

A New Implicit Graph Based Prime and Essential Prime Computation Technique

O. Coudert and J.C. Madre
Bull Corporate Research Center
Rue Jean Jaurès
78340 Les Clayes-sous-bois FRANCE

Abstract

Implicit set manipulation techniques based on binary decision diagrams (BDD) have made possible the verification of finite state machines with state graphs too large to be built. More recently, it has been shown that these techniques could be used with success to compute the sets of prime and essential prime implicants of Boolean functions for which these sets are much too large to be explicitly built. The key property of this approach is that the cost of these procedures are not related anymore with the number of implicants to be computed but with the sizes of the manipulated BDDs. This paper presents a new implicit prime computation procedure that allows us to handle a larger class of problems than the previously published ones do.

1 Introduction

We have recently introduced a prime implicant computation technique that makes possible to handle Boolean functions with sets of prime and of essential prime implicants too large to be explicitly built [5, 7]. The key ideas that underlie this technique are to represent and to compute these sets implicitly using *meta-products* [6], and to represent these meta-products with binary decision diagrams (BDD) [3]. This technique overcomes all previously known computation techniques because its cost is related to the size of the BDDs it manipulates, and not to the number of computed prime implicants.

The BDDs of meta-products that represent sets of prime implicants are very redundant. Eliminating these redundancies dramatically reduces the size of these BDDs, and produces a new canonical representation of *prime sets*, called the *Implicit Prime Set* (IPS) representation. However IPSs cannot be used to represent anything but prime sets. The problem is that the techniques presented in [7] manipulate sets of products that are not all prime implicants, and so they cannot use IPSs. In this paper, we present a new graph calculus specifically dedicated to the implicit manipulation of prime sets and we show that this calculus supports prime and essential prime implicant computation algorithms that

are more efficient than the ones presented in [7]. In particular these algorithms can handle with success all the vectorial Boolean functions described in the MCNC benchmark [15], among which there are vectorial functions that could not be handled by any published procedure.

The paper is divided in 6 parts. Section 2 presents the problems addressed here, and introduces the notations and the elementary concepts that will be used to solve them. Section 3 explains why BDDs of *meta-products* of prime sets are redundant, and defines the IPS representation. Section 4 presents the IPS based prime and essential prime implicant computation of partial Boolean functions. Section 5 presents the theorems that allow us to handle vectorial Boolean functions using the procedure presented here. Section 6 gives experimental results obtained with this new procedure.

2 Definitions and Notations

2.1 Formulas and Functions

The commutative binary operator “ \vee ” and the unary operator “ \neg ” are defined on the set $\{0, 1, *\}$ by:

$$\begin{aligned}\neg 0 &= 1 \\ \neg 1 &= 0\end{aligned}$$

$$\begin{aligned}
\neg * &= * \\
0 \vee 0 &= 0 \\
0 \vee 1 &= 1 \\
1 \vee 1 &= 1 \\
* \vee 0 &= * \\
* \vee 1 &= 1
\end{aligned}$$

The set of propositional formulas built out the variables $\{x_1, \dots, x_n\}$ is recursively defined as follows: 0, 1, * and x_1, \dots, x_n are propositional formulas ; if f and g are propositional formulas, then $(\neg f)$ and $(f \vee g)$ are too. This logic system contains the usual Propositional Logic, and is called the 3-valued Logic of Kleene [10]. A propositional formula denotes a unique function from $\{0, 1\}^n$ into $\{0, 1, *\}$, so we will use the same symbol to represent a propositional formula and the function it denotes. A Boolean function is a function from $\{0, 1\}^n$ into $\{0, 1\}$.

A *literal* is a propositional variable x_k or its negation, also noted $\overline{x_k}$. The couple of functions $(f_{\overline{x_k}}, f_{x_k})$ is the *Shannon expansion* of f with respect to x_k [1]. We will use the abbreviations $(\exists x_k f) = (f_{\overline{x_k}} \vee f_{x_k})$, and $(f \wedge g) = \neg(\neg f \vee \neg g)$.

A function f from a set E into $\{0, 1\}$ denotes a unique subset $f^{-1}(1)$ of E . Conversely any subset S of E is denoted by a unique function from E into $\{0, 1\}$, called its *characteristic function*, that is valued to 1 for every x of S , and that is valued to 0 elsewhere. Thanks to this correspondence, we will make no difference between a set and its characteristic function.

Let f be a function from E into $\{0, 1, *\}$. We note f^0, f^1 , and f^* the characteristic functions of the sets $f^{-1}(0)$, $f^{-1}(1)$, and $f^{-1}(*)$ respectively. Any function f from $\{0, 1\}^n$ into $\{0, 1, *\}$ can be written as $f = (* \wedge f^*) \vee f^1$, where f^* and f^1 are exclusive Boolean functions. The function f is said to be *completely defined* if f^* , also called the *don't care set* of f , is empty.

2.2 Products and Implicants

A *product* built on the space $E = E_1 \times \dots \times E_n$ is a non empty cartesian product $S_1 \times \dots \times S_n$, in which each set S_k is a subset of the set E_k . The containment relation " \supseteq " is a partial order on the products. Let f be a function from E into $\{0, 1, *\}$. A product p is an *implicant* of f iff $p \cap f^0 = \emptyset$. A product p is a *prime implicant* of f iff p is an implicant of f , and if there is no other implicant of f that contains p . In other words, p is a prime implicant of f iff it is a maximal element of the set of implicants of f with respect to " \supseteq ". A product p is an *essential prime implicant* of f iff there exists an element x of E such that $f(x) = 1$ and such that p is the only prime implicant of f that contains x . We will note $Prime(f)$ and $Ess(f)$

the sets of prime implicants and of essential prime implicants of the function f respectively.

In the particular case where each set E_k is equal to $\{0, 1\}$, we note P_n the set of products. In fact $P_n = \{\epsilon, \overline{x_1}, x_1\} \times \dots \times \{\epsilon, \overline{x_n}, x_n\}$, where ϵ is the empty string. An element of P_n is a string interpreted as the conjunction of its literals, which is the characteristic function of the set it represents. For instance the product $x_1 \overline{x_2} x_4$ of P_4 represents the subset $\{1\} \times \{0\} \times \{0, 1\} \times \{1\}$, i.e. $\{[1001], [1011]\}$, of $\{0, 1\}^4$.

Let P be a subset of P_n . We note P_{ϵ_k} the subset of products of P in which none of the literals $\overline{x_k}$ and x_k occurs. We note $P_{\overline{x_k}}$ the set of products containing no occurrence of $\overline{x_k}$, such that $\{\overline{x_k}\} \times P_{\overline{x_k}}$ is the subset of products of P in which the literal $\overline{x_k}$ occurs. We note P_{x_k} the set of products containing no occurrence of x_k , such that $\{x_k\} \times P_{x_k}$ is the subset of products of P in which the literal x_k occurs. For example, if $P = \{x_1 \overline{x_4}, x_2, x_2 x_4, \overline{x_2} x_3\}$, we have $P_{\epsilon_2} = \{x_1 \overline{x_4}\}$, $P_{\overline{x_2}} = \{x_3\}$, and $P_{x_2} = \{\epsilon, x_4\}$. The sets defined above allow us to build the following canonical partition of the set P :

$$P = P_{\epsilon_k} \cup (\{\overline{x_k}\} \times P_{\overline{x_k}}) \cup (\{x_k\} \times P_{x_k}).$$

2.3 IP, EE, and Prime Sets

Let P be a set of products built on E , and S be a subset of E . We note $IP(P, S)$ (for *intersecting products*) the set of products of P that contain at least one element of S :

$$IP(P, S) = \{p \in P \mid p \cap S \neq \emptyset\}.$$

We note $EE(P)$ (for *essential elements*) the set of elements of E that are contained by one and only one product of P :

$$EE(P) = \{x \in E \mid \exists! p \in P, x \in p\}.$$

A *prime set* is a set of products P such that there is no product q contained by $(\bigcup_{p \in P} p)$ that strictly contains a product of P , which is the case if the following predicate holds:

$$\neg(\exists q \in 2^{E_1} \times \dots \times 2^{E_n}, \exists p \in P, (\bigcup_{p \in P} p) \supseteq q \supset p).$$

Note that a prime set is a set of products that are all maximal with respect to " \supseteq ", but the converse is not true. For instance the set $\{\overline{x_1}, x_1\}$ is a set of maximal products, but it is not a prime set. In particular, the sets $Prime(f)$ and $Ess(f)$ are prime sets. The set of prime sets is closed under the intersection, the difference, the cartesian product, the IP operation, and the canonical decomposition defined above. However it is not closed under the complementation, the union, or the concatenation.

3 The IPS Representation

We have presented in [6] an implicit representation of sets of products, called the *meta-product* representation. We have shown that this representation supports with a very good efficiency the prime and essential prime implicant computation of Boolean functions with sets of such implicants much too large to be explicitly built [5, 7]. In this section we show that meta-products are very redundant when used to represent prime sets, and we present the *implicit prime set* (IPS) which is the representation obtained by eliminating these redundancies from the meta-product representation. We assume the reader familiar with the binary decision diagrams (BDDs), which are a compact and canonical representation of Boolean functions [3].

3.1 Meta-products

We first define a many-to-one mapping σ from the set $\{0, 1\}^n \times \{0, 1\}^n$ onto the set P_n as follows [6]: $\sigma([o_1 \dots o_n], [s_1 \dots s_n]) = l_1 \dots l_n$, where $l_k = \epsilon$ if $o_k = 0$, $l_k = \overline{x_k}$ if $o_k = 1$ and $s_k = 0$, and finally $l_k = x_k$ if $o_k = 1$ and $s_k = 1$. For instance, the couple $([1101], [1001])$ denotes the product $x_1 \overline{x_2} x_4$. The variables o_k are the *occurrence* variables, and the variables s_k the *sign* variables. In the sequel we will note o and s the vectors $[o_1 \dots o_n]$ and $[s_1 \dots s_n]$ respectively.

We call *meta-product* \mathcal{P} of a subset of products P of P_n the characteristic function of the set $(\bigcup_{p \in P} \sigma^{-1}(p))$. Since the collection $(\sigma^{-1}(p))_{p \in P_n}$ is a *partition* of $\{0, 1\}^n \times \{0, 1\}^n$, meta-products are a canonical functional representation of subsets of P_n [6]. For the same reason, set manipulations of meta-products are directly related with logical operators [6]: 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' iff $(\mathcal{P} \Rightarrow \mathcal{P}') = 1$; moreover, the meta-product of P_{ϵ_k} is $(\neg o_k \wedge \mathcal{P}_{\overline{o_k}})$, the meta-product of $P_{\overline{x_k}}$ is $(\neg o_k \mathcal{P}_{o_k \overline{s_k}})$, and the meta-product of P_{x_k} is $(\neg o_k \wedge \mathcal{P}_{o_k s_k})$.

3.2 Implicit Prime Sets

We define the Implicit Prime Set (IPS) representation as follows.

Definition 1 Let P be a prime set, and \mathcal{P} its meta-product. The IPS of P is the function:

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

where \downarrow is the “constrain” operator defined in [4].

Note that the IPS of P depends on the variable ordering because of the use of the “ \downarrow ” operator, which is not the case with the meta-product representation.

Theorem 1 IPSs are a canonical representation of prime sets.

Proof. The meta-product \mathcal{P} of any set of products P is such that: (1) $\mathcal{P}_{\overline{o_k s_k}} = \mathcal{P}_{\overline{o_k} s_k}$. Suppose for instance that this is false for $k = 1$. Then there exist $o' = [o_2 \dots o_n]$ and $s' = [s_2 \dots s_n]$ belonging to $\{0, 1\}^{n-1}$ such that $\mathcal{P}(0o', 0s') \neq \mathcal{P}(0o', 1s')$. For instance, assume that $\mathcal{P}(0o', 0s') = 1$ and $\mathcal{P}(0o', 1s') = 0$. Then the product $p = \sigma(0o', 0s')$ belongs to P . But the definition of σ implies that $\sigma(0o', 1s') = \sigma(0o', 0s')$, so $(0o', 1s') \in \sigma^{-1}(p)$, which implies that $\mathcal{P}(0o', 1s') = 1$, which is impossible by hypothesis. A consequence of this property is that, for any path in the BDD of \mathcal{P} on which the occurrence variable o_k is set to 0, changing the value of the sign variable s_k does not change the leaf that is reached.

Moreover, the meta-product \mathcal{P} of any prime set P is such that: (2) $(\mathcal{P}_{\overline{o_k}} \wedge \mathcal{P}_{o_k}) = 0$. Suppose for instance that this is false for $k = 1$. Then there exist $o' \in \{0, 1\}^{n-1}$ and $s \in \{0, 1\}^n$ such that $\mathcal{P}(1o', s) = \mathcal{P}(0o', s) = 1$. Let $p = \sigma(1o', s)$. Both products p and $l_1 p$, where $l_1 = \overline{x_1}$ if $s_1 = 0$ and $l_1 = x_1$ if $s_1 = 1$, belong to P , which implies that P is not a prime set, because $(\bigcup_{p \in P} p) \supseteq p \supset l_1 p$. A consequence of this property is that all the occurrence variables occur on each path from the root of the BDD of \mathcal{P} to 1.

Finally, the meta-product \mathcal{P} of any prime set P is such that: (3) $(P_{o_k \overline{s_k}} \wedge P_{o_k s_k}) = 0$. Suppose for instance that this is false for $k = 1$. Then there exist o' and s' belonging to $\{0, 1\}^{n-1}$ such that $\mathcal{P}(1o', 0s') = \mathcal{P}(1o', 1s') = 1$. Let $q = \sigma(1o', 0s')$. Both the products $x_1 q$ and $\overline{x_1} q$ belong to P , which implies that $(\bigcup_{p \in P} p) \supseteq (x_1 q \cup \overline{x_1} q) = q \supset x_1 q$, and so P is not a prime set. A consequence of this property is that the sign variable s_k occurs on each path from the root of the BDD of \mathcal{P} to the leaf 1 on which the occurrence variable o_k is set to 1.

It remains to prove that Properties (1), (2), and (3) are sufficient to build the meta-product of a prime set from its IPS. However, since this proof is technically tedious [8], it is not presented here. \square

Applying the “ \downarrow ” operator on the BDD of \mathcal{P} using the function $(\exists s \mathcal{P}(o, s))$ as the constraint consists in iteratively applying the reduction rules shown in Figure 1 on this BDD until saturation.

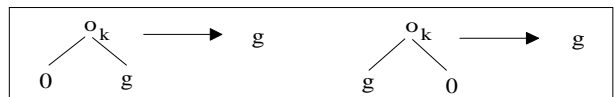


Figure 1. Reduction rules.

Figure 2 shows the BDD with 14 vertices of the meta-product of the prime set $\{x_1 \overline{x_4}, x_2 x_4, \overline{x_2} x_3\}$ of

P_4 and the BDD with 6 vertices of its IPS, both built with the variable ordering $o_1 < s_1 < \dots < o_4 < s_4$. Experiences actually show that the ratio between the number of vertices in the BDD of the meta-product of a prime set and the number of vertices in the BDD of its IPS is more than 10 for large examples, which means that this new representation is dramatically more compact than the meta-product representation.

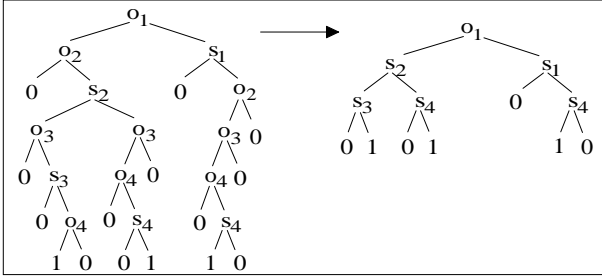


Figure 2. Meta-product and IPS.

4 Prime Computation of Boolean Functions

In this section, we show how the sets of prime and essential prime implicants of a partial Boolean function can be computed using manipulations of prime sets. We present a new algorithm for implicit essential prime computation that will be shown in Section 6 to be much more efficient than the one presented in [7].

4.1 Prime Implicants

Let f be a function from $\{0, 1\}^n$ into $\{0, 1, *\}$. The theorem below shows that we need only manipulations of prime sets to compute, using a recursive bottom-up scheme, the set $Prime(f)$. Note that the set of meaningful prime implicants, i.e. that contain at least one element in the care set, is $IP(Prime(f), \neg f^*)$.

Theorem 2 *The set of prime implicants $Prime(f)$ of the function f from $\{0, 1\}^n$ into $\{0, 1, *\}$ is defined by:*

$$\begin{aligned} Prime(0) &= \emptyset \\ Prime(1) &= \{\epsilon\} \\ Prime(*) &= \{\epsilon\} \\ Prime(f) &= \\ &Prime(f_{\overline{x_k}} \wedge f_{x_k}) \cup \\ &\{\overline{x_k}\} \times (Prime(f_{\overline{x_k}}) \setminus Prime(f_{\overline{x_k}} \wedge f_{x_k})) \cup \\ &\{x_k\} \times (Prime(f_{x_k}) \setminus Prime(f_{\overline{x_k}} \wedge f_{x_k})) \end{aligned}$$

4.2 Essential Prime Implicants

The following result is deduced from the definition of essential prime implicants of a function.

Theorem 3 *The set of essential prime implicants $Ess(f)$ of the function f from $\{0, 1\}^n$ into $\{0, 1, *\}$ is defined by:*

$$Ess(f) = IP(Prime(f), \neg f^* \wedge EE(Prime(f)))$$

The functions EE and IP are made explicit by the following results.

Theorem 4 *Let P be a set of products, and f be a Boolean function. The following equations [7] are sufficient to compute the set $IP(P, f)$:*

$$\begin{aligned} IP(P, 0) &= \emptyset \\ IP(P, 1) &= P \\ IP(P, f) &= IP(P_{\epsilon_k}, f_{\overline{x_k}} \vee f_{x_k}) \cup \\ &\{\overline{x_k}\} \times IP(P_{\overline{x_k}}, f_{\overline{x_k}}) \cup \\ &\{x_k\} \times IP(P_{x_k}, f_{x_k}) \end{aligned}$$

Theorem 5 *Let P be a set of products. $EE(P)$ can be computed with the following equations, where Cov is a function from 2^{P^n} into $(\{0, 1\}^n \rightarrow \{0, 1, *\})$, and \otimes is a commutative operator over $(\{0, 1\}^n \rightarrow \{0, 1, *\})$.*

$$\begin{aligned} EE(P) &= (Cov(P))^1 \\ Cov(\emptyset) &= 0 \\ Cov(\{\epsilon\}) &= 1 \\ Cov(P) &= Cov(P_{\epsilon_k}) \otimes ((\overline{x_k} \wedge Cov(P_{\overline{x_k}})) \vee \\ &(x_k \wedge Cov(P_{x_k}))) \\ 0 \otimes f &= f \\ * \otimes f &= * \\ 1 \otimes 1 &= * \\ f \otimes g &= (\overline{x_k} \wedge (f_{\overline{x_k}} \otimes g_{\overline{x_k}})) \vee \\ &(x_k \wedge (f_{x_k} \otimes g_{x_k})) \end{aligned}$$

Proof. We first show that $(f \otimes g)^0 = f^0 \wedge g^0$, and that $(f \otimes g)^1 = (f^1 \wedge g^0) \vee (f^0 \wedge g^1)$. Then we can prove that $Cov(P)$ is a function f from $\{0, 1\}^n$ into $\{0, 1, *\}$, such that f^0 is the set of elements of $\{0, 1\}^n$ that are not covered by any product of P , f^1 is the set of elements that are covered by a unique product of P , and f^* is the set of elements that are covered by at least two different products of P . \square

4.3 Discussion

The theorems of Sections 4.1 and 4.2 hold whatever representation is used to denote the functions and the prime sets. They show that prime and essential prime implicant computation of functions from $\{0, 1\}^n$ into $\{0, 1, *\}$ can be done by manipulating only prime sets.

The functions f from $\{0, 1\}^n$ into $\{0, 1, *\}$ can be represented with BDD's in two different ways, either using BDD's with the 3 terminal leaves 0, 1, and *, or with couples of BDD's that can be either (f^1, f^0) or (f^1, f^*) , or (f^0, f^*) . Theorems given in Sections 4.1 and 4.2 can be directly used on the meta-product representation, because set operations on meta-products can be all expressed with logical operations and Shannon expansion, that are easily evaluated with BDDs. However, this is not true with IPSs, and a specific graph calculus has been developed to realize on IPSs the operations used in the prime implicant computation. The complete description of this calculus can be found in [8].

The IPS of the set $Ess(f)$ can be computed using two strategies. The first strategy consists in first computing $EE(Prime(f))$, then building the term $C = (\neg f^* \wedge EE(Prime(f)))$, and finally evaluating $IP(Prime(f), C)$. Experience shows that the BDD of $EE(Prime(f))$ is very costly to build and that this construction is the bottleneck of the procedure. The second strategy consists in computing the term C without computing $EE(Prime(f))$, by taking into account $\neg f^*$ during the computation of C . This is done in the function $EE2(Prime(f), \neg f^*)$, where the structure of the BDD of $\neg f^*$ is used to prune the recursion tree defined by the equations given in Section 4.2. Experience shows that this strategy leads to a dramatically more efficient treatment of large examples.

5 Prime Computation of Boolean Vectorial Functions

This section we present the theorems that allow us to compute the sets of prime and essential prime implicants of partially defined vectorial Boolean functions using the procedure described in the preceding section.

Let $f = [f_1 \dots f_m]$ be a Boolean vectorial function from $\{0, 1\}^n$ into $\{0, 1, *\}^m$. By definition [13], the prime and essential prime implicants of f are the one's of the partial single-output multi-valued Boolean function $\lambda x. \lambda k. (f_k(x))$ from $\{0, 1\}^n \times \{1, \dots, m\}$ into $\{0, 1, *\}$, and computing the prime and essential prime implicants of such functions can be done in two ways.

The first way, presented in [11], consists in using the generalized definition of meta-products on the domain $\{0, 1\}^n \times \{1, \dots, m\}$, and its associated calculus that is based on Boolean equation solving. However, experience shows that this approach is limited by the computational costs of the composition and the quantified variable elimination operations [7, 11]. Though there exist recursive algorithms to cope with this computation, generalized meta-products have an

heterogeneous structure that makes their implementation costly and quite complicated.

The second way consists in fully exploit the power of the IPS based recursive algorithms we have introduced in this paper. This is done by reducing the problem of computing the prime computation of the partial Boolean vectorial function f to the one of a partial Boolean function. This is done in the following way [5].

Definition 2 Let $f = [f_1 \dots f_m]$ be a partial Boolean vectorial function from $\{0, 1\}^n$ into $\{0, 1, *\}^m$. We define the Boolean functions $\mathcal{F}^{1*}(f)$ and $\mathcal{F}^*(f)$ from $\{0, 1\}^{n+m}$ into $\{0, 1\}$ using the following equations, where $y = [y_1 \dots y_m]$:

$$\begin{aligned} \mathcal{F}^{1*}(f) &= \lambda x. \lambda y. \left(\bigwedge_{k=1}^m (\neg y_k \vee f_k^*(x) \vee f_k^1(x)) \right) \\ \mathcal{F}^*(f) &= \lambda x. \lambda y. \left(\neg \bigwedge_{k=1}^m (\neg y_k \vee f_k^*(x) \vee \bigvee_{\substack{1 \leq j \leq m \\ j \neq k}} y_j) \right) \end{aligned}$$

Definition 3 We define the function ϱ from the set $P_n \times (\{\bar{y}_1, \epsilon\} \times \dots \times \{\bar{y}_m, \epsilon\})$ onto $P_n \times 2^{\{1, \dots, m\}}$ such that $\varrho(p, q) = p \times Q$, where $k \in Q$ iff $q_k = \epsilon$.

Then we have the two following fundamental theorems [5].

Theorem 6 Let f be a partial Boolean vectorial function from $\{0, 1\}^n$ into $\{0, 1, *\}^m$. ϱ is a one-to-one mapping from $Prime(\mathcal{F}^{1*}(f)) \setminus \{\bar{y}_1 \dots \bar{y}_m\}$ onto $Prime(f)$.

Theorem 7 Let f be a partial Boolean vectorial function from $\{0, 1\}^n$ into $\{0, 1, *\}^m$. Then ϱ is a one-to-one mapping from $IP(Ess(\mathcal{F}^{1*}(f)), \mathcal{F}^*(f))$ onto $Ess(f)$.

Note that the function $IP(Ess(\mathcal{F}^{1*}(f)), \mathcal{F}^*(f))$ is equal to

$$IP(Prime(\mathcal{F}^{1*}(f)), \mathcal{F}^*(f) \wedge EE(Prime(\mathcal{F}^{1*}(f))))$$

and also equal to

$$IP(Prime(\mathcal{F}^{1*}(f)), EE2(Prime(\mathcal{F}^{1*}(f)), \mathcal{F}^*(f))).$$

These two strategies (see Section 4.3) will be compared in the next section. Using these definitions and theorems, prime computation of a partial Boolean vectorial function f essentially comes down to computing the prime implicants of the Boolean function $\mathcal{F}^{1*}(f)$, and to computing the essential prime implicants of $\mathcal{F}^{1*}(f)$ using $\mathcal{F}^*(f)$ as a filter.

6 Experimental Results

Tables 1, 2, 3 and 4 give the experimental results obtained using the implicit prime computation techniques presented here on the MCNC benchmark [15]. Table 5 gives experimental results obtained on circuits of the ISCAS benchmark. The CPU times printed in these tables are in seconds on a SUN Sparc2 workstation with 48Mb of main memory.

Each file of the MCNC benchmark describes a function $f = [f_1 \dots f_m]$ from $\{0, 1\}^n$ into $\{0, 1, *\}^m$ by giving its functions f_k^1 and f_k^* . Each file has first been translated from its original format into an intermediate form in which the Boolean functions f_k^1 and f_k^* are represented with a textual form of BDD's. This translation required 9.2 seconds for the "math" directory (23 files), 56 seconds for the "random" directory (11 files), and 99 seconds for the "indust" directory (111 files). The variable ordering used to build these BDD's is the same for all functions of a file.

The first step in the implicit prime computation method presented in this paper is to build the BDDs of the Boolean functions $\mathcal{F}^{1*}(f)$ and $\mathcal{F}^*(f)$. The variable ordering used to build these BDDs is computed using the heuristics proposed by Touati in [14]. This heuristics, that was proposed to help building transition relations of sequential circuits, applies here as well because the Boolean functions $\mathcal{F}^{1*}(f)$ and $\mathcal{F}^*(f)$ have the same characteristics as transition relations. The CPU times needed to compute these BDDs are included in the CPU times given in the tables.

Name	i/o	#Pri	#Ess	tP	tE
Z5xp1	7/10	390	8	0.3	0.5
Z9sym	9/1	1680	0	0.2	0.3
add6	12/7	8568	153	0.2	0.3
addm4	9/8	1122	24	0.9	1.4
adr4	8/5	397	35	0.2	0.3
bcd.div3	*4/4	13	9	0.2	0.2
co14	14/1	14	14	0.2	0.2
dist	8/5	401	23	0.5	0.7
f51m	8/8	561	13	0.3	0.5
lserr	*8/8	142	15	0.3	0.4
life	9/1	224	56	0.2	0.2
log8mod	8/5	105	13	0.3	0.4
m181	15/9	1636	2	0.4	0.8
mlp4	8/8	606	12	0.7	1.1
radd	8/5	397	35	0.2	0.3
rckl	32/7	302	6	0.9	1.7
rd53	5/3	51	21	0.2	0.2
rd73	7/3	211	106	0.2	0.3
root	8/5	152	9	0.3	0.4
sqr6	6/12	205	3	0.3	0.6
sym10	10/1	3150	0	0.2	0.3
tial	18/4	7145	220	8.4	16.3
z4	7/4	167	35	0.2	0.3

Table 1. The complete "math" benchmark.

For each file, column **i/o** gives the numbers of input and of output variables of the Boolean vectorial function (i.e. the values of n and m respectively), columns **#Pri** and **#Ess** give its number of prime and essential prime implicants respectively. A "*" in the column **i/o** indicates that non empty don't care sets are associated with the outputs of the circuit. The column **tE** gives the CPU times in seconds to read the file, to compute $\mathcal{F}^{1*}(f)$ and $\mathcal{F}^*(f)$, to compute the IPSs of $Prime(f)$ and of $Ess(f)$, and to count their number of products. The column **tP** gives the CPU times in seconds for the same tasks except that $Ess(f)$ is not computed.

Table 1 gives the results that have been obtained for each of the 23 files of the directory "math". All files are treated in a few seconds. Table 2 gives the results that have been obtained for each of the 11 files of the directory "random". All files are treated, in particular the files "ex1010", "test2", and "test3" for which, as far as we know, prime implicants computation has never been treated successfully before. Their numbers of prime implicants are too large to allow these functions to be ever handled by methods that perform product by product manipulations. Moreover, the implicit prime computation methods we have introduced in [6, 7] ran out of memory for "test2" and "test3".

Name	i/o	#Pri	#Ess	tP	tE
bench	*6/8	391	0	0.5	0.8
bench1	*9/9	5972	0	6.3	11.0
ex1010	*10/10	25888	0	30	56
exam	*10/10	4955	0	2.5	4.6
fout	*6/10	436	2	0.7	1.1
p1	*8/18	287	25	1.0	1.9
p3	*8/14	185	22	0.6	1.0
test1	*8/10	2407	0	2.6	5.1
test2	*11/35	109099	0	998	2318
test3	*10/35	41344	0	285	749
test4	*8/30	6139	0	17.5	75

Table 2. The complete "random" benchmark.

Table 3 and Table 4 give the results that have been obtained for a selection of the 111 functions of the directory "indust". Table 3 gives the results for the Boolean vectorial functions f whose numbers of prime implicants are already known and are larger than 1000, or for which the CPU time needed to compute the IPSs of $Prime(f)$ and $Ess(f)$, i.e. **tE**, is more than 1 second. Table 4 gives the results that have been obtained for the 16 Boolean vectorial functions of directory "indust" for which the numbers of prime implicants were unknown. All these functions have been successfully treated. Their numbers of prime implicants are very large, and in most cases much too large to allow these functions to be ever handled by methods that perform product by prod-

Name	i/o	#Pri	#Ess	tP	tE
<i>al2</i>	16/47	9179	16	1.2	1.6
<i>alcom</i>	15/38	4657	16	0.8	1.1
<i>alu2</i>	*10/8	434	36	0.8	1.6
<i>alu3</i>	*10/8	540	27	0.8	1.4
<i>b10</i>	*15/11	938	51	1.4	4.0
<i>b12</i>	15/9	1490	2	0.4	0.7
<i>b2</i>	16/17	928	54	3.0	7.0
<i>b3</i>	*32/20	3056	123	4.8	14.2
<i>b4</i>	*33/23	6455	40	5.6	14.3
<i>b9</i>	16/5	3002	48	0.5	1.3
<i>b10</i>	*15/11	938	51	1.4	3.1
<i>bc0</i>	26/11	6596	37	9.2	24.4
<i>bca</i>	*26/46	305	144	7.2	10.4
<i>bcb</i>	*26/39	255	137	4.8	7.0
<i>bcc</i>	*26/45	237	119	5.6	8.0
<i>bcd</i>	*26/38	172	100	3.8	5.3
<i>chkn</i>	29/7	671	86	1.0	2.0
<i>cps</i>	24/109	2487	57	34	54
<i>ex5</i>	8/63	2532	28	5.3	16.8
<i>ex7</i>	16/5	3002	48	0.4	1.4
<i>exp</i>	*8/18	238	30	0.9	1.5
<i>exep</i>	*30/63	558	82	9.9	22.6
<i>exps</i>	*8/38	852	56	5.2	10.0
<i>gary</i>	15/11	706	60	1.2	2.8
<i>in0</i>	15/11	706	60	1.3	2.8
<i>in1</i>	16/17	928	54	2.9	7.0
<i>in2</i>	19/10	666	85	1.1	2.1
<i>in3</i>	35/29	1114	44	4.1	12.0
<i>in4</i>	32/20	3076	118	4.2	12.7
<i>in5</i>	24/14	1067	53	1.8	4.6
<i>in6</i>	33/23	6174	40	2.7	8.1
<i>in7</i>	26/10	2112	31	1.1	3.7
<i>intb</i>	15/7	6522	186	7.3	13.0
<i>lin</i>	7/36	1087	8	4.3	9.5
<i>luc</i>	8/27	190	14	0.8	1.5
<i>m3</i>	8/16	344	4	0.7	1.3
<i>m4</i>	8/16	670	11	1.0	2.1
<i>mark1</i>	*20/31	208	1	1.4	2.5
<i>max1024</i>	10/6	1278	14	1.1	2.0
<i>max128</i>	7/24	469	6	0.9	1.9
<i>max512</i>	9/6	535	20	0.6	1.1
<i>opa</i>	17/69	477	22	6.7	9.8
<i>pope</i>	6/48	593	12	2.2	5.3
<i>prom1</i>	9/40	9326	182	54	128
<i>prom2</i>	9/21	2635	9	8.3	15.5
<i>spla</i>	*16/46	4972	33	11.4	17.4
<i>t1</i>	21/33	15135	7	2.8	4.9
<i>t2</i>	*17/16	233	25	0.7	1.3
<i>vg2</i>	25/8	1188	100	0.4	0.8
<i>vtx1</i>	27/6	1220	100	0.7	1.4
<i>x1dn</i>	27/6	1220	100	0.7	1.4
<i>x6dn</i>	39/5	916	60	0.9	3.2
<i>x9dn</i>	27/7	1272	110	0.7	1.5
<i>xparc</i>	41/73	15039	140	34	98

Table 3. 54 examples from the “indust” benchmark.

Name	i/o	#Pri	#Ess	tP	tE
<i>accpla</i>	50/69	1758057	97	359	1021
<i>ex4</i>	128/28	1.8348 $e+14$	138	1.3	2.7
<i>ibm</i>	48/17	1047948800	172	3.5	34
<i>jbp</i>	36/57	2496809	0	204	367
<i>mainpla</i>	27/54	87692	29	398	1100
<i>misg</i>	56/23	6499491839	3	0.4	0.6
<i>mish</i>	94/43	1.1243 $e+15$	3	2.7	6.4
<i>misj</i>	35/14	139103	13	0.3	0.5
<i>pdc</i>	*16/40	23231	2	164	243
<i>shift</i>	19/16	165133	100	4.5	15.4
<i>signet</i>	39/8	78735	104	207	4019
<i>soar</i>	83/94	3.3047 $e+14$	2	9.9	19.9
<i>ti</i>	47/72	836287	46	56	129
<i>ts10</i>	22/16	524280	128	14.5	27
<i>x2dn</i>	82/56	1.1488 $e+16$	2	9.7	28
<i>x7dn</i>	66/15	566698631	378	29	117

Table 4. The 16 very large examples from the “indust” benchmark.

Name	i/o	#Pri	#Ess	tP	tE
<i>cbp16</i>	33/17	6.8 $e+10$	327662	0.6	0.8
<i>cbp32</i>	65/33	2.8 $e+21$	2.1 $e+10$	1.9	2.4
<i>cont</i>	30/28	17060286	79	15.0	44
<i>mclc</i>	22/17	56895	6	0.9	1.9
<i>s298</i>	17/20	106307	36	20.4	32
<i>s344</i>	24/26	3270148	0	70	145
<i>s344.V</i>	*24/26	17116666	0	154	366
<i>s382</i>	24/27	11142527	6	1.2	2.9
<i>s526</i>	24/27	21523813	40	37	76
<i>s713.V</i>	*54/42	47461616	6	47	128
<i>s1196</i>	32/32	3365759	27	451	1288
<i>seq</i>	69/57	9.8 $e+09$	36	24.8	99
<i>add2</i>	13/7	10110	15	2.5	5.1
<i>add3</i>	21/11	3573092	15	7.8	33
<i>add4</i>	29/12	4.9 $e+08$	80	8.6	68
<i>addsub</i>	31/15	3.6 $e+09$	32794	1.5	4.6
<i>add32</i>	65/32	1.5 $e+21$	1.7 $e+10$	3.8	5.8
<i>mul06</i>	12/12	26264	13	33	52
<i>mul07</i>	14/14	163361	10	371	577
<i>mul08</i>	16/16	984384	?	6872	?
<i>pitch</i>	16/48	27560	35	15.4	40
<i>rip08</i>	16/9	182893	631	0.3	0.4

Table 5. Other Very large examples.

uct manipulations. Moreover the files “*accpla*” and “*x7dn*” cannot be treated by the implicit prime computation methods presented in [6, 7].

Table 5 gives the results that have been obtained for some other combinatorial and sequential circuits, coming from the ISCAS and MIS benchmarks. Our technique achieves to compute the set of prime implicants for all of them, which was, as far as we know, never been achieved before. However we failed in computing the essential prime implicants of the 8-bit multiplier *mul08*. For large examples we note that essential prime implicant computation induces

a quite large overhead compared to prime implicant computation alone.

Among the two strategies for evaluating $Ess(f)$ that have been presented in Section 4.2, the procedure based in the operator $EE2$ is clearly much more efficient than the one based on the operator EE . Experience shows that the latter procedure usually uses about 2 times less memory than the former, and that it is at least 2 times faster. For instance, computing $Ess(mark1)$ takes 55 seconds and 3.1 Mb with the former procedure, and only 2.5 seconds and 0.5 Mb with the latter. These CPU times and amounts of memory are respectively 909 s and 33 Mb versus 244 s and 12 Mb for *pd*, 231 s and 12 Mb versus 128 s and 12 Mb for *prom1*, 141 s and 5.2 Mb versus 28 s and 4.7 Mb for *x2dn*. The first procedure runs out of memory for *jbp*, *mainpla* and *signet*.

7 Conclusion

We have presented in this paper a new prime and essential prime computation procedure of Boolean functions that overcomes the limits of all already known procedures. The performances of this procedure result from the combination of the *implicit prime set* (IPS) representation and of its associated calculus. In particular, this procedure allows us to handle much more complex functions than the procedure based on meta-products presented in [7].

The development of this new procedure was motivated by the following remark. Meta-products of *prime sets* are very redundant, and the elimination of this redundancy produces the new canonical IPS representation dedicated to prime set representation that is much more compact than meta-products. However taking advantage of IPSs gives rise to two major problems. First there is no relation at all between set operations on IPSs and logical operations, as it was the case for meta-products. Second the essential prime implicant computation procedure presented in [7] manipulates sets of products that are not prime. This led us to the definition of the new graph calculus and the essential prime implicant computation procedure presented here.

However, though the essential prime implicant computation technique proposed in this paper is able to treat functions that are unreachable with any previously known technique, it requires prime implicant computation as a first step. Work is under way to make this essential prime implicant computation more efficient.

The IPS based calculus introduced here can also be used to compute irredundant prime covers of partial vectorial Boolean functions, and experimentations are made on such a procedure that opens the way to a new 2-level minimization algorithm.

References

- [1] S. B. Akers, "Binary Decision Diagrams", *IEEE Transactions on Computers*, Vol C-27, N°6, 1978.
- [2] R. E. Brayton, G. D. Hachtel, C. T. McMullen, A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.
- [3] R. E. Bryant, "Graph-based Algorithms for Boolean Functions Manipulation", *IEEE Trans. on Computers*, Vol C-35, 1986.
- [4] O. Coudert, J. C. Madre, "A Unified Framework for the Formal Verification of Sequential Circuits", in Proc. of *ICCAD'90*, Santa-Clara, November 1990.
- [5] O. Coudert, J. C. Madre, "Symbolic Prime Computation of Multiple-Output Boolean functions", *BULL Research Report N°91034*, November 1991.
- [6] O. Coudert, J. C. Madre, "A New Method to Compute Prime and Essential Prime Implicants of Boolean Functions", in *Advanced research in VLSI and Parallel Systems*, T. Knight and J. Savage Editors, The MIT Press, pp. 113–128, March 1992.
- [7] O. Coudert, J. C. Madre, "Implicit and Incremental Computation of Primes and Essential Primes of Boolean functions", in Proc. of *DAC'92*, June 1992.
- [8] O. Coudert, J. C. Madre, "The IPS Based Computation of Primes and Essential Primes of Boolean functions", Bull Research Report, 1992.
- [9] S. J. Hong, S. Muroga, "Absolute Minimization of Completely Specified Switching Functions", *IEEE Trans. on Computers*, Vol 40, pp. 53–65, 1991.
- [10] S. Kleene, *Introduction of Metamathematics*, Van Nostrand, 1952.
- [11] B. Lin, O. Coudert, J. C. Madre, "Symbolic Prime Generation for Multiple-Valued Functions", in Proc. of *DAC'92*, June 1992.
- [12] J. C. Madre, O. Coudert, "A Logically Complete Reasoning Maintenance System Based on Logical Constrain Solver", in Proc. of *IJCAI'91*, Sydney, Australia, August 1991.
- [13] T. Sasao, "An Application of Multiple-Valued Logic to a Design of Programmable Logic Arrays", Proc. of *ISMVL'78*, 1978.
- [14] H. J. Touati, H. Savoj, B. Lin, R. K. Brayton, A. Sangiovanni-Vincentelli, "Implicit State Enumeration of Finite State Machines using BDD's", in Proc. of *ICCAD'90*, Santa Clara CA, November 1990.
- [15] S. Yang, *Logic Synthesis and Optimization Benchmarks User Guide*, Microelectronics Center of North Carolina, January 1991.