Implicit and Incremental Computation of Primes and Essential Primes of Boolean functions

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

Abstract

Recently introduced implicit set manipulation techniques have made it possible to formally verify finite state machines with state graphs too large to be built. This paper shows that these techniques can also be used with success to compute and manipulate implicitly extremely large sets of prime and of essential prime implicants of incompletely specified Boolean functions. These sets are denoted by meta-products that are represented with binary decision diagrams. This paper describes two procedures. One is based on the standard BDD operators, and the other, more efficient, takes advantage of the structural properties of BDDs and of meta-products to handle a larger class of functions than the former one.

1 Introduction

We have recently introduced a technique [4] for verifying finite state machines that can deal with machines with state graphs too large to be built. The key concepts that make this verification possible are to denote subsets of \( \{0, 1\}^n \) with their characteristic functions, and represent these Boolean functions with binary decision diagrams (BDDs) that are a very compact graph representation of such functions [3]. Since there is no relation between the size of a set and the size of the BDD that denotes it, the computation cost of this technique is completely independent from the number of states of the machines [4].

In this paper we show that these concepts can be used with success to implement implicit computation procedures of the sets of prime and of essential prime implicants of incompletely specified Boolean functions. These procedures have costs that are independent of the sizes of these sets, and thus they overcome the limitations of methods based on explicit prime manipulations [2, 8, 11, 12].

This paper is divided in 6 parts. Section 2 presents the elementary concepts that will be used to solve the problem treated here. Section 3 introduces the canonical meta-product representation of sets of products built out of finite sets of Boolean variables. Section 4 gives the expressions that define the sets of prime and of essential prime implicants of incompletely specified Boolean functions, and shows that these equations can be evaluated with the standard BDD operators. Section 5 describes how this standard procedure can be transformed into a more efficient incremental procedure. Section 6 gives experimental results obtained with these procedures and discusses them.

2 Definitions

A Boolean function \( f \) from \( \{0, 1\}^n \) into \( \{0, 1\} \) denotes a unique subset \( S_f \) of \( \{0, 1\}^n \), made of all the elements \( x \) of \( \{0, 1\}^n \) such that \( f(x) = 1 \). Conversely any subset \( S \) of \( \{0, 1\}^n \) can be represented by a unique Boolean function \( \chi_S \) from \( \{0, 1\}^n \) into \( \{0, 1\} \), called its characteristic function, such that \( \chi_S(x) = 1 \) if and only if \( x \in S \). Characteristic functions are a very interesting representation of Boolean sets because Boolean operators correspond with set operators [4].

We note \( P_n \) the set of products that can be built out of the set of variables \( \{x_1, \ldots, x_n\} \). A product is a formula of the form \((l_1 \ldots l_k)_p \), \( k \leq n \), whose literals \( l_j \) are built out of distinct variables. The product \( p \) contains the product \( p' (p \geq p') \) if and only if \( S_p \supseteq S_{p'} \). The relation \( \geq \) is a partial order on \( P_n \).

An incompletely specified Boolean function \( f_c \), also called a function with a care set, is defined by a couple \((C, f)\), where \( f \) is a function from \( \{0, 1\}^n \) into \( \{0, 1\} \), and the care set \( C \) is a subset of \( \{0, 1\}^n \). This function is defined by \( f_c(x) = f(x) \) if \( x \in C \), and \( f_c(x) = * \) if \( x \notin C \), where the symbol “*” can be either 0 or 1 [2].

The product \( p \) of \( P_n \) is a cube of the function \( f_c = (C, f) \) if and only if the set \( S_p \cap C \) is not empty, and for any element of this set, the function \( f \) evaluates to 1. The cube \( p \) of \( f_c \) is a prime implicant or prime of \( f_c \) if and only if \( p \) is a maximal element of the set of cubes of \( f_c \) with respect to the partial order “\( \geq \)”. Finally the
prime \( p \) of \( f_c \) is an essential prime implicant or essential prime of \( f_c \) if and only if the function denoted by the sum of all the other primes of \( f_c \) is not equal to \( f \) on the care set \( C \) [2].

Quantified propositional formulas are formulas of the form \((Q_1 x_1 \ldots Q_n x_n f)\), where \( Q_1, \ldots, Q_n \) are either the universal or the existential quantifier, \( f \) is a propositional formula, and \( x_1, \ldots, x_n \) are variables occurring in \( f \) [4]. Any quantified propositional formula can be rewritten into a propositional formula, using the identities \((\forall x \ f) = (\neg \exists x \ \neg f)\), and \((\exists x \ f) = (\neg \forall x \ \neg f)\), where \( x \) is free in \( f \), and \((f \neg \exists f)\) is Shannon’s expansion of \( f \) with respect to \( x \) [4]. Adding quantifiers to Propositional Logic does not increase its expressive power, but it allows us to write very concisely expressions whose quantifier-free forms have exponential sizes.

3 Meta-Products

The set of products \( P_n \) has \( 3^n \) elements, so \( \lceil \log_2(3^n) \rceil \)

Booleans are sufficient to represent any element and any subset of \( P_n \). However, using this optimal encoding, there is not direct link between an element of \( P_n \) and its representation. In order to establish such a direct link, we use \( n + n \) variables, the first \( n \) noted \( o_1, \ldots, o_n \), and the second \( n \) noted \( s_1, \ldots, s_n \), and called the sign variables. We define the mapping \( \sigma \) from the set \( \{0, 1\}^n \times \{0, 1\}^n \) into the set \( P_n \) in the following way:

\[
\sigma(o, s) = l_1 \ldots l_n,
\]

where \( l_k \) is the empty string if \( o_k = 0 \), \( l_k \) is \( \pi_k \) if \( o_k = 1 \), and \( s_k = 0 \), and finally \( l_k \) is \( x_k \) if \( o_k = 1 \), and \( s_k = 1 \). For instance, the couple \((0111), (1101)\) denotes the product \((x_2 \pi_2 x_4)\). The meta-product \( P \) of the subset \( P = \{p_1, \ldots, p_m\} \) of \( P_n \) is the characteristic function of the subset \((\{m \}\sigma^{-1}(p_k))\) of \( \{0, 1\}^n \times \{0, 1\}^n \). It is a canonical functional representation [7].

3.1 Properties of Meta-Products

A couple \((o, s)\) of \( \{0, 1\}^n \times \{0, 1\}^n \) denotes a unique product of \( P_n \), which itself denotes a unique subset of \( \{0, 1\}^n \), whose characteristic function, noted \( \chi(o, s) \), is defined by the equation

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

The predicate \( \equiv \) expresses that the products \( p \) and \( p' \) represented by the couples \((o, s)\), and \((o', s')\) respectively are identical, which is the case if and only if they are made of the same literals:

\[
c \equiv c' = (\bigwedge_{k=1}^{n} (o_k \Leftrightarrow o'_k) \land (o_k \Rightarrow (s_k \Leftrightarrow s'_k))).
\]

The predicate \( \geq \) expresses that the product \( p \) denoted by the couple \((o, s)\) contains the product \( p' \) denoted by the couple \((o', s')\), which is the case if and only if all the literals of \( p \) also occur in \( p' \):

\[
c \geq c' = (\bigwedge_{k=1}^{n} (o_k \Rightarrow (o'_k \land (s_k \Leftrightarrow s'_k)))).
\]

3.2 Representing Meta-Products with BDDs

The implicit prime and essential prime computation procedures that will be presented in the next sections make an intensive use of the predicates \( \leq, \geq \), and of the function \( \lambda(o, s) \). The variable orderings that minimize the BDDs of these predicates are all compatible with the relation

\[
o_{\pi(1)} < o'_{\pi(1)} < \{s_{\pi(1)}, s'_{\pi(1)}, x_{\pi(1)}\} <
\]

\[
\ldots
\]

\[
o_{\pi(n)} < o'_{\pi(n)} < \{s_{\pi(n)}, s'_{\pi(n)}, x_{\pi(n)}\},
\]

where \( \pi \) is a permutation of the integers \( \{1, \ldots, n\} \). These variable orderings are good candidates for building BDDs of meta-products. Among them, we have chosen the ordering that minimizes the computational cost of the routines manipulating meta-products [7]:

\[
o_1 < o'_1 < s_1 < s'_1 < x_1 < \cdots < o_n < o'_n < s_n < s'_n < x_n
\]

4 Implicit Prime and Essential Prime Computation

The theorems given in this section show that the usual Boolean operators provided by available BDD packages, for instance [1, 10], are sufficient to compute in an implicit way the sets of primes and of essential primes of incompletely specified Boolean functions.

4.1 Implicit Cube Computation

The meta-product \( Cube(f_c) \) of the set of cubes of the function \((C, f)\) evaluates to 1 for the couple \((o, s)\) if and only if the product \( p \) denoted by \( c \) is such that the set \( p \cap C \) is not empty, and is included in \( S_f \). This is expressed by the following theorem [7].

Theorem 1 The meta-product \( Cube(f_c) \) of the set of products that are cubes of the function \( f_c \) is

\[
\lambda o. \lambda s. ((\exists x \ \chi_C(x) \land \chi(o, s)(x)) \land
\]

\[
(\forall x \ (\chi_C(x) \land \chi(o, s)(x)) \Rightarrow f(x))).
\]

4.2 Implicit Prime Computation

By definition the prime implications of the function \((C, f)\) are cubes of this function that are maximal with respect to the partial order \( \equiv \) defined in Section 2 [7].

Theorem 2 The meta-product \( Prime(f_c) \) of the set of products that are primes of the function \( f_c \) is:

\[
\lambda o. \lambda s. (Cube(f_c)(o, s) \land
\]

\[
(\forall (o', s') \ (Cube(f_c)(o', s') \land (o', s') \geq (o, s)) \Rightarrow
\]

\[
(o', s') \equiv (o, s))).
\]
4.3 Implicit Essential Prime Computation

By definition, the meta-product $\mathcal{E}_{ss}(f_c)$ evaluates to 1 for the couple $(a, s)$ if and only if the subset of $\{0, 1\}^n$ that this couple denotes contains an element of the care set $C$ that is not contained in any of the subsets of $\{0, 1\}^n$ denoted by the other primes of $f_c$ [2, 7].

**Theorem 3** The meta-product $\mathcal{E}_{ss}(f_c)$ of the set of essential primes of the partial function $f_c$ is equal to

$$\lambda o. \lambda s. (\exists s' \ x_{C}(s') \land f(s') \land (\forall o' \ \text{Prime}(f_c)(o', s') \Rightarrow (o', s') \equiv (o, s))).$$

It is immediate, using the preceding theorems, to compute in a implicit way the BDDs of the meta-products $\text{Prime}(f_c)$ and $\mathcal{E}_{ss}(f_c)$ from the BDDs of $f$ and $x_C$. These computations, referred to as SP and SE in Section 6, that are shown by experience to be fairly efficient, have nevertheless a major drawback. It is necessary in both cases to compute the BDD of Cube($f_c$), that is shown by experience to be in most cases much larger than the resulting BDDs.

5 Incremental Prime and Essential Prime Computation

It has been shown [2] that the primes of the function $f = (x_k f_{\pi} + x_k f_{x_k})$ can be obtained by combining the primes of the functions $f_{\pi}$ and $f_{x_k}$ respectively. This process gives a way to compute the primes of $f$ without computing its cubes. Since each vertex of the BDD of $f$ represents one of the functions that can be obtained by applying, with a given order, Shannon expansion on $f$, the number of vertices in this BDD is the number of sets of primes and of combinations that are necessary to get the primes of $f$.

We show here that the procedure given in [11], which implements this computation scheme with explicit prime manipulations, can be generalized to obtain a combination technique that manipulates sets of primes implicitly. This new procedure relies on implicit prime set manipulations that are presented in [5]. We will consider here that the function to be treated is completely defined, the general case is handled in [5].

5.1 Incremental Prime Computation

The following theorem explains how the meta-products $\text{Prime}(f_{\pi})$ and $\text{Prime}(f_{x_k})$ can be combined to get $\text{Prime}(x_k f_{\pi} + x_k f_{x_k})$.

**Theorem 4** The meta-product $\text{Prime}(f)$ of the set of primes of the function $f = (x_k f_{\pi} + x_k f_{x_k})$ is

$$((\overline{x_k} \land \text{Prime}(f_{\pi} \land f_{x_k})) \lor (o_k \land \overline{x_k} \land \text{Prime}(f_{\pi}) \land \neg \text{Prime}(f_{x_k} \land f_{x_k})) \lor (o_k \land s_k \land \text{Prime}(f_{x_k}) \land \neg \text{Prime}(f_{\pi} \land f_{x_k})))$$

Computing the meta-product $\text{Prime}(f_{\pi} \land f_{x_k})$ can be done in two ways. The first and simplest way, used in the method referred to as IP1 in Section 6, consists in building the BDD of the function $(f_{\pi} \land f_{x_k})$ and in applying the incremental procedure on this BDD. Since this new BDD can contain vertices that are in none of the BDDs of the functions $f_{\pi}$ and $f_{x_k}$, this method can induce a non polynomial number of recursions with respect to the size of the BDD of the function $f$. The second method [5], used in the procedure referred to as IP2 in Section 6, consists in building the meta-product $\text{Prod}$ of all the possible conjunctions of one prime of $f_{\pi}$ and one prime of $f_{x_k}$, and then in extracting the required meta-product from $\text{Prod}$. Using this method, each vertex of the BDD of $f$ is treated once.

5.2 Incremental Essential Prime Computation

The problem addressed here is to combine the meta-products $\text{Prime}(f_{\pi})$, $\mathcal{E}_{ss}(f_{\pi})$, $\text{Prime}(f_{x_k})$, and $\mathcal{E}_{ss}(f_{x_k})$ in order to get $\mathcal{E}_{ss}(x_k f_{\pi} + x_k f_{x_k})$.

This combination process relies on 3 set operators that are evaluated implicitly. The operators $\mathcal{F}C$ and $\mathcal{FNC}$ are closely related. By definition, for any meta-product $P$ and any function $f$, $\mathcal{F}C(P, f)$ is the meta-product of the subset of products denoted by $P$ that are cubes of the function $f$, and $\mathcal{FNC}(P, f)$ is the meta-product of the subset of products denoted by $P$ that are not cubes of the function $f$. Both operators have computational costs in $O(|P| \times |f|)$, where $|P|$ and $|f|$ are the sizes of the BDDs of $P$ and $f$ respectively. The operator $\mathcal{FAE}$ computes, in a non-polynomial cost in the worst case, for any meta-products $P$ and $P'$, the meta-product of the subset of products of $P$ that contain an element that is not covered by any of the products denoted by $P'$. The implicit essential combination method is based on the following theorems.

**Theorem 5** The meta-product of the set of essential primes of the function $(x_k f_{\pi} + x_k f_{x_k})$, in which the literal $(x_k)$ occurs, is

$$o_k \land \overline{x_k} \land (\mathcal{F}C(\mathcal{E}_{ss}(f_{\pi})), \neg f_{x_k}) \lor \mathcal{E}_{ss}(f_{\pi}))$$

In this equation, $\mathcal{E}_{ss}(f_{\pi})$ is defined by

$$\mathcal{E}_{ss}(f_{\pi}) = \mathcal{FNC}(\mathcal{E}_{ss}(f_{\pi}), \neg f_{x_k}) \land \mathcal{FNC}(\mathcal{E}_{ss}(f_{\pi}), f_{x_k})$$

$$\mathcal{E}_{ss}(f_{x_k}) = \mathcal{FAE}(\mathcal{E}_{ss}(f_{\pi}), \text{Prime}(f_{x_k}) \lor \text{Prime}(f_{x_k} \land f_{x_k})).$$

**Theorem 6** The meta-product of the set of essential primes of the function $(x_k f_{\pi} + x_k f_{x_k})$, in which neither the literal $(x_k)$ nor the literal $(x_{k})$ occur, is

$$\overline{x_k} \land (\mathcal{F}C(\mathcal{E}_{ss}(f_{\pi}), f_{x_k}) \lor \mathcal{F}C(\mathcal{E}_{ss}(f_{x_k}), f_{\pi})).$$

We will compare two implicit essential prime computation procedures in Section 6. Procedure IE1, based on Theorem 3, computes meta-products of essential primes from meta-products of primes computed using Procedure IP1. Procedure IE1 is based on the preceding theorems.
6 Experimental Results

Table 1 gives the characteristics of each treated circuit: \#in and \#fun are the numbers of inputs and of outputs respectively; \#Prime and \#Ess are the cumulated sums of the numbers of primes and of essential primes respectively obtained for each output; CS is “yes” if output functions have a care set. The entries indust, math, and random correspond to the directories of the MCNC 2-level minimization benchmark. For these entries, column \#file is the number of circuits in this directory.

<table>
<thead>
<tr>
<th>Name</th>
<th>#in</th>
<th>#fun</th>
<th>CS</th>
<th>#Prime</th>
<th>#Ess</th>
</tr>
</thead>
<tbody>
<tr>
<td>math</td>
<td>23</td>
<td>128</td>
<td>yes</td>
<td>8616</td>
<td>1937</td>
</tr>
<tr>
<td>indust</td>
<td>111</td>
<td>2412</td>
<td>yes</td>
<td>87797</td>
<td>17284</td>
</tr>
<tr>
<td>random</td>
<td>11</td>
<td>189</td>
<td>yes</td>
<td>118141</td>
<td>169</td>
</tr>
<tr>
<td>dsp</td>
<td>14</td>
<td>421</td>
<td>no</td>
<td>22850</td>
<td>2223</td>
</tr>
<tr>
<td>mul07</td>
<td>14</td>
<td>14</td>
<td>no</td>
<td>28972</td>
<td>1551</td>
</tr>
<tr>
<td>mul08</td>
<td>16</td>
<td>16</td>
<td>no</td>
<td>152051</td>
<td>3879</td>
</tr>
<tr>
<td>addsub</td>
<td>31</td>
<td>15</td>
<td>no</td>
<td>408190</td>
<td>102413</td>
</tr>
<tr>
<td>s1423</td>
<td>91</td>
<td>79</td>
<td>no</td>
<td>460307</td>
<td>36226</td>
</tr>
<tr>
<td>add4</td>
<td>29</td>
<td>12</td>
<td>no</td>
<td>688167</td>
<td>247</td>
</tr>
<tr>
<td>cbp32</td>
<td>65</td>
<td>33</td>
<td>no</td>
<td>4.3e10</td>
<td>4.3e10</td>
</tr>
</tbody>
</table>

Table 1. Characteristics of Examples.

Each file has first been translated from its original format into an intermediate form in which Boolean functions are represented with a textual form of BDDs. The CPU times needed to perform these translations for all files of indust, math, and random directories is 160 seconds on a Sun Sparc2 machine. The variable ordering used to build these BDDs is the same for all functions of a circuit, and the occurrence and sign variables are ordered according to this input variable ordering.

<table>
<thead>
<tr>
<th>Name</th>
<th>SP</th>
<th>IP1</th>
<th>IP2</th>
<th>SE</th>
<th>IE1</th>
<th>IE2</th>
</tr>
</thead>
<tbody>
<tr>
<td>math</td>
<td>7</td>
<td>16</td>
<td>32</td>
<td>22</td>
<td>115</td>
<td></td>
</tr>
<tr>
<td>indust</td>
<td>258</td>
<td>83</td>
<td>364</td>
<td>577</td>
<td>424</td>
<td>-</td>
</tr>
<tr>
<td>random</td>
<td>495</td>
<td>211</td>
<td>873</td>
<td>636</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>dsp</td>
<td>24</td>
<td>17</td>
<td>34</td>
<td>63</td>
<td>49</td>
<td>343</td>
</tr>
<tr>
<td>mul07</td>
<td>137</td>
<td>57</td>
<td>345</td>
<td>272</td>
<td>199</td>
<td>-</td>
</tr>
<tr>
<td>mul08</td>
<td>1002</td>
<td>377</td>
<td>-</td>
<td>-</td>
<td>1429</td>
<td>-</td>
</tr>
<tr>
<td>addsub</td>
<td>3</td>
<td>1</td>
<td>29</td>
<td>8</td>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>s1423</td>
<td>153</td>
<td>36</td>
<td>102</td>
<td>347</td>
<td>231</td>
<td>-</td>
</tr>
<tr>
<td>add4</td>
<td>18</td>
<td>7</td>
<td>35</td>
<td>43</td>
<td>33</td>
<td>329</td>
</tr>
<tr>
<td>cbp32</td>
<td>13</td>
<td>2</td>
<td>5</td>
<td>31</td>
<td>16</td>
<td>175</td>
</tr>
</tbody>
</table>

Table 2. CPU times in seconds on Sun Sparc2.

The experimental results given in Table 2 show that these procedures can handle Boolean functions with very large numbers of primes and of essential primes. It appears clearly that the incremental prime computation method IP1 is much more efficient than the standard method SP. Surprisingly, though the computational cost of Procedure IP2 is theoretically lower than the cost of IP1, the former is much less efficient than the latter on these examples. In the same way, Procedure IE1 is much more efficient than Procedure IE2. In this table, symbol “-” means that CPU time is larger than 1500 seconds.

7 Conclusion

In this paper we have presented procedures for computing and manipulating in an implicit way the sets of primes, and of essential primes of incompletely specified Boolean functions. These methods allow us to deal with functions for which these sets are too large to be handled by enumeration based procedures [2, 8, 11, 12].

It has been shown [12] that computing the primes and essential primes of a multiple-valued input and output function comes down to computing the primes and essential primes of a multiple-valued inputs Boolean function. There are several ways to extend the methods presented here in order to deal with such Boolean functions, that correspond to different ways to encode the multiple-valued input variables of the function into Boolean variables [9]. A procedure dedicated to multiple output Boolean functions has also been developed [6].

References