

Towards a Symbolic Logic Minimization Algorithm

Olivier Coudert Jean Christophe Madre

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

Abstract

This paper presents the application of the recently introduced implicit sets of products manipulations based on binary decision diagrams to the generation of irredundant prime covers of Boolean functions and of multiple output Boolean functions. The results obtained with this procedure are compared with the ones obtained with ESPRESSO and with the procedure presented in [11].

1 Introduction

Techniques have been recently introduced to manipulate in an implicit way sets of products built out of finite sets of Boolean variables [5]. These techniques have made possible to implement prime and essential prime implicant computation procedures that can handle Boolean functions with sets of primes much too large to be computed using any previously available technique [5, 6]. The key ideas that underlie these procedure are to represent and to compute these sets implicitly using *metaproducts* [5], or *implicit prime sets (IPS)* and to denote these metaproducts and IPSs with binary decision diagrams (BDDs) [4]. These techniques overcome all previously known computation techniques because their costs are related to the sizes of the BDDs they manipulate, and not to the number of computed prime implicants.

This paper presents the application of these techniques to the computation of irredundant prime covers of Boolean functions. This new procedure is very close to the procedure developed concurrently by S. Minato and presented in [11], except that this one makes use of IPSs to represent in an implicit way the sets of primes that are manipulated. The other major difference is that the procedure presented here generates the IPS of the cover, which can be further manipulated, while the procedure of [11] does not keep in memory any trace of the generated products because it would not be possible for very large covers. This new procedure

can deal with functions that cannot be handled by ESPRESSO [2], and is faster than the one proposed in [11] when dealing with complex functions. Moreover, it produces in some cases smaller irredundant prime covers than this procedure.

The paper is divided in 5 parts. Section 2 presents the problem addressed here, and introduces the notations and the elementary concepts that will be used to solve them. Section 3 defines the IPS representation. Section 4 presents the IPS based irredundant prime cover computation algorithm and briefly explains how it can be used to deal with partially defined vectorial Boolean functions. Section 5 gives experimental results obtained with this new procedure and compares them with the ones obtained with ESPRESSO and with the procedure presented in [11].

2 Definitions

We consider functions from the set $E = E_1 \times \dots \times E_n$, where each E_k is a finite set, into the set $\{0, 1, *\}$. Such a function f defines a partition of the set E into the 3 subsets $f^0 = f^{-1}(0)$, $f^1 = f^{-1}(1)$, and $f^* = f^{-1}(*)$, called the *off-set*, the *on-set*, and the *don't care-set* of the function f respectively. In the sequel 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.

A *product* built on the set E 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 . 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 .

A set of products P is a *prime cover* of the function f iff it is made of primes of the function f , and $f^1 \subseteq (\bigcup_{p \in P} p) \subseteq f^{1*}$. This prime cover is *irredundant* iff any proper subset P' of P is not a cover of f . It is minimal if there does not exist another prime cover P'

of f that is smaller, i.e., that has less products, than P .

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 is isomorphic to the set of strings $\{\epsilon, \overline{x_1}, x_1\} \times \dots \times \{\epsilon, \overline{x_n}, x_n\}$, where ϵ is the empty string. Such a string is interpreted as the conjunction of its literals, which is the characteristic function of the product it represents. For instance the string $x_1\overline{x_2}x_4$ of P_4 represents the product $\{1\} \times \{0\} \times \{0, 1\} \times \{1\}$, 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 the variable 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 the variable 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 $P = \{x_1\overline{x_4}, x_2, x_2x_4, \overline{x_2}x_3\}$, 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 :

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

Any subset S of $\{0, 1\}^n$ can be represented by a unique Boolean function from the set $\{0, 1\}^n$ into $\{0, 1\}$, called its *characteristic function*, that assigns any element x of $\{0, 1\}^n$ the value 1 if it is in S and the value 0 otherwise. By definition the set S is the on-set of its associated completely defined characteristic function. Thanks to this perfect correspondence we will often use, in the sequel, the same symbol to denote a subset of $\{0, 1\}^n$ and its characteristic function.

We assume the reader familiar with binary decision diagrams (BDD), that are a canonical graph representation of Boolean functions introduced in [4]. BDDs 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. This means that very large subsets of $\{0, 1\}^n$ can be represented with small BDDs, and that the elementary set operations on this representation have costs not related with the numbers of elements that are in the denoted sets.

3 Implicit Prime Set Manipulation

The set P_n has 3^n elements so $\lceil \log_2(3^n) \rceil$ Boolean variables are sufficient to represent any of its subsets. However, though this number is theoretically

sufficient, it is not the most interesting from the computing point of view. This section presents a graph representation of subsets of P_n that uses $2n$ Boolean variables, so more than necessary, but that makes the operations on these sets easy to express and to evaluate.

3.1 Metaproducts

Metaproducts are built using the many-to-one mapping σ 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 $o_k = 0$, $l_k = \overline{x_k}$ 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 $x_1\overline{x_2}x_4$. 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 *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 diagram of this function. Figure 1 shows the metaproduct \mathcal{P} of the subset of products $P = \{x_2\overline{x_4}, x_1x_3x_4, \overline{x_1}x_2\overline{x_3}x_4\}$ 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 $\mathcal{P}(o, s) = 1$.

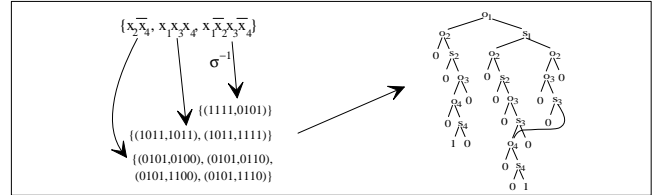


Figure 1. Metaproduct of the subset $\{x_2\overline{x_4}, x_1x_3x_4, \overline{x_1}x_2\overline{x_3}x_4\}$ of P_4 .

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 the same 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_{ϵ_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:

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

where π 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.

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 particular sets of products that we call prime sets. Then we explain how these redundancies can be eliminated from these particular metaproducts to produce the *implicit prime set* representation.

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 . This characteristic is formally expressed by the following predicate that holds on a set of products iff it is a prime set:

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

The set of primes of any function from E to $\{0, 1\}$ is a prime set. The set of prime sets is closed under the intersection, the difference, the cartesian product, and the canonical decomposition defined above. However it is not closed under the complementation, the union, or the concatenation.

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

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

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

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

The property (1) expresses 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 property (2) implies that all the occurrence variables occur on every path from the root of the metaproduct of a prime set to the leaf “1”. The property (3) implies 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 .



Figure 2. Reduction rules transforming metaproducts into IPSs.

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

A proof of this theorem can be found in [7]. Figure 3 shows the metaproduct with 16 vertices of the prime set $\{x_2 \overline{x_4}, x_1 x_3 x_4, \overline{x_1} x_2 \overline{x_3} x_4\}$ of P_4 and its IPS with only 8 vertices.

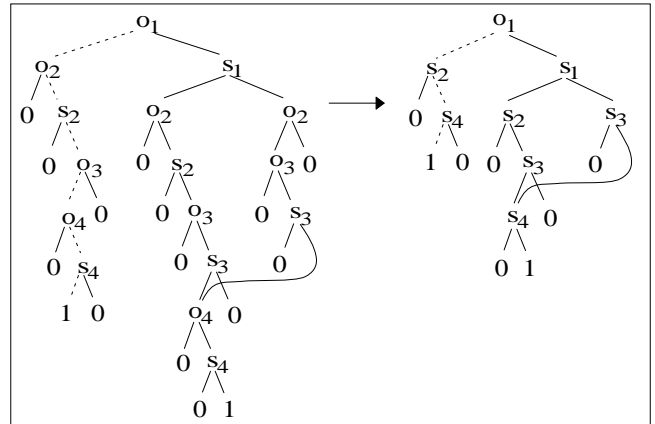


Figure 3. Metaproduct and IPS of the set $\{x_2 \overline{x_4}, x_1 x_3 x_4, \overline{x_1} x_2 \overline{x_3} x_4\}$.

This figure also illustrates the way IPSs must be interpreted. 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 s_2 occurs on the path but not the variable o_2 , which means that the rule (R2) has been effective, so the variable o_2 must be set to 1. The variable s_4 occurs on the path but not the variable o_4 , so for the same reason the variable o_4 must be set to 1. Finally the variables o_3 and s_3 do not occur on this path, which means that the rule (R1) has been effective so the variable o_3 must be set to 0. 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 $x_2 \overline{x_4}$, which is indeed an element of the set of products denoted by this IPS, and it corresponds to the path marked with dotted lines in the metaproduct.

The drawback that is paid for obtaining a more compact representation of prime sets is that the elementary set operations on prime sets do not correspond with logical operations on their IPSs. These set operations can nevertheless be realized, using a dedicated calculus, with costs polynomial with respect to the sizes of these IPSs [7].

4 Irredundant Prime Cover Computation

The irredundant prime cover algorithm presented here is based on the recursive computation scheme proposed by E. Morreale in [12]. This computation scheme uses a divide and conquer strategy based on Shannon expansion to build an irredundant prime cover made of 3 parts, the first one being made of primes in which the literal \bar{x}_k occurs, the second part made of primes in which the literal x_k occurs, and the last part made of primes in which the variable x_k does not occur.

S. Minato has recently presented in [11] the adaptation of this computation scheme to Boolean functions represented with binary decision diagrams. The canonicity of the BDD representation allows this algorithm to avoid redundant computations by making identical sub-problems recognition much easier to be realised. The algorithm presented here, which was developed concurrently with the one presented in [11], is very closed to it except that it makes use of IPSs to represent the sets of prime implicants it manipulates.

4.1 Dealing with Boolean Functions

The algorithm for generating irredundant prime cover of partially defined Boolean functions is shown in Figure 4. The function *IrrCover* takes as input the BDDs denoting the sets f^1 and f^{1*} , and returns the IPS of an irredundant prime cover of the function f defined by these two sets. This function makes use of two other functions. The function *GetFun* produces the BDD of the function f equal to the sum of the products in the set denoted by an IPS \mathcal{P} , using the equality $f = \lambda s.(\exists o \mathcal{P}(o, s))$ [7]. The function *Norme* is responsible for creating a canonical IPS from a root variable and two sub-graphs.

Proving that this algorithm recursively generates an irredundant prime cover of the function f can be made using induction on the number of variables of f . Let us note $I(k)$ the predicate stating that *IrrCover* generates an irredundant prime cover of f , where f uses

the variables $\{x_1, \dots, x_k\}$. if f does not depend on any variable, then either $f^1 = 0$ or $f^1 = 1$. But since $(f^1 \Rightarrow f^{1*})$ is a tautology, this means that either $f^1 = 0$ or $f^{1*} = 1$. These two cases are treated by the first tests made in the function *IrrCover*. In the first case the function returns the IPS 0 which denotes the empty set, and in the second case the function returns the IPS 1 which denotes the set $\{\epsilon\}$. Both are irredundant prime covers of the corresponding functions. Therefore $I(0)$ is true.

```

function IrrCover( $f^1$  : BDD,  $f^{1*}$  : BDD) : IPS;
if  $f^1 = 0$  return 0;
if  $f^{1*} = 1$  return 1;
let  $g_0 = f_{x_k}^1 \wedge \neg f_{x_k}^{1*}$  and
      $g_1 = f_{x_k}^1 \wedge \neg f_{x_k}^{1*}$  and
      $irr0 = \text{IrrCover}(g_0, f_{x_k}^{1*})$  and
      $irr1 = \text{IrrCover}(g_1, f_{x_k}^{1*})$  and
      $m^1 = (f_{x_k}^1 \wedge \neg \text{GetFun}(irr0)) \vee$ 
          $(f_{x_k}^1 \wedge \neg \text{GetFun}(irr1))$  and
      $m^{1*} = f_{x_k}^{1*} \wedge f_{x_k}^{1*}$  in
     return Norme( $o_k$ ,
                  IrrCover( $m^1, m^{1*}$ ),
                  Norme( $s_k, irr0, irr1$ ));

```

Figure 4. Algorithm *IrrCover*.

Now assume that $I(j)$ is true for $0 \leq j < k$, and assume that f uses the variables $\{x_1, \dots, x_k\}$. If one of the two terminal cases is satisfied, then the function *IrrCover* generates a correct irredundant prime cover. Now suppose that $f^1 \neq 0$ and $f^{1*} \neq 1$. Since $(f^1 \Rightarrow f^{1*})$ is a tautology, this implies that neither f^1 nor f^{1*} are constant functions, and one of these two functions uses the variable x_k . The functions g_0 , g_1 , $f_{x_k}^{1*}$, and $f_{x_k}^{1*}$ use at most $k - 1$ variables, so by hypothesis, $irr0$ and $irr1$ are irredundant prime covers of the partial Boolean functions described by the couples $(g_0, f_{x_k}^{1*})$ and $(g_1, f_{x_k}^{1*})$ respectively.

Every prime of the irredundant prime cover $irr0$ contains at least one minterm of $f_{x_k}^1$ that is not in $f_{x_k}^{1*}$. This means that the literal \bar{x}_k must be added to these primes in order to get primes of the function f . The resulting primes form the first part of the irredundant prime cover of f . In the same way, every prime of the irredundant prime cover $irr1$ contains at least one element of $f_{x_k}^1$ that is not in $f_{x_k}^{1*}$, which means that the literal x_k must be added to these primes in order to get primes of the function f . The resulting primes form the second part of the irredundant prime cover of f .

The remaining minterms of the function f that have not been covered yet are in the set m^1 . This set is

included in the set m^{1*} which is disjoint from the sets g_0 and g_1 . The primes of the irredundant prime cover of m^1 are all included in this set so they are primes of the function f , and none of them contains any occurrence of the variable x_k . These primes form the last part of the irredundant prime cover of f .

4.2 Dealing with Boolean Vectorial Functions

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 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 $\{0, 1, *\}$. Dealing with such functions can be done in two ways. We have presented in [10] an extension of the metaproduct representation to the domain $\{0, 1\}^n \times \{1, \dots, m\}$ and the associated prime computation procedure. More recently, we have shown in [7] that it is possible to reduce the problem of prime computation of this function to the one of a partial Boolean function defined in the following way.

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 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 $y = [y_1 \dots y_m]$:

$$\mathcal{F}^{1*}(f) = \lambda x. \lambda y. \left(\bigwedge_{k=1}^m (\neg y_k \vee f_k^{1*}(x)) \right)$$

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

The function $\mathcal{F}(f)$ is always well defined, because the formula $(\mathcal{F}^1(f) \Rightarrow \mathcal{F}^{1*}(f))$ is a tautology. We have shown in [7] that there is a one-to-one mapping between the primes of the function defined above and those of the function f , and that any prime computation problem on f , including irredundant prime cover generation, corresponds with the same problem on $\mathcal{F}(f)$.

5 Experimental Results

The method presented here has been successfully applied on all the MCNC benchmark examples [15]. Except for 22 out of the 145 vectorial Boolean functions

of this benchmark, the irredundant prime cover generated using this method is less than 15% larger than the minimal prime cover. The variable ordering used to build the BDDs of the functions is computed using the heuristics proposed by Touati in [14].

Name	[11]		IPS based		Espresso	
	#I	T	#I	T	#I	T
<i>add8</i>	2519	13.3	2519	1.4	2519	443.1
<i>achil8n</i>	6561	8.7	6561	1.2	6561	3513.7
<i>mult6</i>	2284	26.7	2284	20.0	1893	1126.2
<i>alupla</i>	2155	20.5	2144	17.5	2144	257.3
<i>duke2</i>	126	3.2	87	3.4	87	28.8
<i>vg2</i>	110	1.9	110	1.0	110	42.8
<i>c432</i>	84235	1744.8	81783	404	–	–

Table 1. Comparison with other methods.

Table 1 compares the experimental results obtained on MIS examples using the technique proposed here with the results presented in [11], and the results of ESPRESSO. The column #I gives the number of primes in the generated irredundant prime covers, and the column T gives the CPU times in seconds on a SUN Sparc2 workstation. A “–” indicates that this CPU time is larger than 10 hours.

Name	i/o	IPS	#I	T
<i>bcc</i>	26/45	2231	138	4.8
<i>bcd</i>	26/38	1436	118	3.4
<i>x2dn</i>	82/56	617	104	7.0
<i>mish</i>	94/43	361	82	2.5
<i>ibm</i>	48/17	514	173	3.5
<i>soar</i>	83/94	2406	392	10.1
<i>cont</i>	30/28	655	231	11.4
<i>s344</i>	24/26	1559	274	14.8
<i>s526</i>	24/27	475	130	13.9
<i>s1196</i>	32/32	5539	1059	163.4
<i>add3</i>	21/11	6732	13630	21.1
<i>add4</i>	29/12	5502	131883	18.9
<i>addsub</i>	31/15	978	169813	3.8
<i>cbp16</i>	33/17	269	655287	1.3
<i>cbp32</i>	65/33	541	42949672823	2.9
<i>mul07</i>	14/14	21535	8826	119.2
<i>mul08</i>	16/16	79827	33464	1628
<i>pitch</i>	16/48	2092	105	11.2

Table 2. Other examples.

Once the IPS of an irredundant prime cover has been built, it can be used for further computations. Note that the method of [11] generates explicitly each product of the cover, which can make difficult the manipulation of large covers. For example, the irredundant prime cover we generated for *c432* is made of 81783 products that have nearly 1 million of literals, but its IPS has only 8183 vertices.

Table 2 presents some other examples coming from the MCNC benchmark and the ISCAS benchmark [3]. For each file, column **i/o** gives the numbers of input and of output variables of the the circuit, column **T** gives the CPU times in seconds for computing an irredundant prime cover, column **#I** gives the number of products of this irredundant prime cover, and column **|IPS|** gives the number of vertices of its IPS.

6 Conclusion

We have presented in this paper a new irredundant prime cover computation procedure based on the *implicit prime set* (IPS) representation. The cost of this procedure is not related to the size of the generated irredundant prime cover but to the size of the BDDs and of the IPSs that are manipulated. This procedure can thus deal with vectorial Boolean functions that can be handled neither by ESPRESSO [2] nor by the procedure presented in [11].

The IPS based irredundant prime cover computation algorithm can be improved by using variable ordering heuristics that determine the degree of monotony of the Boolean function with respect to its variables. Work is being done to exploit this information in the best way. IPS based technique can also be used for dealing with the first step of the Quine-McCluskey 2-level minimization procedure, which consists in computing the cyclic core of the function under treatment. The work being done in this direction will be presented in a forthcoming paper.

Acknowledgments

The authors would like to thank S. Minato for providing us with some examples.

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, "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.
- [6] O. Coudert, J. C. Madre, "Implicit and Incremental Computation of Primes and Essential Primes of Boolean functions", in Proc. of *DAC'92*, June 1992.
- [7] O. Coudert, J. C. Madre, "A New Graph Based Prime and Essential Prime Computation Technique", in Proc. of the *International Symposia on Information Sciences (ISKIT'92)*, Iizuka, Fukuoka, Japan, July 1992.
- [8] S. J. Hong, S. Muroga, "Absolute Minimization of Completely Specified Switching Functions", *IEEE Trans. on Computers*, Vol 40, pp. 53–65, 1991.
- [9] S. Kleene, *Introduction of Metamathematics*, Van Nostrand, 1952.
- [10] B. Lin, O. Coudert, J. C. Madre, "Symbolic Prime Generation for Multiple-Valued Functions", in Proc. of *DAC'92*, June 1992.
- [11] S. Minato, "Fast Generation of Irredundant Sum-of-Products Forms from Binary Decision Diagrams", in Proc. of *SASIMI'92*, Kobe, Japan, April 1992.
- [12] E. Morreale, "Recursive Operators for Prime Implicant and Irredundant Normal Form Determination", *IEEE Trans. on Computers*, 1970.
- [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.