the functions $\chi(o, s)$ and $\chi(o', s')$ are identical, which is the case if and only if the formula $Equal((o, s), (o', s'))$ is a tautology, where $Equal$ is defined by the equation:

$$Equal((o, s), (o', s')) = \left(\bigwedge_{k=1}^{k=n} (o_k \Leftrightarrow o'_k) \wedge (o_k \Rightarrow (s_k \Leftrightarrow s'_k)) \right)$$

This formula expresses that a variable occurs in the product denoted by the couple (o, s) if and only if it occurs in the product denoted by the couple (o', s') , and in this case it occurs with the same sign in both products.

The product p denoted by the couple (o, s) contains the product p' denoted by the couple (o', s') if and only if the formula $(\chi(o', s') \Rightarrow \chi(o, s))$ is

a tautology. This is the case if and only the formula $Contain((o, s), (o', s'))$ is a tautology, where $Contain$ is defined by the equation:

$$Contain((o, s), (o', s')) = \left(\bigwedge_{k=1}^{k=n} o_k \Rightarrow (o'_k \wedge (s_k \Leftrightarrow s'_k)) \right)$$

This formula expresses that if a variable occurs in the product p , it also occurs in the product p' , and it occurs with the same sign in both products.

The following theorem, whose proof can be found in [4], gives a very easy way to compute the Boolean function that is equal to the sum of all the products belonging to a set of products denoted by a meta-product. This theorem is critical in the implicit essential prime implicant computation method that will be introduced in Section 3.4.

Theorem 3.1 *Let $P = \{p_1, \dots, p_m\}$ be a set of products built out of the variables $\{x_1, \dots, x_n\}$, \mathcal{P} its meta-product, and f the Boolean function denoted by the sum of these products. We have the identity:*

$$f = \lambda s. (\exists o \mathcal{P}(o, s))$$

3.3 Relatively Prime Implicants

The product p of the set of products $P = \{p_1, \dots, p_m\}$ is a *relatively* prime implicant of P if and only if there is no product p' in the set P in which p is strictly contained. For example, in the set $P = \{x_1\bar{x}_2, \bar{x}_1x_2, \bar{x}_1x_2\bar{x}_3\}$, the products $(x_1\bar{x}_2)$ and (\bar{x}_1x_2) are relatively prime implicants, but $(\bar{x}_1x_2\bar{x}_3)$ is not, because it is contained in the product (\bar{x}_1x_2) . The following theorem [4] makes explicit the relation that exists between the meta-product of the set P and the meta-product of the subset of relatively prime implicants of P .

Theorem 3.2 *Consider a set of products P and its meta-product \mathcal{P} . The meta-product $\mathcal{I}(\mathcal{P})$ of the subset of relatively prime implicants of P is defined by the equation:*

$$\begin{aligned} \mathcal{I}(\mathcal{P}) = \lambda o. \lambda s. (\mathcal{P}(o, s) \wedge \\ (\forall o' \forall s' (\mathcal{P}(o', s') \wedge Contain((o', s'), (o, s))) \Rightarrow \\ Equal((o', s'), (o, s))))), \end{aligned}$$

that can be simplified into:

$$\mathcal{I}(\mathcal{P}) = \lambda o. \lambda s. (\mathcal{P}(o, s) \wedge (\forall o' \mathcal{P}(o \wedge o', s) \Rightarrow (o \leq o'))),$$

where $(o \wedge o') = (o_1 \wedge o'_1, \dots, o_n \wedge o'_n)$, and $(o \leq o') = \left(\bigwedge_{k=1}^{k=n} (o_k \Rightarrow o'_k) \right)$.

3.4 Relatively Essential Prime Implicants

The product p of the set of products $P = \{p_1, \dots, p_m\}$ is a *relatively essential* prime implicant of P if and only if the Boolean function that is the sum of all the products of the set $P \setminus \{p\}$ is different from the Boolean function that is the sum of all the products of the set P . The following theorem [4] states that the meta-product of the subset of relatively essential prime implicants of the set of products P can be computed from the meta-product \mathcal{P} of the set P .

Theorem 3.3 *Consider a set of products P and its meta-product \mathcal{P} . The meta-product $\mathcal{E}(\mathcal{P})$ of the subset of relatively essential prime implicants of P is defined by the equation:*

$$\begin{aligned} \mathcal{E}(\mathcal{P}) = \lambda o. \lambda s. (\mathcal{P}(o, s) \wedge \\ (\exists s' (\exists o' \mathcal{P}(o', s')) \oplus \\ (\exists o' \mathcal{P}(o', s') \wedge \neg \text{Equal}((o', s'), (o, s)))). \end{aligned}$$

4 Representing Meta-Products with BDD's

Binary decision diagrams are a very compact canonical graph representation of Boolean functions that has been introduced in 1986 by R. E. Bryant [2]. In addition with being canonical, the BDD of a function is in most cases smaller than any sum of products that denotes this function. Combining two BDD's G_1 and G_2 using the usual operators such as the conjunction or the disjunction is done in $O(|G_1| \times |G_2|)$, where $|G|$ is the number of vertices of the graph G . This means that the algorithms manipulating BDD's have a much more predictable behavior than the algorithms manipulating sums of products, because these basic operations are exponential in the worst case on this representation. Binary decision diagrams also are a good representation of subsets of $\{0, 1\}^n$ because there is no direct correspondence between the number of elements in such a subset and the size of the BDD that denotes this subset, so very large subsets of $\{0, 1\}^n$ can be represented with small BDD's.

4.1 The Variable Ordering Problem

Except for the special class of symmetric functions, the size of the BDD of a Boolean function heavily depends on the input variable ordering used to build this BDD. There exist functions from $\{0, 1\}^n$ into $\{0, 1\}$ whose BDD has a polynomial size (with respect to n) for the best input variable ordering and an exponential size for the worst ordering [2]. This section shows that there exists a good variable ordering of the variables $\{o_1, \dots, o_n\}$ and $\{s_1, \dots, s_n\}$ used to built BDD's of meta-products.

The equations that define the meta-products of relatively prime and relatively essential prime implicants use the terms $\text{Equal}((o, s), (o', s'))$,

$Contain((o, s), (o', s'))$, and $\chi(o, s)(x)$. There exist several variable orderings that minimize the sizes of the BDD's of these expressions. For the term $Equal((o, s), (o', s'))$, these variables orderings have the following property:

$$\begin{aligned} \{o_{\pi(1)}, o'_{\pi(1)}\} &< \{s_{\pi(1)}, s'_{\pi(1)}\} < \\ \{o_{\pi(2)}, o'_{\pi(2)}\} &< \{s_{\pi(2)}, s'_{\pi(2)}\} < \\ &\vdots \\ \{o_{\pi(n)}, o'_{\pi(n)}\} &< \{s_{\pi(n)}, s'_{\pi(n)}\}, \end{aligned}$$

where π is a permutation of the integers $\{1, \dots, n\}$. This relation states that the variables o_k and s_k must be clustered and that the different clusters thus formed can be permuted. The BDD of the term $Equal((o, s), (o', s'))$, shown in Figure 1, has $6n$ vertices for any variable ordering compatible with the relation given above.

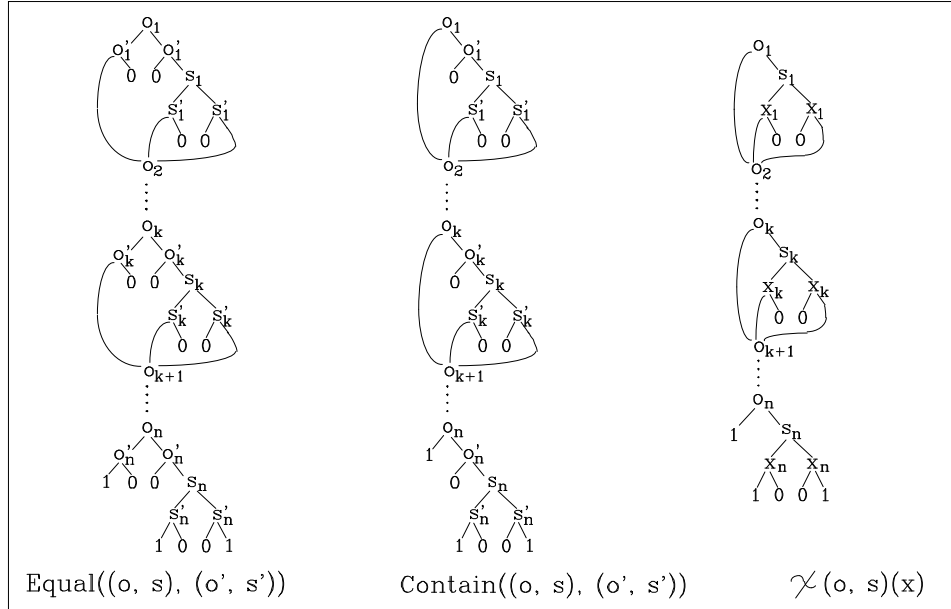


Figure 1. Minimal BDD's of the terms $Equal((o, s), (o', s'))$, $Contain((o, s), (o', s'))$, and $\chi(o, s)(x)$.

The different variable orderings that minimize the BDD of the term $Contain((o, s), (o', s'))$ are defined by the relation:

$$\begin{aligned} o_{\pi(1)} &< o'_{\pi(1)} < \{s_{\pi(1)}, s'_{\pi(1)}\} < \\ o_{\pi(2)} &< o'_{\pi(2)} < \{s_{\pi(2)}, s'_{\pi(2)}\} < \\ &\vdots \\ o_{\pi(n)} &< o'_{\pi(n)} < \{s_{\pi(n)}, s'_{\pi(n)}\}, \end{aligned}$$

where π is a permutation of the integers $\{1, \dots, n\}$. This relation states that the variables o_k, s_k, o'_k , and s'_k must be clustered, and that the different clusters thus formed can be permuted. The BDD of the term $Contain((o, s), (o', s'))$, shown in Figure 1, has $5n$ vertices for any variable ordering compatible with the relation given above. These variable orderings are compatible with the variable orderings that minimize the BDD of the expression $Equal((o, s), (o', s'))$.

Finally, the different variable orderings that minimize the BDD of the term $\chi(o, s)(x)$ are defined by the relation:

$$\begin{aligned} o_{\pi(1)} &< \{s_{\pi(1)}, x_{\pi(1)}\} < \\ o_{\pi(2)} &< \{s_{\pi(2)}, x_{\pi(2)}\} < \\ &\vdots \\ o_{\pi(n)} &< \{s_{\pi(n)}, x_{\pi(n)}\}, \end{aligned}$$

where π is a permutation of the integers $\{1, \dots, n\}$. This relation states that the variable o_k must be put before the variables s_k and x_k , and that the different clusters thus formed can be permuted. The BDD of the term $\chi(o, s)(x)$, shown in Figure 1, has $4n$ vertices for any variable ordering compatible with the relation given above.

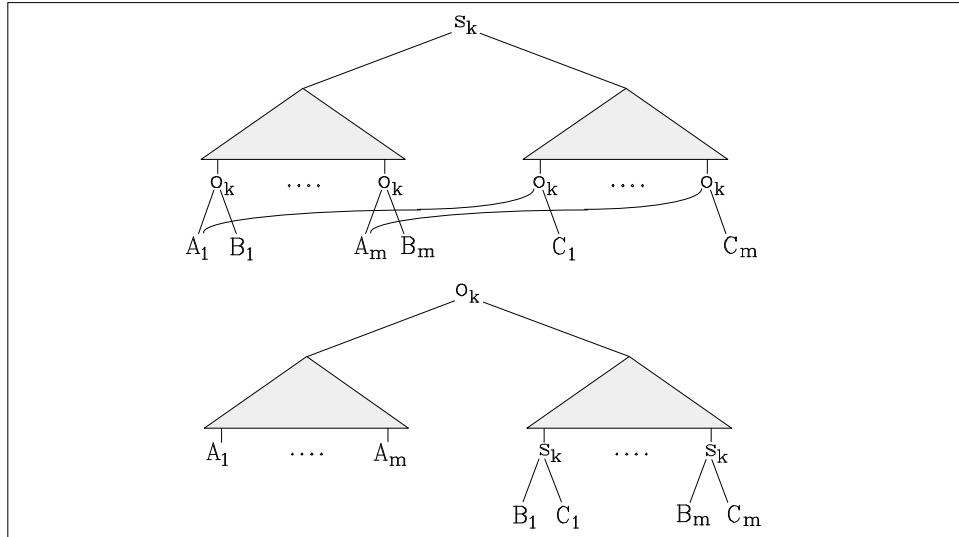


Figure 2. Swapping the variables o_k and s_k in BDD's of meta-products.

The different variable orderings given above are compatible, and it can be shown that they are also good for representing meta-products. Figure 2 shows that the variable s_k should not be put before the variable o_k because in many cases it will lead to a larger BDD. It can also be shown that the interleaving defined above makes some useful computations, for instance counting the number of products in the set denoted by a meta-product (see

Section 4.2), linear with respect to the size of the BDD of this meta-product. In the following we will consider that the BDD's of meta-products are built with the ordering:

$$o_1 < s_1 < o_2 < s_2 < \dots < o_n < s_n$$

4.2 Product and Literal Costs

Though there is no direct correspondence between the number of elements in the subset S_f of $\{0,1\}^n$ denoted by the function f and the size of the BDD of f , counting this number of elements can be done in a time linear with respect to the number of vertices in this BDD. Since several couples (o, s) can denote the same product, the number of elements in the subset $S_{\mathcal{P}}$ of $\{0,1\}^n \times \{0,1\}^n$ denoted by the *characteristic function* \mathcal{P} is not equal to the number of products in the subset of P_n denoted by the *meta-product* \mathcal{P} . We show here that the two cost functions on sums of products defined in Section 2 can be evaluated in times linear with respect to the numbers of vertices of the BDD's of their meta-products.

Consider the meta-product \mathcal{P} built out of the set of variables $\{o_1, \dots, o_n, s_1, \dots, s_n\}$, and P the set of products it denotes. Shannon expansion of \mathcal{P} with respect to the variable o_k provides us with the couple of functions $(\mathcal{P}_{\overline{o_k}}, \mathcal{P}_{o_k})$. The function $(\overline{o_k} \wedge \mathcal{P}_{\overline{o_k}})$ denotes the subset of P of products that do not contain the variable x_k . The function $(o_k \wedge \mathcal{P}_{o_k})$ denotes the subset of P of products that contain the variable x_k . This set is the union of two disjoint sets that are the subsets of P of products that contain the literal $(\overline{x_k})$ and the literal (x_k) respectively. These sets are denoted by the functions $(o_k \wedge \overline{s_k} \wedge \mathcal{P}_{o_k \overline{s_k}})$ and $(o_k \wedge s_k \wedge \mathcal{P}_{o_k s_k})$ respectively, where the couple of functions $(\mathcal{P}_{o_k \overline{s_k}}, \mathcal{P}_{o_k s_k})$ is Shannon expansion of the function \mathcal{P}_{o_k} with respect to the variable s_k . We thus have the equality:

$$\begin{aligned} ProductCost(\mathcal{P}) &= ProductCost(\mathcal{P}_{\overline{o_k}}) + \\ &\quad ProductCost(\mathcal{P}_{o_k \overline{s_k}}) + \\ &\quad ProductCost(\mathcal{P}_{o_k s_k}), \end{aligned}$$

from which it is immediate to derive a product counting function that has a complexity linear with respect to the number of vertices in the BDD of the meta-product \mathcal{P} .

The literal cost of the meta-product \mathcal{P} can also be evaluated with a complexity linear with respect to the number of vertices in the BDD of the meta-product \mathcal{P} . The literal cost associated with the subset of P of products containing the literal $(\overline{x_k})$ is the cost of the products in the set denoted by $\mathcal{P}_{o_k \overline{s_k}}$ plus the number of elements in this set, because the literal $(\overline{x_k})$ will be added to each element of this set. The same can be said of the set of products that contain the literal (x_k) . We thus have the equality:

$$LiteralCost(\mathcal{P}) = LiteralCost(\mathcal{P}_{\overline{o_k}}) +$$

$$\begin{aligned} & \text{LiteralCost}(\mathcal{P}_{o_k \overline{s_k}}) + \text{ProductCost}(\mathcal{P}_{o_k \overline{s_k}}) + \\ & \text{LiteralCost}(\mathcal{P}_{o_k s_k}) + \text{ProductCost}(\mathcal{P}_{o_k s_k}), \end{aligned}$$

from which it is also immediate to derive a literal counting function that has a complexity linear with respect to the number of vertices in the BDD of the meta-product \mathcal{P} .

5 Cubes, Primes and Essential Primes of Boolean Functions

The usual Boolean operators in addition with the quantified variable elimination are sufficient to compute the meta-products that denote the sets of prime implicants and of essential prime implicants of Boolean functions as well as Boolean functions with a care set. This computation is done in two main steps. The first step consists in computing the meta-product of the set of cubes of this function, and the second step consists in extracting the meta-products of the set of prime implicants and of essential prime implicants from this meta-product using the equations given in Section 3.

5.1 Cubes

The following theorem states the meta-product $Cube(f_C)$ of the set of cubes of a Boolean function f_C with the care set C can be computed using the usual Boolean operators in addition with the quantified variable elimination.

Theorem 5.1 *Let f_C be a Boolean function from the set $\{0,1\}^n$ into the set $\{0,1\}$ with the care set C . The meta-product $Cube(f_C)$, which denotes the set of all products built out of the set of variables $\{x_1, \dots, x_n\}$ that are cubes of the function f_C , is a Boolean function from the set $\{0,1\}^n \times \{0,1\}^n$ into $\{0,1\}$ defined by the equation:*

$$\begin{aligned} \text{Cube}(f_C) = \lambda o. \lambda s. (& (\exists x \chi(o, s)(x) \wedge \chi_C(x)) \wedge \\ & (\forall x \chi_C(x) \Rightarrow (\chi(o, s)(x) \Rightarrow f(x)))). \end{aligned}$$

In the case where the care set of the function f_C is the complete input set, this equation can be simplified into:

$$\text{Cube}(f_{\{0,1\}^n}) = \lambda o. \lambda s. (\forall x \chi(o, s)(x) \Rightarrow f(x)).$$

5.2 Prime Implicants and Essential Prime Implicants

Since the meta-product $Cube(f_C)$ of a function f_C represents all the products built out of the set of variables $\{x_1, \dots, x_n\}$ that are cubes of this function, computing the meta-product of all the products that are prime implicants of f_C comes down to computing the relatively prime implicants of this set of products.

Theorem 5.2 Let f_C be a Boolean function from the set $\{0,1\}^n$ into the set $\{0,1\}$ with the care set C , and $Cube(f_C)$ be the meta-product of the set of cubes of f_C . The meta-product $Prime(f_C)$, which is the meta-product of the sets of products built out the set of variables $\{x_1, \dots, x_n\}$ that are prime implicants of the function f_C , is equal to:

$$\begin{aligned} Prime(f_C) &= \mathcal{I}(Cube(f_C)) \\ &= \lambda o.\lambda s.(Cube(o, s) \wedge (\forall o' Cube(o \wedge o', s) \Rightarrow (o \leq o'))) \end{aligned}$$

Theorem 5.3 Let f_C be a Boolean function from the set $\{0,1\}^n$ into the set $\{0,1\}$ with the care set C , $Cube(f_C)$ be the meta-product of the set of cubes of f_C , and $Prime(f_C)$ be the meta-product of the set of prime implicants of f_C . The meta-product $Essential(f_C)$, which is the meta-product of the sets of products built out the set of variables $\{x_1, \dots, x_n\}$ that are essential prime implicants of the function f_C , is defined by the equation:

$$\begin{aligned} Essential(f_C) &= \mathcal{E}(Cube(f_C)) \\ &= \lambda o.\lambda s.(Cube(o, s) \wedge \\ &\quad (\exists s' f(s') \oplus \\ &\quad (\exists o' Cube(o', s') \wedge \neg Equal((o', s'), (o, s)))))) \\ &= \mathcal{E}(Prime(f_C)) \\ &= \lambda o.\lambda s.(Prime(o, s) \wedge \\ &\quad (\exists s' f(s') \oplus \\ &\quad (\exists o' Prime(o', s') \wedge \neg Equal((o', s'), (o, s)))))) \end{aligned}$$

The equations above show that there are two ways to compute the meta-product $Essential(f_C)$, either directly from the meta-product $Cube(f_C)$, or in two steps, by first extracting the meta-product $Prime(f_C)$ from the meta-product $Cube(f_C)$, and then by extracting the meta-product $Essential(f_C)$ from $Prime(f_C)$. Both ways use only the standard BDD operations such as the conjunction and the disjunction, in addition with the quantified variables elimination which itself can be implemented using these operators.

6 Experimental Results

We have used the method presented in this paper to compute the sets of prime and of essential primes implicants of all the output functions of some combinational and sequential circuits. Table 1 gives the characteristics of the circuits. Except for the entries *indust*, *math*, and *random*, **#Ins** is the number of input variables, **#Funs** is the number of output functions,

Name	#Ins #Files	#Funs	#Primes	#Essentials
<i>math</i>	23	128	8616	1937
<i>indust</i>	111	2412	87797	17284
<i>random</i>	11	189	118141	169
<i>cbp16</i>	32	17	655287	655287
<i>cbp32</i>	64	33	42949672960	42949672960
<i>dsip</i>	452	421	22850	2223
<i>mm10</i>	43	40	10015327	127139
<i>mm20_{no outputs}</i>	83	60	5243035	5242976
<i>pewd</i>	123	213	11442	1119
<i>s1423</i>	91	79	469397	36226
<i>prod</i>	20	33	24831	1950
<i>clm3</i>	46	32	55386	782
<i>clm1</i>	46	115	15219	862
<i>bred</i>	49	1	27778	27778
<i>add3</i>	21	11	77008	199
<i>add4</i>	29	12	688167	247
<i>addsub</i>	31	15	408190	102413
<i>rip04</i>	8	5	75	75
<i>rip08</i>	16	9	1499	1499
<i>mul06</i>	12	12	5430	765
<i>mul07</i>	14	14	28972	1551
<i>mul08</i>	16	16	152051	3879

Table 1. Characteristics of Examples.

#Primes is equal to $\sum_f \text{ProductCost}(\text{Prime}(f))$ for all output functions f , and **#Essentials** is equal to $\sum_f \text{ProductCost}(\text{Essential}(f))$. The entries *indust*, *math*, and *random* correspond to the directories of the MCNC 2-level minimization benchmark. Note that the output functions of these circuits have care sets. For these entries, column **#Files** is the number of circuits in this directory, and **#Funs**, **#Primes**, and **#Essentials** are the sums of the figures obtained for each circuit of the directory. All the circuits of Table 1 put together have 3867 output functions.

Each file has first been translated from its original format into an intermediate form in which Boolean functions are represented with a textual form of BDD's. The CPU times needed to perform these translations are not included in the results presented here. The variable ordering used to build these BDD's, computed using the heuristics described in [8], is the same for all functions of a file, and the occurrence and sign variables are ordered according to this input variable ordering.

Table 2 gives the CPU times in seconds and the memory sizes in Mbytes needed to perform the computations on a SUN Sparc2 workstation. Column **Time_p** is the CPU time needed to compute the meta-product of the set of prime implicants (Theorem 5.2) of each output function of the circuit,

Name	Primes		Essentials			
	Time _p	Mem _p	Time _{ec}	Mem _{ec}	Time _{ep}	Mem _{ep}
<i>math</i>	17.6	1.2	50.5	2.1	32.5	1.7
<i>indust</i>	289	1.8	1273	4.3	639	3.5
<i>random</i>	511	2.4	1427	4.3	892	2.7
<i>cbp16</i>	2.5	0.8	4.3	1.1	5.4	1.1
<i>cbp32</i>	12.3	1.4	22.5	2.0	30.5	1.9
<i>dsip</i>	21.9	1.0	71.8	1.5	62.9	1.3
<i>mm10</i>	598	4.0	5694	10.8	884	5.3
<i>mm20</i>	10.7	1.3	23.5	2.0	24.2	1.9
<i>pewd</i>	15.1	1.0	46.5	1.5	39.6	1.3
<i>s1423</i>	154	1.8	891	3.8	347	3.0
<i>prod</i>	14.2	1.1	45.7	1.8	34.2	1.5
<i>clm3</i>	32.1	1.6	128	3.2	69.2	2.2
<i>clm1</i>	47.4	2.9	358	9.4	93.5	4.3
<i>bred</i>	1.1	0.5	2.3	0.9	2.5	0.9
<i>add3</i>	11.6	1.3	36.1	1.7	30.3	1.4
<i>add4</i>	18.5	1.3	55.0	2.0	43.4	1.5
<i>addsub</i>	3.2	0.9	6.4	1.3	7.7	1.3
<i>rip04</i>	0.1	0.1	0.2	0.1	0.2	0.1
<i>rip08</i>	0.5	0.2	0.8	0.4	0.9	0.5
<i>mul06</i>	23.8	1.4	108	3.9	48.2	1.8
<i>mul07</i>	138	3.9	932	18.2	275	5.5
<i>mul08</i>	1001	15.6	–	> 24	1903	22.1

Table 2. Experimental Results.

and **Mem_p** is the memory size needed to make this computation. Columns **Time_{ec}** and **Mem_{ec}** are the CPU time and memory size to compute the meta-product of the set of essential prime implicants of each function, by first computing the meta-product of the set of cubes of this function (Theorem 5.1), and then using this meta-product to compute the requested meta-product (Theorem 5.3). Finally, columns **Time_{ep}** and **Mem_{ep}** are the CPU time and memory size to compute the meta-product of the set of essential prime implicants of each function, by first computing the meta-product of the set of cubes of this function, then extracting the meta-product of the set of prime implicants of the function (Theorem 5.2), and finally using this meta-product to compute the result (Theorem 5.3).

These experimental results show that the method presented here can be applied to Boolean functions with very large number of primes. For all but small examples, the essential prime implicant computation that uses the intermediate meta-product *Prime* is more efficient than the computation using the meta-product *Cube*. This comes from the fact that for all large examples, the BDD of the meta-product *Prime* is much smaller than the BDD of *Cube* and that computation costs are directly related to the sizes

of these BDD's. This is particularly the case for the multipliers *mul07* and *mul08*, and the *mm10* circuit. Moreover the results show that there is *no relation* between the CPU times and the number of primes in the sets denoted by the meta-products computed here.

7 Conclusion

In this paper we have presented a method to compute in an implicit way the sets of cubes, of prime implicants, and of essential prime implicants of Boolean functions with care sets. The key ideas used in this method are to represent products built out of the set of variables $\{x_1, \dots, x_n\}$ with formulas, called *meta-products*, built out of 2 sets of variables $\{o_1, \dots, o_n\}$ (the occurrence variables) and $\{s_1, \dots, s_n\}$ (the sign variables), and to denote these formulas with binary decision diagrams.

Though this method allows us to deal with functions with very large numbers of prime implicants and of essential prime implicants, there are functions for which the memory size needed to store intermediate results is too large to make these computations possible. In order to decrease this memory size, we have developed an *incremental* implicit prime computation method that performs a depth-first traversal of the BDD of the function under consideration and computes the result bottom-up. This method [5], which will be presented in a forthcoming paper, allows us to handle functions that cannot be handled by the method presented here.

It has been shown [10, 11] that computing the prime implicants and essential primes implicants of a *multiple-valued* input and output function comes down to computing the prime implicants and essential primes implicants of a multiple-valued input Boolean function. There are several ways to extend the method presented here in order to deal with such Boolean functions, that correspond to different ways to encode the multiple-valued input variables of the function into Boolean variables. With these different encodings correspond different meta-product representations and their associated calculi. These different techniques will also be presented in a forthcoming paper.

Acknowledgments

The authors would like to thank Henri Fraisse who introduced the meta-product representation, and Gary Hachtel and Bill Lin for many fruitful discussions.

References

- [1] Brayton, R. E., Hachtel, G. D., McMullen, C. T., Sangiovanni-Vincentelli, A. L., *Logic Minimization Algorithms for VLSI Synthesis*,

Kluwer Academic Publishers, 1984.

- [2] Bryant, R. E., “Graph-based Algorithms for Boolean Functions Manipulation”, *IEEE Trans. on Computers*, Vol C35, 1986.
- [3] Coudert, O., Madre, J. C., “A Unified Framework for the Formal Verification of Sequential Circuits”, in *Proc. of ICCAD’90*, Santa-Clara, November 1990.
- [4] Coudert, O., Madre, J. C., *A New Method to Compute Prime and Essential Prime Implicants of Boolean Functions*, Bull Research Report N°91028, November 1991.
- [5] Coudert, O., Madre, J. C., *Implicit and Incremental Computation of Prime and Essential Prime Implicants of Boolean Functions*, Bull Research Report N°91033, November 1991.
- [6] Hong, S. J., Muroga, S., “Absolute Minimization of Completely Specified Switching Functions”, *IEEE Trans. on Computers*, pp. 53-65, Vol 40, 1991.
- [7] Madre, J. C., Coudert, O., “A Logically Complete Reasoning Maintenance System Based on a Logical Constrain Solver”, in *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*, Sydney, Australia, August 1991.
- [8] Minato, S., Ishiura, N., Yajima, S., “Fast Tautology Checking Using Shared Binary Decision Diagrams - Experimental Results”, in *Proc. of the Workshop on Applied Formal Methods for Correct VLSI Design*, L. Claesen Ed., North Holland, 1989.
- [9] Rudell, R. L., *Multiple-Valued Logic Minimization for PLA Synthesis*, Technical Report M86/65, UCB, 1986.
- [10] Rudell, R. L., Sangiovanni-Vincentelli, A. L., “Multiple Valued Minimization for PLA Optimization”, *IEEE Trans. on CAD*, Vol 6, No 5, September 1987.
- [11] Sasao, T., “An Application of Multiple-Valued Logic to a Design of Programmable Logic Arrays”, in *Proc. of 8th International Symposium on Multiple Valued Logic*, 1978.