

---

## A NEW GRAPH BASED PRIME COMPUTATION TECHNIQUE

O. Coudert      J.C. Madre

*BULL Corporate Research Center, Rue Jean Jaurès  
78340 Les Clayes-sous-bois      FRANCE*

### ABSTRACT

Computing prime and essential primes of Boolean functions is a problem that has applications in many different areas of computer science including computer design [2, 9], automated reasoning [12], and reliability analysis [8]. Though much effort has been spent on this problem over the last decades, all the prime computation techniques that have been developed so far are of limited power because they all manipulate sets of primes explicitly. This chapter presents a new prime computation procedure that overcomes this limitation because its complexity is not related to the number of primes to be computed but to the sizes of the graphs used to represent the sets of primes implicitly.

### 1.1 INTRODUCTION

A prime implicant computation technique has been recently introduced that makes possible to handle Boolean functions with sets of primes and of essential primes too large to be explicitly built [6, 7]. The key ideas that underlie this technique are to represent and to compute these sets implicitly using *metaproducts* [6] that are a canonical representation of sets of products, and to represent these metaproducts with binary decision diagrams (BDD) [4]. This technique overcomes the limitations of all previously known prime computation techniques because its cost is related to the size of the BDDs it manipulates, and not to the number of primes to be computed.

The BDDs of metaproducts that represent sets of primes and of essential primes are very redundant. The elimination of these redundancies produces dramatically smaller BDDs, and it can be done in such a way that the resulting representation, called the *Implicit Prime Set* (IPS) representation, is still canonical. However, metaproducts were defined in such a way that set operations on set on products correspond to logical operations on metaproducts, and this correspondence does not exist anymore with IPSs, so that the prime computation techniques presented in [7] cannot be implemented using IPSs because they manipulate sets of products that are not all primes and such sets cannot be represented with IPSs.

This chapter presents the implicit prime set representation and the new prime and essential prime computation procedure based on this representation. This procedure has been shown by experience to be more powerful than any previously known procedure, including the one based on metaproducts, since it can handle with success all the vectorial Boolean functions described in the MCNC benchmark [15] that include examples that had never been treated before.

This chapter 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. We assume the reader familiar with binary decision diagrams [4]. Section 3 briefly presents the metaproduct representation, explains why BDDs of metaproducts of prime sets are redundant, and then introduces the IPS representation. Section 4 presents the IPS based prime and essential prime computation of 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.

## 1.2 DEFINITIONS AND NOTATIONS

### 1.2.1 Formulas and Functions

A propositional formula built out of  $n$  propositional variables denotes a unique Boolean function from  $\{0, 1\}^n$  into  $\{0, 1\}$  [10]. A *literal* is a propositional variable  $x_k$  or its negation, also noted  $\overline{x_k}$ . We note  $\Delta(x_k, L, H)$  the function  $(\overline{x_k} \wedge L) \vee (x_k \wedge H)$ . We note  $(f_{\overline{x_k}}, f_{x_k})$  the unique couple of functions obtained using the *Shannon expansion* of  $f$  with respect to  $x_k$  [1],

A function  $f$  from a set  $E$  into  $\{0, 1\}$  denotes a unique subset of  $E$  that is

$f^{-1}(1)$ . 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 not make any 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. The sets  $f^0$ ,  $f^1$ , and  $f^*$  are often called the *off-set*, the *on-set*, and the *don't care-set* of the function  $f$  respectively. We will note  $f^{1*}$  the set  $f^{-1}(1) \cup f^{-1}(*)$ . The function  $f$  is said to be *completely defined* if  $f^*$  is empty.

## 1.2.2 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  $\{0, 1, *\}$  and  $p$  be a product built on  $E$ . The product  $p$  is an *implicant* of  $f$  iff  $p \cap f^0 = \emptyset$ . It is a *prime* 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 of  $f$  iff it is a maximal element of the set of implicants of  $f$  with respect to “ $\supseteq$ ”. Finally, the product  $p$  is an *essential prime* of  $f$  iff there exists an element  $x$  of  $f^1$  such that  $p$  is the only prime of  $f$  that contains  $x$ . In the sequel, the sets of primes and of essential primes of the function  $f$  will be noted  $Prime(f)$  and  $Ess(f)$  respectively.

In the particular case where all the sets  $E_k$  are the set  $\{0, 1\}$ , the set of products that can be built on  $E$  is noted  $P_n$ . By definition,  $P_n = \{\epsilon, \bar{x}_1, x_1\} \times \dots \times \{\epsilon, \bar{x}_n, x_n\}$ , where  $\epsilon$  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  $x_1\bar{x}_2x_4$  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  $P_{\epsilon_k}$  the subset of products of  $P$  in which none of the literals  $\bar{x}_k$  and  $x_k$  occurs. We note  $P_{\bar{x}_k}$  the set of products containing no occurrence of  $\bar{x}_k$ , such that  $\{\bar{x}_k\} \times P_{\bar{x}_k}$  is the subset of products of  $P$  in which the literal  $\bar{x}_k$  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  $P = \{x_1\bar{x}_4, x_2, x_2x_4, \bar{x}_2x_3\}$ , we have  $P_{\epsilon_2} = \{x_1\bar{x}_4\}$ ,  $P_{\bar{x}_2} = \{x_3\}$ , and  $P_{x_2} = \{\epsilon, x_4\}$ . The sets defined above allow us to build the following canonical partition of the set  $P$ :

$$P = P_{\epsilon_k} \cup (\{\bar{x}_k\} \times P_{\bar{x}_k}) \cup (\{x_k\} \times P_{x_k}).$$

### 1.2.3 *IntPro*, *Deg*, and Prime Sets

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

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

We note  $Deg(P, k)$  the set of elements of  $E$  that are contained by exactly  $k$  products of  $P$ :

$$\begin{aligned} Deg(P, 0) &= \{x \in E \mid \forall p \in P, x \notin p\} \\ Deg(P, k+1) &= \{x \in E \mid \exists p \in P, x \in p \wedge x \in Deg(P \setminus \{p\}, k)\} \end{aligned}$$

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  $P$ . In other words,  $P$  is a prime set iff 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  $\{\bar{x}_1, x_1\}$  is a set of maximal products, but it is not a prime set. In particular, for any function  $f$  from  $E$  to  $\{0, 1\}$ , 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 *IntPro* operation, and the canonical decomposition defined above. However it is not closed under the complementation, the union, or the concatenation.

## 1.3 THE IPS REPRESENTATION

In this section we briefly present the metaproduct representation and explain why it is very redundant when used to represent prime sets. Then we introduce the *implicit prime set* (IPS) representation obtained after eliminating these redundancies, and explain how the elementary set operations are realized on this representation.

### 1.3.1 Metaproducts

Metaproducts are built using the many-to-one mapping  $\sigma$  from the set  $\{0, 1\}^n \times \{0, 1\}^n$  onto the set  $P_n$  defined as follows [6]:  $\sigma([o_1 \dots o_n], [s_1 \dots s_n]) = l_1 \dots l_n$ , where  $l_k = \epsilon$  if  $o_k = 0$ ,  $l_k = \bar{x}_k$  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  $x_1\bar{x}_2x_4$ . 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.

Figure 1. From set of products to metaproducts.

As shown in Figure 1, we call *metaproduct*  $\mathcal{P}$  of a subset of products  $P$  of  $P_n$ , the characteristic function of the set  $(\bigcup_{p \in P} \sigma^{-1}(p))$ , and, by extension, the binary decision diagrams of this function. Figure 2 shows the metaproduct  $\mathcal{P}$  of the subset of products  $P = \{x_2\bar{x}_4, x_1x_3x_4, \bar{x}_1x_2\bar{x}_3x_4\}$  of  $P_4$ . Every path from the root of this BDD to the leaf 1 defines a partial assignment of the occurrence and sign variables  $o$  and  $s$ , such that  $\sigma(o, s) \in P$ . Conversely, any couple  $(o, s)$  of  $\{0, 1\}^n \times \{0, 1\}^n$ , such that  $\sigma(o, s) \in P$ , satisfies  $\mathcal{P}(o, s) = 1$ .

Figure 2. Metaproduct of the subset  $\{x_2\bar{x}_4, x_1x_3x_4, \bar{x}_1x_2\bar{x}_3x_4\}$  of  $P_4$ .

Since the collection  $(\sigma^{-1}(p))_{p \in P_n}$  is a *partition* of  $\{0, 1\}^n \times \{0, 1\}^n$ , metaproducts are a canonical functional representation of subsets of  $P_n$  [6]. For this reason, operations on sets of products correspond with logical operations on their metaproducts [6]. For any metaproducts  $\mathcal{P}$  and  $\mathcal{P}'$ , the function  $(\mathcal{P} \vee \mathcal{P}')$  is the metaproduct of the union of the sets of products  $P$  and  $P'$  denoted by  $\mathcal{P}$  and  $\mathcal{P}'$  respectively;  $(\neg \mathcal{P})$  is the metaproduct of  $(P_n \setminus P)$ ; the set  $P$  is included in the set  $P'$  iff  $(\mathcal{P} \Rightarrow \mathcal{P}') = 1$ ; finally, the metaproduct of  $P_{\epsilon_k}$  is  $(\neg o_k \wedge \mathcal{P}_{\overline{o_k}})$ , the metaproduct of  $P_{\overline{x_k}}$  is  $(o_k \wedge \neg s_k \wedge \mathcal{P}_{o_k \overline{s_k}})$ , and the metaproduct of  $P_{x_k}$  is  $(o_k \wedge s_k \wedge \mathcal{P}_{o_k s_k})$ .

The properties given above make many operations on sets of products, for instance counting the number of elements in a set of products, linear with respect to the size of the metaproducts of these sets, if these BDDs are built with the variable ordering:

$$o_{\pi(1)} < s_{\pi(1)} < o_{\pi(2)} < s_{\pi(2)} < \cdots < o_{\pi(n)} < s_{\pi(n)},$$

where  $\pi$  is a permutation of the integers  $\{1, \dots, n\}$  [6]. In the following we will consider that the metaproducts are always built with such a variable ordering, and moreover, that the permutation defining this ordering is the same as the one defining the variable ordering used to build the BDD of the function under treatment.

### 1.3.2 Implicit Prime Sets

In this section we first give a theorem showing that the metaproducts have properties that make them very redundant when used to represent prime sets. Then we explain how these redundancies can be eliminated from these metaproducts to produce the implicit prime set representation.

**Theorem 1** *Consider a prime set  $P$  and its metaproduct  $\mathcal{P}$ . Then for any  $k$ , the three following properties hold:*

$$\mathcal{P}_{\overline{o_k s_k}} = \mathcal{P}_{\overline{o_k} s_k} \quad (1)$$

$$(\mathcal{P}_{\overline{o_k}} \wedge \mathcal{P}_{o_k}) = 0 \quad (2)$$

$$(\mathcal{P}_{o_k \overline{s_k}} \wedge \mathcal{P}_{o_k s_k}) = 0 \quad (3)$$

*Proof.* Suppose that (1) is false, for instance for  $k = 1$ . Then there exist  $o'$  and  $s'$  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  $\sigma$  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. Note that this property holds for any metaproduct.

Now suppose that (2) is false, for instance for  $k = 1$ . Then there exist  $o' \in \{0, 1\}^{n-1}$  and  $s \in \{0, 1\}^n$  such that  $\mathcal{P}(0o', s) = \mathcal{P}(1o', s) = 1$ . Let  $p = \sigma(0o', s)$ . Both products  $p$  and  $l_1p$ , where  $l_1 = \bar{x}_1$  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  $(\bigcup_{p \in P} p) \supseteq p \supset l_1p$ .

Finally suppose that (3) is false, for instance 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  $q = \sigma(0o', 0s')$ . Both the products  $x_1q$  and  $\bar{x}_1q$  belong to  $P$ , which implies that  $(\bigcup_{p \in P} p) \supseteq (x_1q \cup \bar{x}_1q) = q \supset x_1q$ , and so  $P$  is not a prime set.  $\square$

The consequence of property (1) is that, for any path in a metaproduct 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. The consequence of property (2) is that all the occurrence variables occur on every path from the root of the metaproduct of a prime set to the leaf 1. The consequence of property (3) is that the sign variable  $s_k$  occurs on every path from the root of the metaproduct of a prime set to the leaf 1 on which the occurrence variable  $o_k$  is set to 1.

**Definition 1** *The implicit prime set (IPS) of a prime set  $P$  is the BDD obtained after having applied the two following reduction rules on the metaproduct of  $P$ . The rule (R1) applies on any subgraph of this BDD denoting a set of products in which the variable  $x_k$  never occurs, and the rule (R2) applies on any subgraph of this BDD denoting a set of products in which the variable  $x_k$  always occurs.*

*Figure 3. Reduction rules transforming metaproducts into IPSs.*

**Theorem 2** *Implicit prime sets are a canonical representation of prime sets.*

The proof of canonicity of the implicit prime set representation is done by showing that properties (1), (2), and (3) given above are sufficient to build the metaproduct of a prime set from its IPS. This proof is based on the following theorem.

**Theorem 3** *Let  $\mathcal{P}$  be an IPS built with the variable ordering  $o_1 < s_1 < \dots < o_n < s_n$ . Then the evaluation of  $\mathcal{IPStoMP}(\mathcal{P}, 1)$  returns the metaproduct of the prime set denoted by the IPS  $\mathcal{P}$ .*

```

function  $\mathcal{IPStoMP}(\mathcal{P} : \text{IPS}, k : \text{int}) : \text{BDD};$ 
if  $\mathcal{P} = 0$  return 0;
if  $\mathcal{P} = 1$ 
  if  $k = n + 1$ 
    return 1;
  else
    return  $\Delta(o_k, \mathcal{IPStoMP}(1, k + 1), 0);$            /* case (c) */
let  $\Delta(v_{k'}, L, H) = \mathcal{P}$  in
  if  $k < k'$ 
    return  $\Delta(o_k, \mathcal{IPStoMP}(\mathcal{P}, k + 1), 0);$        /* case (c) */
  else if  $v = o$ 
    return  $\Delta(o_k, \mathcal{IPStoMP}(L, k + 1), \mathcal{IPStoMP2}(H));$  /* case (a) */
  else
    return  $\Delta(o_k, 0, \mathcal{IPStoMP2}(\mathcal{P}));$              /* case (b) */

function  $\mathcal{IPStoMP2}(\mathcal{P} : \text{IPS}) : \text{BDD};$ 
let  $\Delta(s_k, L, H) = \mathcal{P}$  in
  return  $\Delta(s_k, \mathcal{IPStoMP}(L, k + 1), \mathcal{IPStoMP}(H, k + 1));$ 

```

Figure 4. Building the metaproduct of a prime set from its IPS.

*Proof.* Consider a path from the root of the IPS  $\mathcal{P}$  to the leaf 1. For any  $k$ , one and only one of the following cases can occur: (a)  $o_k$  is on the path ; (b)  $o_k$  is not on the path but  $s_k$  is ; (c) neither  $o_k$  nor  $s_k$  are on the path. In the case (a), the path assigns the variable  $o_k$  a value, and if this value is 1, then the variable  $s_k$  is also on the path, so the function does not modify the values of these variables. If this value is 0, then the variable  $s_k$  is not on the path, and it does not have to be valued. In the case (b), the path does not satisfy the property (2) given in theorem 1. However the properties (1) and (3) given in this theorem state that  $s_k$  occurs on a path of a metaproduct if and only if this path assigns  $o_k$  to 1. This means that the reduction rule (R2) has been effective, therefore  $o_k$  must be set to 1. Finally, in the case (c), the path does not satisfy the property (2) but since the variable  $s_k$  does not occur on this path, this means that the reduction rule (R1) has been effective, and  $o_k$  must be then set to 0.

When the term  $\mathcal{IPStoMP}(\mathcal{P}, k)$  is evaluated, the function  $\mathcal{IPStoMP}$  expects

that the root  $v_{k'}$  of the BDD  $\mathcal{P}$  is the occurrence variable  $o_k$ . If this is not the case, then one of the two reduction rules has been effective, and the function tests whether the current root variable is  $s_k$ , in which case it was the rule (R2) that had been applied on the metaproduct, or whether the current root variable has an order greater than both  $o_k$  and  $s_k$  ( $k \downarrow k'$ ), in which case it was the rule (R1) that had been applied. In all cases the function *IPStoMP* rebuilds without ambiguity the metaproduct of the set denoted by  $\mathcal{P}$ .  $\square$

Figure 5 shows the metaproduct with 16 vertices of the prime set  $\{x_2\bar{x}_4, x_1x_3x_4, \bar{x}_1x_2\bar{x}_3x_4\}$  of  $P_4$  and its IPS with 8 vertices, both of them built with the variable ordering  $o_1 < s_1 < \dots < o_4 < s_4$ . The experimental results given in the sequel actually show that the ratio between the size of the metaproduct of a prime set and the size of its IPS can be up to 10 for some examples, which means that this new representation is dramatically more compact than the metaproduct representation.

Figure 5. Metaproduct and IPS of the set  $\{x_2\bar{x}_4, x_1x_3x_4, \bar{x}_1x_2\bar{x}_3x_4\}$ .

The way IPSs must be interpreted is illustrated in Figure 5. The path marked with dotted lines in the IPS defines the following partial assignment:  $o_1 = 0$ ,  $s_2 = 1$ , and  $s_4 = 0$ . The variable  $o_1$  occurs in the path, which corresponds to case (a). The variable  $s_2$  occurs on the path but not the variable  $o_2$  (case (b)), so the variable  $o_2$  must be set to 1. The variables  $o_3$  and  $s_3$  do not occur on this path (case (c)), so the variable  $o_3$  must be set to 0. Finally the variable  $s_4$  occurs on the path but not the variable  $o_4$  (case (b)), so the variable  $o_4$  must be set to 1. This path thus defines the assignment  $o_1 = 0$ ,  $o_2 = 1$ ,  $s_2 = 1$ ,  $o_3 = 0$ ,  $o_4 = 1$ , and  $s_4 = 0$  which denotes the product  $x_2\bar{x}_4$ , which is indeed an element of the set of products denoted by this IPS, and it correspond to the path marked with dotted lines in the metaproduct.

### 1.3.3 Fundamental Operations on IPSs

Metaproducts have been defined in such a way that operations on sets on products correspond with logical operations on metaproducts. This correspondence does not exist anymore with IPSs so that realizing the set operations on this representation must be done using a dedicated calculus. This section shows an element of this calculus, the function *Minus*, that computes the IPS denoting the difference of two prime sets from the IPSs of these sets.

```

function Minus( $\mathcal{P} : \text{IPS}, \mathcal{P}' : \text{IPS}$ ) : IPS;
if  $\mathcal{P} = \mathcal{P}'$  or  $\mathcal{P} = 0$  return 0;
if  $\mathcal{P} = 1$  or  $\mathcal{P}' = 0$  or  $\mathcal{P}' = 1$  return  $\mathcal{P}$ ;
let  $\Delta(v_k, L, H) = \mathcal{P}$  and  $\Delta(v_{k'}, L', H') = \mathcal{P}'$  in
  if  $k < k'$ 
    if  $v = o$ 
      return Norme( $o_k, \text{Minus}(L, \mathcal{P}'), H$ );
    else
      return  $\mathcal{P}$ ;
  if  $k > k'$ 
    if  $v' = o$ 
      return Minus( $\mathcal{P}, L'$ );
    else
      return  $\mathcal{P}$ ;
  if  $v = o$ 
    if  $v' = o$ 
      return Norme( $o_k, \text{Minus}(L, L'), \text{Minus2}(H, H')$ );
    else
      return Norme( $o_k, L, \text{Minus2}(H, \mathcal{P}')$ );
  else
    if  $v' = o$ 
      return Minus2( $\mathcal{P}, H'$ );
    else
      return  $\Delta(s_k, \text{Minus}(L, L'), \text{Minus}(H, H'))$ ;

function Minus2( $\mathcal{P} : \text{IPS}, \mathcal{P}' : \text{IPS}$ ) : IPS;
if  $\mathcal{P} = \mathcal{P}'$  return 0;
let  $\Delta(s_k, L, H) = \mathcal{P}$  and  $\Delta(s_k, L', H') = \mathcal{P}'$  in
  return  $\Delta(s_k, \text{Minus}(L, L'), \text{Minus}(H, H'))$ ;

```

Figure 6. Evaluating the set difference on IPSs.

The evaluation scheme of the function *Minus*, which is the same as the one of the functions realizing the other set operations on IPSs, consists of traversing in parallel the two BDDs to be combined, and in building the resulting BDD in a bottom-up way. At each recursion, the function *Minus* expects the roots of the two BDDs to be the same occurrence variable, but, due to the reduction rules (R1) and (R2), it actually has to consider 9 different cases that can be reduced to 7 different cases, as shown in Figure 6. By using a cache where partial results are stored, redundant computations can be avoided, and the complexity of *Minus* is in  $O(|\mathcal{P}| \times |\mathcal{P}'|)$ , where  $|\mathcal{P}|$  and  $|\mathcal{P}'|$  are the numbers of vertices in the IPSs  $\mathcal{P}$  and  $\mathcal{P}'$  respectively.

```

function Norme( $x$  : var,  $\mathcal{P}$  : IPS,  $\mathcal{P}'$  : IPS) : IPS;
if  $\mathcal{P} = \mathcal{P}'$  or  $\mathcal{P}' = 0$  return  $\mathcal{P}$ ;
if  $\mathcal{P} = 0$  return  $\mathcal{P}'$ ;
return  $\Delta(x, \mathcal{P}, \mathcal{P}')$ ;

```

Figure 7. The function *Norme*.

The normalization function *Norme*, shown in Figure 7, takes as input a variable and two branches. It tests whether one of the two reduction rules can be applied, and in this case it returns the non zero branch. If none of the rules applies, it creates a new vertex according to the construction rules of BDDs [4]. Thanks to this normalization function, no useless node, i.e., node that would be eliminated by the reduction rules, is created.

## 1.4 PRIME COMPUTATION OF BOOLEAN FUNCTIONS

In this section, we show how the sets of primes and of essential primes of a partial Boolean function can be computed using manipulations of prime sets.

### 1.4.1 Prime Computation

Let  $f$  be a incompletely defined function from  $\{0, 1\}^n$  into  $\{0, 1, *\}$ . By definition,  $Prime(f) = Prime(f^{1*})$ , so it is sufficient to deal with prime computation of completely defined Boolean functions. The theorem below shows that manipulations of prime sets are sufficient to compute, using a recursive bottom-up scheme, the set  $Prime(f)$  [2]. Note that the set of useful primes, i.e., that contain at least one element of the care set, is  $IntPro(Prime(f), f^1)$ .

**Theorem 4** *The set of primes  $Prime(f)$  of the Boolean function  $f$  from  $\{0, 1\}^n$  into  $\{0, 1\}$  is defined by:*

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

It is immediate, using the theorem above, to compute in a recursive way, the IPS (or the metaproduct) of the set of primes of a Boolean function from the BDD of this function. This computation scheme is made explicit by Figure 8 showing the recursive rules that can be used to produce this BDD. When using IPSs for representing the sets of primes, these rules are written:

$$\begin{aligned} Prime(0) &= 0 \\ Prime(1) &= 1 \\ Prime(f) &= Norme(o_k, Prime(f_{\overline{x_k}} \wedge f_{x_k})) \\ &\quad \Delta(s_k, Minus(Prime(f_{\overline{x_k}}), Prime(f_{\overline{x_k}} \wedge f_{x_k})), \\ &\quad \quad Minus(Prime(f_{x_k}), Prime(f_{\overline{x_k}} \wedge f_{x_k}))) \end{aligned}$$

Note that these rules are different from the rules used to build the metaproduct of this set, for instance the set of products  $\{\epsilon\}$  of  $P_n$  is represented by the IPS 1, while it is represented by the metaproduct  $\neg(\bigvee_{k=1}^n o_k)$ .

Figure 8. Recursive rules for prime computation with metaproducts.

### 1.4.2 Essential Prime Computation

The essential prime computation technique presented in [7] is based on the same recursion scheme as the prime computation scheme presented above. In this

chapter we present a new computation technique that makes use of a completely different computation scheme based on the following theorems.

**Theorem 5** *The set of essential primes  $Ess(f)$  of the function  $f$  from  $\{0, 1\}^n$  into  $\{0, 1, *\}$  is equal to:*

$$Ess(f) = IntPro(Prime(f^{1*}), f^1 \wedge Deg(Prime(f^{1*}), 1))$$

**Theorem 6** *Let  $P$  be a set of products, and  $f$  be a Boolean function. The following equations [7] are sufficient to compute the set  $IntPro(P, f)$ :*

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

**Theorem 7** *Let  $P$  be a set of products. The following equations are sufficient to compute  $Deg(P, 0)$  and  $Deg(P, 1)$ :*

$$\begin{aligned} Deg(\emptyset, 0) &= 1 \\ Deg(\emptyset, 1) &= 0 \\ Deg(\{\epsilon\}, 0) &= 0 \\ Deg(\{\epsilon\}, 1) &= 1 \\ Deg(P, 0) &= \Delta(x_k, \\ &\quad Deg(P_{\epsilon_k}, 0) \wedge Deg(P_{\overline{x_k}}, 0), \\ &\quad Deg(P_{\epsilon_k}, 0) \wedge Deg(P_{x_k}, 0)) \\ Deg(P, 1) &= \Delta(x_k, \\ &\quad (Deg(P_{\epsilon_k}, 1) \wedge Deg(P_{\overline{x_k}}, 0)) \vee (Deg(P_{\epsilon_k}, 0) \wedge Deg(P_{\overline{x_k}}, 1)), \\ &\quad (Deg(P_{\epsilon_k}, 1) \wedge Deg(P_{x_k}, 0)) \vee (Deg(P_{\epsilon_k}, 0) \wedge Deg(P_{x_k}, 1))) \end{aligned}$$

Figure 10 shows the recursive rules that can be used to build the metaproduct of  $IntPro(\mathcal{P}, f)$  from the BDDs of  $\mathcal{P}$  and  $f$ , and Figure 9 shows the algorithm for evaluating the function  $IntPro$  on IPSs. The function  $Deg$  is based on the recursive rules shown in Figure 11 that are formalized using the following theorem.

```

function IntPro( $\mathcal{P} : \text{IPS}$ ,  $f : \text{BDD}$ ) :  $\text{IPS}$ ;
if  $f = 0$  or  $\mathcal{P} = 0$  return 0;
if  $f = 1$  or  $\mathcal{P} = 1$  return  $\mathcal{P}$ ;
let  $\Delta(v_k, L, H) = \mathcal{P}$  and  $\Delta(x_{k'}, L', H') = f$  in
  if  $k > k'$ 
    return IntPro( $\mathcal{P}, L' \vee H'$ );
  if  $v = s$ 
    return IntPro2( $\mathcal{P}, f$ );
  if  $k = k'$ 
    return Norme( $o_k, \text{IntPro}(L, L' \vee H'), \text{IntPro2}(H, f)$ );
  return Norme( $o_k, \text{IntPro}(L, f), \text{IntPro2}(H, f)$ );

function IntPro2( $\mathcal{P} : \text{IPS}$ ,  $f : \text{BDD}$ ) :  $\text{IPS}$ ;
let  $\Delta(s_k, L, H) = \mathcal{P}$  and  $\Delta(x_{k'}, L', H') = f$  in
  if  $k = k'$ 
    return  $\Delta(s_k, \text{IntPro}(L, L'), \text{IntPro}(H, H'))$ ;
  else
    return  $\Delta(s_k, \text{IntPro}(L, f), \text{IntPro}(H, f))$ ;

```

Figure 9. Evaluating the operator *IntPro* on IPSs.Figure 10. Evaluating *IntPro* on metaproducts.Figure 11. Evaluating *Deg* on metaproducts.

The IPS of the set  $Ess(f)$  can be computed using two strategies. The first strategy consists of computing  $Deg(Prime(f^{1*}), 1)$ , then of building the term  $T = (f^1 \wedge Deg(Prime(f^{1*}), 1))$ , and finally of evaluating  $IntPro(Prime(f^{1*}), T)$ . Experience shows that the BDD of  $Deg(Prime(f^{1*}), 1)$  is very costly to build and that this construction is the bottleneck in the procedure. The second strategy consists of computing the term  $T$  without computing  $Deg(Prime(f^{1*}), 1)$ , by taking into account  $f^1$  during the computation of  $C$ . This is done in the function  $Deg'(Prime(f^{1*}), f^1)$ , in which the BDD of  $f^1$  is traversed in parallel of the BDD  $\mathcal{P}$  and is used to prune the recursion tree defined by the equations given above. Experience shows that this strategy leads to a dramatically more efficient treatment of large examples.

## 1.5 PRIME COMPUTATION OF BOOLEAN VECTORIAL FUNCTIONS

This section presents the theorems that allow us to compute the sets of primes and of essential primes 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 primes and essential primes of  $f$  are the ones 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 primes and essential primes of such functions can be done in two ways.

The first way, presented in [11], consists of using the generalized definition of metaproducts on the domain  $\{0, 1\}^n \times \{1, \dots, m\}$ , which is similar to the positional cube notation, and its associated calculus 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 metaproducts have an heterogeneous structure that makes their implementation costly and quite complicated.

The second way consists of fully exploiting the power of the IPS based recursive algorithms we have introduced in this chapter. This is done by reducing the problem of 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  $f = [f_1 \cdots f_m]$  be a partial Boolean vectorial function from  $\{0, 1\}^n$  into  $\{0, 1, *\}^m$ . We define the partial Boolean function  $\mathcal{F}(f)$  from  $\{0, 1\}^{n+m}$  into  $\{0, 1, *\}$  using the following equations defining the functions  $\mathcal{F}^{1*}(f)$  and  $\mathcal{F}^1(f)$ , where  $y = [y_1 \dots y_m]$ :

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

The function  $\mathcal{F}(f)$  is always well defined, because the formula  $\mathcal{F}^1(f) \Rightarrow \mathcal{F}^{1*}(f)$  is a tautology.

**Definition 3** We define the function  $\varrho$  from the set  $P_n \times \{\overline{y_1}, \epsilon\} \times \cdots \times \{\overline{y_m}, \epsilon\}$  onto  $P_n \times 2^{\{1, \dots, m\}}$  such that  $\varrho(p, q) = p \times Q$ , where  $k \in Q$  iff  $q_k = \epsilon$ .

The variable  $y$  that ranges over  $\{0, 1\}^m$  encodes a subset of  $\{1, \dots, m\}$ . When  $y$  denotes the singleton  $\{k\}$ , i.e., when  $y_k = 1$  and  $y_j = 0$  for  $j \neq k$ , then  $\mathcal{F}(f)(x, y) = f_k(x)$ . So the function  $\mathcal{F}(f)$  is nothing but an extension of the function  $\lambda x. \lambda k. (f_k(x)) : \{0, 1\}^n \times \{1, \dots, m\} \rightarrow \{0, 1, *\}$  on the domain  $\{0, 1\}^n \times 2^{\{1, \dots, m\}}$ , where the latter is identified to the space  $\{0, 1\}^n \times \{0, 1\}^m$ . One can see that with this extension, two notions are preserved through the one-to-one mapping  $\varrho$ . Firstly, the containment relation on products built on the space  $\{0, 1\}^n \times \{1, \dots, m\}$  and the one on products built on the space  $\{0, 1\}^n \times 2^{\{1, \dots, m\}}$ . Secondly, the notion of implicant of the function  $\lambda x. \lambda k. (f_k(x))$  and the one of implicant of the function  $\mathcal{F}(f)$ . The only slight difference is that any product  $p \times \overline{y_1} \dots \overline{y_m}$  with  $p \in P_n$  denotes, thanks to the mapping  $\varrho$ , the empty product in the space  $\{0, 1\}^n \times \{1, \dots, m\}$ , which is not considered as an implicant of the function  $\lambda x. \lambda k. (f_k(x))$ . Therefore we have the two following fundamental theorems [5].

**Theorem 8** Let  $f$  be a partial Boolean vectorial function from  $\{0, 1\}^n$  into  $\{0, 1, *\}^m$ .  $\varrho$  is a one-to-one mapping from  $\text{Prime}(\mathcal{F}(f)) \setminus \{\overline{y_1} \dots \overline{y_m}\}$  onto  $\text{Prime}(f)$ .

**Theorem 9** Let  $f$  be a partial Boolean vectorial function from  $\{0, 1\}^n$  into  $\{0, 1, *\}^m$ . Then  $\varrho$  is a one-to-one mapping from  $\text{Ess}(\mathcal{F}(f))$  onto  $\text{Ess}(f)$ .

Using these definitions and theorems, prime computation of a partial Boolean vectorial function  $f$  essentially comes down to prime computation of the function  $\mathcal{F}(f)$ . Note that the term  $Ess(\mathcal{F}(f))$  is equal to

$$IntPro(Prime(\mathcal{F}^{1*}(f)), \mathcal{F}^1(f) \wedge Deg(Prime(\mathcal{F}^{1*}(f)), 1)),$$

and also equal to

$$IntPro(Prime(\mathcal{F}^{1*}(f)), Deg'(Prime(\mathcal{F}^{1*}(f)), \mathcal{F}^1(f))).$$

The two strategies to evaluate  $Ess(f)$  will be compared in the next section.

## 1.6 EXPERIMENTAL RESULTS

Tables 1, 2, 3 and 4 give the experimental results obtained using the prime computation technique presented in this chapter on the problems of the MCNC benchmark [15]. Table 5 gives experimental results obtained on circuits from the ISCAS-85 [3] and MIS [15] benchmarks. The CPU times given in these tables are in seconds on a SUN SPARC Station 2 with 48Mb of main memory.

Each file of the MCNC benchmark describes a vectorial function  $f = [f_1 \dots f_m]$  from  $\{0, 1\}^n$  into  $\{0, 1, *\}^m$ , and each function  $f_k$  is specified using the 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 BDDs. 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 BDDs is the same for all functions of a file.

Name	i/o	#P	#E	T <sub>P</sub>	T <sub>E</sub>	Name	i/o	#P	#E	T <sub>P</sub>	T <sub>E</sub>
<i>bench</i>	*6/8	391	0	0.5	0.8	<i>p3</i>	*8/14	185	22	0.6	1.0
<i>bench1</i>	*9/9	5972	0	6.3	10.5	<i>test1</i>	*8/10	2407	0	2.6	5.1
<i>ex1010</i>	*10/10	25888	0	29	56	<i>test2</i>	*11/35	109099	0	998	2318
<i>exam</i>	*10/10	4955	0	2.5	4.6	<i>test3</i>	*10/35	41344	0	285	749
<i>fout</i>	*6/10	436	2	0.7	1.1	<i>test4</i>	*8/30	6139	0	17.5	75
<i>p1</i>	*8/18	287	25	1.0	1.9						

Table 1. The complete “random” benchmark.

First step in the implicit prime computation method presented in this chapter is to build the BDDs of the Boolean functions  $\mathcal{F}^{1*}(f)$  and  $\mathcal{F}^1(f)$ . The variable

ordering used to build these BDDs is computed using the heuristics proposed by Touati in [14]. This heuristics, that was proposed to build transition relations of sequential circuits, applies here as well because the Boolean functions  $\mathcal{F}^{1*}(f)$  and  $\mathcal{F}^1(f)$  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	#P	#E	T <sub>P</sub>	T <sub>E</sub>	Name	i/o	#P	#E	T <sub>P</sub>	T <sub>E</sub>
<i>Z5xp1</i>	7/10	390	8	0.3	0.5	<i>m181</i>	15/9	1636	2	0.4	0.8
<i>Z9sym</i>	9/1	1680	0	0.2	0.3	<i>mlp4</i>	8/8	606	12	0.7	1.1
<i>add6</i>	12/7	8568	153	0.2	0.3	<i>radd</i>	8/5	397	35	0.2	0.3
<i>addm4</i>	9/8	1122	24	0.9	1.4	<i>rckl</i>	32/7	302	6	0.9	1.7
<i>adr4</i>	8/5	397	35	0.2	0.3	<i>rd53</i>	5/3	51	21	0.2	0.2
<i>bcd.div3</i>	*4/4	13	9	0.2	0.2	<i>rd73</i>	7/3	211	106	0.2	0.3
<i>co14</i>	14/1	14	14	0.2	0.2	<i>root</i>	8/5	152	9	0.3	0.4
<i>dist</i>	8/5	401	23	0.5	0.7	<i>sqr6</i>	6/12	205	3	0.3	0.6
<i>f51m</i>	8/8	561	13	0.3	0.5	<i>sym10</i>	10/1	3150	0	0.2	0.3
<i>lserr</i>	*8/8	142	15	0.3	0.4	<i>tial</i>	18/4	7145	220	8.4	16.1
<i>life</i>	9/1	224	56	0.2	0.2	<i>z4</i>	7/4	167	35	0.2	0.3
<i>log8mod</i>	8/5	105	13	0.3	0.4						

Table 2. The complete “math” benchmark.

For each function, the column **i/o** gives the numbers of input and of output variables (i.e., the values of  $n$  and  $m$  respectively), and a “\*” in this column indicates that this function is partially defined, the columns **#P** and **#E** give its number of primes and essential primes respectively, the column **T<sub>P</sub>** gives the CPU time needed to read the BDD description of the function, to compute the BDDs of  $\mathcal{F}^{1*}(f)$  and  $\mathcal{F}^1(f)$ , to compute the IPS of  $Prime(f)$  and to compute the size of this set, and finally the column **T<sub>E</sub>** is the CPU time needed to perform all these task and in addition, to compute the IPS of  $Ess(f)$ , and to compute the size of this set.

Table 1 gives the results obtained for the 11 files of the directory “random”, and Table 2 gives the results obtained for the 23 files of the directory “math”. All these functions can be treated, in particular the functions “*ex1010*”, “*test2*”, and “*test3*” for which, as far as we know, prime computation had never been achieved successfully before by procedures manipulating primes individually, which is understandable because of the very large number of primes of these functions. Note that the prime computation methods based on metaproducts introduced in [6, 7] ran out of memory for the functions “*test2*” and “*test3*”.

Table 3 and Table 4 give the results 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 primes are already known and are larger than 1000, or for which the CPU time needed to compute the IPSs of  $Prime(f)$  and  $Ess(f)$  is larger than 1.3 seconds. Table 4 gives the results obtained for the 16 Boolean vectorial functions of directory “indust” for which the numbers of primes were unknown so far. All these functions have been successfully treated. Their numbers of primes are very large, and in most cases much too large to allow these functions to be ever handled by methods that perform product by product manipulations. Moreover the functions “*acpla*” and “*x7dn*” could not be treated by the implicit prime computation methods presented in [6, 7].

Name	i/o	#P	#E	T <sub>P</sub>	T <sub>E</sub>	Name	i/o	#P	#E	T <sub>P</sub>	T <sub>E</sub>
<i>al2</i>	16/47	9179	16	1.2	1.6	<i>in1</i>	16/17	928	54	2.9	7.0
<i>alcom</i>	15/38	4657	16	0.8	1.1	<i>in2</i>	19/10	666	85	1.1	2.1
<i>alu2</i>	*10/8	434	36	0.8	1.6	<i>in3</i>	35/29	1114	44	4.1	11.2
<i>alu3</i>	*10/8	540	27	0.8	1.4	<i>in4</i>	32/20	3076	118	4.2	12.7
<i>b10</i>	*15/11	938	51	1.4	4.0	<i>in5</i>	24/14	1067	53	1.8	4.6
<i>b12</i>	15/9	1490	2	0.4	0.7	<i>in6</i>	33/23	6174	40	2.7	8.1
<i>b2</i>	16/17	928	54	3.0	7.0	<i>in7</i>	26/10	2112	31	1.1	3.7
<i>b3</i>	*32/20	3056	123	4.8	14.2	<i>intb</i>	15/7	6522	186	7.3	13.0
<i>b4</i>	*33/23	6455	40	5.6	13.9	<i>lin</i>	7/36	1087	8	4.3	9.5
<i>b9</i>	16/5	3002	48	0.5	1.3	<i>m4</i>	8/16	670	11	1.0	2.1
<i>b10</i>	*15/11	938	51	1.4	3.1	<i>mark1</i>	*20/31	208	1	1.4	2.5
<i>bc0</i>	26/11	6596	37	9.2	21.8	<i>max1024</i>	10/6	1278	14	1.1	2.0
<i>bca</i>	*26/46	305	144	7.2	10.4	<i>opa</i>	17/69	477	22	6.7	9.8
<i>bcb</i>	*26/39	255	137	4.8	7.0	<i>pope</i>	6/48	593	12	2.2	5.3
<i>bcc</i>	*26/45	237	119	5.6	8.0	<i>prom1</i>	9/40	9326	182	54	123
<i>bcd</i>	*26/38	172	100	3.8	5.3	<i>prom2</i>	9/21	2635	9	8.3	15.5
<i>chkn</i>	29/7	671	86	1.0	2.0	<i>spla</i>	*16/46	4972	33	11.4	14.3
<i>cps</i>	24/109	2487	57	34	54	<i>t1</i>	21/33	15135	7	2.8	4.9
<i>ex5</i>	8/63	2532	28	5.3	16.8	<i>t2</i>	*17/16	233	25	0.7	1.3
<i>ex7</i>	16/5	3002	48	0.4	1.4	<i>vg2</i>	25/8	1188	100	0.4	0.8
<i>exp</i>	*8/18	238	30	0.9	1.5	<i>x1dn</i>	27/6	1220	100	0.7	1.4
<i>xep</i>	*30/63	558	82	7.9	10.0	<i>x6dn</i>	39/5	916	60	0.9	3.2
<i>exps</i>	*8/38	852	56	5.2	10.2	<i>x9dn</i>	27/7	1272	110	0.7	1.5
<i>in0</i>	15/11	706	60	1.3	2.8	<i>xparc</i>	41/73	15039	140	34	84

Table 3. 48 examples from the “indust” benchmark.

Table 5 gives the results that have been obtained for some other vectorial Boolean functions from the ISCAS-85 [3] and the MIS [15] benchmarks. These

Name	i/o	#P	#E	T <sub>P</sub>	T <sub>E</sub>
<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	1047948792	172	3.5	30
<i>jbp</i>	36/57	2496809	0	184	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>pdn</i>	*16/40	23231	2	129	214
<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	8.0	18.2
<i>ti</i>	47/72	836287	46	52	115
<i>ts10</i>	22/16	524280	128	14.4	27
<i>x2dn</i>	82/56	1.1488 e+16	2	9.7	28
<i>x7dn</i>	66/15	566698632	378	29	117

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

functions are either the output functions of combinational circuits or the transition functions of sequential circuits. The functions coming from the ISCAS-85 benchmark are described by multi level networks which makes very difficult to apply on them computation methods such as the one presented in [2] because this method requires the functions to be given as sums of products. Note that it has not been possible to compute the essential primes of the 8-bit multiplier *mul08*. Note also that for large examples essential prime computation is much more costly than computing only the primes.

The essential prime computation procedure based on the operator *Deg'* was shown by experience to be much more efficient than the one based on the operator *Deg*. The former procedure uses in most cases about 2 times less memory than the latter, and it is at least 2 times faster. For instance, computing *Ess(mark1)* takes 55 seconds and 3.1 Mb with the latter procedure, and only 2.5 seconds and 0.5 Mb with the former. These CPU times and amounts of memory are respectively 909 s and 33 Mb versus 244 s and 12 Mb for *pdn*, 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 second procedure ran out of memory for *jbp*, *mainpla* and *signet*.

Experimental results show that IPSs are generally at least two times smaller than their corresponding metaproducts. Table 6 gives a comparison for some of examples appearing in the preceding tables. For each of these examples, this

table gives in column **#P** the number of primes of the function, in column **|MP|** the size of the metaproduct denoting these primes, in column **|IPS|** the size of the corresponding IPS, and in column **ratio** the ratio between these two sizes. This table shows that the ratio between the sizes of the metaproduct and its corresponding IPS is greater than 2, and is more than 8 and 10 for the examples *OutOfStage* and *accpla* respectively. This ratio is similar for the CPU times needed to compute these BDDs.

Name	i/o	#P	#E	T <sub>P</sub>	T <sub>E</sub>
<i>cbp16</i>	33/17	6.8751 e+10	327662	0.6	0.8
<i>cbp32</i>	65/33	2.8469 e+21	2.1474 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	9.8531 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	32
<i>add4</i>	29/12	4.9283 e+08	80	8.6	68
<i>addsub</i>	31/15	3.6032 e+09	32794	1.5	4.6
<i>add32</i>	65/32	1.5677 e+21	1.7179 e+10	3.8	5.8
<i>mul06</i>	12/12	26264	13	31	48
<i>mul07</i>	14/14	163361	10	320	512
<i>mul08</i>	16/16	984384	?	6872	?
<i>pitch</i>	16/48	27560	35	14.5	39
<i>rip08</i>	16/9	182893	631	0.3	0.4

Table 5. Functions from the ISCAS-85 and MIS benchmarks.

Name	#P	MP	IPS	ratio
<i>s344</i>	3270148	106994	54735	1.9
<i>xparc</i>	15039	74267	33083	2.2
<i>ex5</i>	2532	29423	11074	2.7
<i>jbp</i>	2496809	297257	138973	2.1
<i>test4</i>	6139	135804	49182	2.7
<i>OutOfStage</i>	5280741	681082	80107	8.5
<i>accpla</i>	1758057	> 2600000	255743	> 10.1

Table 6. Comparison between metaproducts and IPSs.

## 1.7 CONCLUSION

We have presented in this chapter a new prime and essential prime computation procedure of Boolean functions that overcomes the limits of all already known procedures. The performance of this procedure results from the combination of the *implicit prime set* (IPS) representation and of its associated calculus. Figure 12 shows the graph obtained by relating the CPU times needed for prime computation, and the number of computed primes. This graph shows that there is no relation at all between this two parameters. The straight line on the left corresponds to a linear dependence between the CPU time and the number of primes, and all techniques manipulating primes explicitly have performances that are above this line. On the contrary the graph shows that the procedure presented in this chapter has on all these examples a cost that is sub linear with respect to the number of computed primes.

Figure 12. CPU times vs numbers of primes.

The development of this new procedure was motivated by the following remark. Metaproducts 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 metaproducts. However taking advantage of IPSs gives rise to two major problems. Firstly there is no relation at all between set operations on IPSs and logical operations, as it was the case for metaproducts. Secondly the essential prime computation procedure presented in [7] manipulates sets of products that are not all primes.

This led us to the definition of a new graph calculus and the essential prime computation procedure presented here. IPS based prime computation is more efficient than the procedure based on metaproducts presented in [7].

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

## Acknowledgments

The authors would like to thank S. Minato, T. Sasao, and F. Somenzi, who reviewed this chapter.

## 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] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran", in Proc. of *Int'l Symposium on Circuits and Systems*, 1985.
- [4] R. E. Bryant, "Graph-based Algorithms for Boolean Functions Manipulation", *IEEE Trans. on Computers*, Vol C-35, 1986.
- [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, "Towards An Interactive Fault Tree Analyser", in Proc. of the *IASTED Conference on Reliability, Quality Control, and Risk Assessment*, Washington D.C., USA, November 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*, Anaheim CA, 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.