

METAPRIME, an Interactive Fault Tree Analyser

Olivier Coudert, Member IEEE

Bull Corporate Research Center, Les Clayes-sous-bois

Jean Christophe Madre, Member IEEE

Bull Corporate Research Center, Les Clayes-sous-bois

Key Words

(Non) coherent fault tree analysis, Binary decision diagrams, Metaproducts.

Reader Aids

Purpose: Overcome state of the art

Special math needed for explanations: Propositional logic

Special math needed to using the results: Same

Results useful to: Reliability engineers and analysts

Summary & Conclusions

The performances of almost all available fault tree analysis tools are limited by the performance of the prime implicant computation procedure they use. All these procedures manipulate the prime implicants of the fault trees in extension, so that the analysis costs are directly related to the number of prime implicants to be generated, which in practice makes these tools difficult to be applied on fault trees with more than 20000 prime implicants.

This paper introduces an analysis method of coherent as well as noncoherent fault trees that overcomes this limitation because its computational cost is related to neither the number of basic events, nor the number of gates, nor the number of prime implicants of these trees. We present the concepts underlying the prototype tool METAPRIME, and the experimental results obtained with this tool on real life fault trees. These results show that these concepts allow us to completely analyse in seconds fault trees that no previously available technique could ever partially analyse, for instance noncoherent fault trees with more than 10^{20} prime implicants.

The concepts that we present here can also be used for the analysis of event trees because such trees denote Boolean functions on which these concepts can be applied. Prime implicant computation is a problem that is also critical in many other domains, in particular in artificial intelligence applications such as reasoning maintenance and multiple fault diagnosis. The application of the concepts underlying METAPRIME to the resolution of these problems is under study.

1 Introduction

Since their introduction at the beginning of the 1960's [10], fault trees have been widely used for reliability modeling because they allow system designers to describe very concisely how systems fail. Over the last decades, fault tree analysis has helped to dramatically improve the reliability of very complex systems, such as chemical and nuclear plants, aircraft and spacecraft, for which failures can result in very important and unacceptable damages [20].

Fault tree analysis is a two-fold problem. The *qualitative* aspect of the problem consists of analysing the set of *prime implicants* of all or part of the fault tree under analysis, that are called *minimal cut sets* in the particular case where the tree is coherent [13]. The *quantitative* aspect of the problem consists of evaluating, using the probabilities associated with the basic events of the fault tree, the probability associated with part

or all the gates of this tree. Computing the prime implicants of a fault tree with n basic events consists of computing the prime implicants of the Boolean function f denoted by this tree. Since, in the worst case, this function can have $O(3^n/n)$ prime implicants, qualitative fault tree analysis is a problem that is by nature *exponential* with respect to n [19]. In the same way, since the probability associated with a fault tree depends on the set of elements of $\{0, 1\}^n$ on which f evaluates to 1, computing this probability is also by nature *exponential* with respect to n .

To both aspects of fault tree analysis a lot of efforts has been devoted over the past decades. Though methods have been developed to evaluate the exact probability of a fault tree directly from its structure [17], almost all available analysis tools make an approximation using Poincaré's formula, which consists of summing the probabilities associated with either all or the most relevant prime implicants of the fault tree [10]. This approximation does not change the exponential nature of the problem, and is in most cases not restrictive enough to allow these analysis tools to handle in a satisfying way very complex fault trees.

The prime implicant computation problem has also been devoted a lot of efforts since its definition by Quine in the 1950's [18], because it has applications in many other engineering fields such as digital circuit design [4], automated reasoning [24], and artificial intelligence [14, 21]. Many prime implicant computation procedures have been developed [2, 4, 16, 24, 25] that, though being all different, have comparable computational costs because they all manipulate sets of prime implicants represented in *extension*. In practice, these procedures are difficult to apply on Boolean functions with more than 20000 prime implicants. Because of this limitation, all currently available fault tree analysis methods use approximations in the prime implicant computation. These approximations consist of computing only the most relevant prime implicants of the fault tree, that are either the ones with the lowest order, or the ones with the largest associated probability.

This paper presents a new fault tree analysis method that is dramatically different from all previously known methods because it manipulates sets of elements of $\{0, 1\}^n$ and sets of prime implicants in *intension*. These sets are denoted by Boolean functions, and all the operations that are necessary to perform an exhaustive fault tree analysis are done in intension through these functions instead of being done in extension on the sets themselves. These functions are represented with binary decision diagrams (BDD's) that are a very compact graph representation of Boolean functions [5]. The computational cost of the method presented here is related to the sizes of the BDD's it manipulates, and since there is *no relation* between the size of a set and the size of the BDD that represents this set [7], this computational cost is *completely independent* of the number of interpretations of the fault tree and of its number of prime implicants.

As a consequence, the method presented here dramatically overcomes the limitations of all previously known methods, because it can treat in seconds fault trees that cannot be analysed by any other technique, for instance non coherent trees with more than 10^{20} prime implicants [9]. It handles in an uniform way coherent as well as noncoherent fault trees. It provides system designers with *exact* qualitative and quantitative informations. This means that *all* prime implicants of the trees are computed, which is impossible using previously available techniques on complex fault trees, and that the probability associated with these trees is computed *without any* approximation, while almost all analysis tools only yield an upper bound. Also METAPRIME provides system designers with a very efficient browser that allows them to refine the qualitative analysis as much as desired. It is between 100 and 1000 times faster than any previously available technique on large fault trees.

The paper is divided in 5 parts. Section 2 gives the key characteristics of the binary decision diagram representation. Section 3 explains how sets of products built out of a finite set of Boolean variables can be represented in a canonical way with Boolean functions that we call *metaproducts*, and gives the expressions that must be evaluated to produce the metaproduct of the set of prime implicants of any Boolean function. Section 4 explains how these concepts are used in the prototype fault tree analysis system METAPRIME. Section 5 gives experimental results obtained with the procedure presented here and discusses them.

2 Notation and Nomenclature

2.1 Notation

f	Boolean function (or fault tree) and its BDD
n	number of variables of the Boolean function f
x_k, o_k, s_k	propositional variables
v_k	element of $\{0, 1\}$
$\overline{x_k}$	negation of the propositional variable x_k
$(f_{\overline{x_k}}, f_{x_k})$	Shannon decomposition of f with respect to x_k , i.e., unique couple of functions such that $f = (\overline{x_k} \wedge f_{\overline{x_k}}) \vee (x_k \wedge f_{x_k})$
$\neg, \wedge, \vee, \oplus$	logic NOT, AND, OR, XOR
$\Delta(., ., .)$	vertex of a BDD
$ \cdot $	size, i.e., number of vertices, of a BDD
l_k	literal
p, q	products
P	set of products
P_n	set of all the products that can be built from the n variables x_1, \dots, x_n
\mathcal{P}	metaproduct

2.2 Nomenclature

basic event	elementary event which reliability datum is given
gate	complex event described by a logical combination of other events
literal	propositional variable or its negation
product	non null conjunction of literals
BDD	Binary Decision Diagram
characteristic function	discriminant Boolean function associated with a set
interpretation	point that values a Boolean function to 1

3 Binary Decision Diagrams

Let f be a Boolean function from $\{0,1\}^n$ into $\{0,1\}$. The function f is the *characteristic function* of the unique subset S_f of $\{0,1\}^n$ defined by: $S_f = f^{-1}(1)$, i.e., $S_f = \{x \in \{0,1\}^n \mid f(x) = 1\}$. There is an obvious correspondence between set operators on subsets of $\{0,1\}^n$ and Boolean operators on their characteristic functions, for instance union corresponds with disjunction, intersection with conjunction, and complementation with negation.

When iterated for all the variables of the Boolean function f , Shannon decomposition produces a binary tree that is unique modulo the variable ordering used during the decomposition. This tree, called the *Shannon tree* of f [1], has $2^n - 1$ vertices and its leaves are the constants 0 and 1. The binary decision diagram (BDD) of f is a compacted representation of its Shannon tree [5]. This BDD is produced by deleting the vertices of this tree that are redundant, i.e., vertices that have isomorphic left and right branches, and by identifying all its remaining isomorphic subtrees [5]. Figure 1 shows the Shannon tree and the BDD of the function $f(x_1, \dots, x_4) = (x_1 \wedge (x_3 \oplus x_4)) \vee (x_2 \wedge (x_3 \oplus \bar{x}_4))$. The left branch (respectively the right branch) of a vertex labeled by a variable x_k denotes the function obtained by setting x_k to 0 (respectively by setting x_k to 1). In the sequel we will note by $\Delta(x, L, H)$ the unique vertex of a BDD that has the variable x at its root, and the left and right hand side BDD's L and H respectively. Note that $\Delta(x, L, H) \Leftrightarrow (\bar{x} \wedge L) \vee (x \wedge H)$.

The usual Boolean operators can be evaluated with a quadratic complexity, and the negation in linear time, on BDD's built with the same variable ordering [3, 5]. This polynomial complexity is a remarkable property that makes BDD's very different from previously used representations of Boolean functions, for instance the negation has an exponential complexity on the disjunctive normal form [7]. BDD's are very interesting representations of Boolean sets because there is *no relation at all* between the number of elements in a set and the size of the BDD that represents its characteristic function. Huge Boolean sets can thus potentially be represented with small BDD's [6], and the set operations can be performed with costs that are not related to the numbers of elements of the sets but to the sizes of the BDD's that represent them.

Because binary decision diagrams are based on Shannon decomposition, all Shannon decomposition based computation methods that have been developed to reason about Boolean functions, in particular the divide and conquer computation methods traversing, without building them, binary decision trees [22, 23], can be implemented using BDD's. The critical advantage of making such an implementation is that BDD's, because of their canonicity and the sharing of subgraphs, allow identical subproblems to be treated only once.

4 Prime Implicant Computation

This section presents the *metaproduct* representation, an original BDD based canonical functional representation of sets of products, that allows us to manipulate these sets with costs related to the sizes of the BDD's that denote them.

4.1 Definitions

The set P_n of products that can be built from the set of variables $\{x_1, \dots, x_n\}$ is the set of formulas $\{l_1 \wedge \dots \wedge l_n \mid l_k \in \{x_k, \overline{x_k}, 1\} \text{ for } 1 \leq k \leq n\}$. A product $l_1 \wedge \dots \wedge l_n$ will be written $l_1 \dots l_n$ in the sequel, and the 1's will be omitted. The *order* of a product is the number of literals occurring in it. The product p of P_n is an *implicant* of f if and only if $S_p \subseteq S_f$, and it is a *prime implicant* of f if and only if it is an implicant of f and there is no other implicant q of f such that $S_p \subset S_q$ [19]. For instance, the product x_1 is a prime implicant of the function $f(x_1, x_2) = x_1$, and $x_1 x_2$ is an implicant of f that is not prime.

4.2 MetaProducts

There are 3^n elements in the set P_n , so a Boolean space of dimension $\lceil n \log_2(3) \rceil$ is sufficient to encode all its elements unambiguously. However, though this dimension is theoretically sufficient, it is not the most interesting from the computational point of view. We present here an encoding using $2n$ variables, thus more than necessary, but whose very strong relations with Shannon decomposition make it easy to implement and efficient the qualitative fault tree analysis procedures.

We first define a many-to-one mapping σ from the set $\{0, 1\}^n \times \{0, 1\}^n$ onto the set P_n in the following way [7]: $\sigma([o_1 \dots o_n], [s_1 \dots s_n]) = l_1 \dots l_n$, where $l_k = 1$ 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 pair $([0111], [1101])$ denotes the product $x_2 \overline{x_3} x_4$. In the sequel we will note by o and s the vectors $[o_1 \dots o_n]$ and $[s_1 \dots s_n]$ respectively.

The *metaproduct* \mathcal{P} of the set of products P is the characteristic function of the set $(\bigcup_{p \in P} \sigma^{-1}(p))$. For instance the metaproduct of the subset $\{x_2 \overline{x_4}, x_1 x_3 x_4, \overline{x_1} x_2 \overline{x_3} x_4\}$ of P_4 is the characteristic function of the set of pairs $\{([0101], [0101]), ([0101], [0100]), ([0101], [0110]), ([0101], [1100]), ([1011], [1011]), ([1011], [1111]), ([1111], [0101])\}$. Since the set $\{\sigma^{-1}(p) \mid p \in P_n\}$ is a partition of $\{0, 1\}^n \times \{0, 1\}^n$, metaproducts are a canonical representation of subsets of P_n , and the set operations on sets of products correspond to the logical operations on metaproducts [7].

Consider a set of products P and its metaproduct \mathcal{P} . This set is the disjoint union of three subsets, one made of the elements of P in which the variable x_k does not occur, whose metaproduct is $\overline{o_k} \wedge \mathcal{P}_{\overline{o_k}}$, one made of the elements of P in which the literal $\overline{x_k}$ occurs, whose metaproduct is $o_k \wedge \overline{s_k} \wedge \mathcal{P}_{o_k \overline{s_k}}$, and one made of the elements of P in which the literal x_k occurs, whose metaproduct is $o_k \wedge s_k \wedge \mathcal{P}_{o_k s_k}$ [8].

The properties given above make many operations on sets of products linear with respect to the size of the BDD of the metaproducts of these sets, if these BDD's are built with the variable ordering [7]: $o_{\pi(1)} < s_{\pi(1)} < o_{\pi(2)} < s_{\pi(2)} < \dots < o_{\pi(n)} < s_{\pi(n)}$, where π is a permutation of the integers $\{1, \dots, n\}$. In the following we will only consider the case where this permutation equals the one defining the variable ordering used to build the BDD of the fault tree under treatment. Figure 2 shows the metaproduct of the set of products $\{x_2 \overline{x_4}, x_1 x_3 x_4, \overline{x_1} x_2 \overline{x_3} x_4\}$.

4.3 Prime Implicant Computation with Metaproducts

The computation of the metaproduct $\mathcal{P}rime(f)$ of the set of prime implicants of the Boolean function f is based on the following theorem, whose proof can be found in [8], that establishes the relation existing between the metaproduct $\mathcal{P}rime(f)$ and the metaproducts $\mathcal{P}rime(f_{\overline{x_k}})$, $\mathcal{P}rime(f_{x_k})$, and $\mathcal{P}rime(f_{\overline{x_k}} \wedge f_{x_k})$.

Theorem 1 Consider a Boolean function f from $\{0, 1\}^n$ into $\{0, 1\}$ and one of its variables x_k . The metaproduct $\mathcal{P}rime(f)$ of the set of prime implicants of f is equal to

$$\mathcal{P}rime(f) = (\overline{o_k} \wedge \mathcal{P}rime(f_{\overline{x_k}} \wedge f_{x_k})) \vee$$

$$(o_k \wedge \overline{s_k} \wedge \mathcal{P}rime(f_{\overline{x_k}}) \wedge \neg \mathcal{P}rime(f_{\overline{x_k}} \wedge f_{x_k})) \vee \\ (o_k \wedge s_k \wedge \mathcal{P}rime(f_{x_k}) \wedge \neg \mathcal{P}rime(f_{\overline{x_k}} \wedge f_{x_k})).$$

This theorem shows that the cost of computing the BDD of the metaproduct $\mathcal{P}rime(f)$ from the BDD's of the metaproducts $\mathcal{P}rime(f_{\overline{x_k}})$, $\mathcal{P}rime(f_{x_k})$, and $\mathcal{P}rime(f_{\overline{x_k}} \wedge f_{x_k})$ is polynomial with respect to the sizes of these BDD's. This theorem also gives the structure of the recursive algorithm that computes this BDD while traversing the BDD of f in a depth first way. In the worst case the number of vertices that this algorithm has to create and traverse is exponential with respect to the size of the BDD of f because for each vertex $\Delta(x, L, H)$ of this BDD the algorithm has to treat the BDD of $(L \wedge H)$ that can contain vertices that are neither in L nor in H . However in the case where the function f is unate, i.e., the fault tree is coherent, the number of recursions that are necessary to compute $\mathcal{P}rime(f)$ is exactly equal to the size of the BDD of f , because for each vertex $\Delta(x, L, H)$ of this BDD, the BDD $(L \wedge H)$ is either equal to L or H [8].

5 Application to Fault Tree Analysis

Figure 3 describes the architecture of the interactive fault tree analysis tool METAPRIME. This tool gives modeling engineers an interactive access, through a command shell, to a comprehensive set of fault tree analysis procedures. After loading a fault tree into memory, the user can select a gate of this tree, compute the BDD of the function associated with this gate, and the metaproduct of its set of prime implicants, and then enter an analysis loop in which the basic step consists of one of the following operations:

- Select a subset P of this set of prime implicants using pattern matching.
- Compute the number of elements of P .
- Compute the distribution of the elements of P w.r.t. their order, their probability, or the occurrence of a given literal.
- Evaluate the contribution of the elements of P to the probability associated with the current gate.
- Pretty print the elements of P .

5.1 Computing the BDD of a Fault Tree

Building the BDD representing a fault tree can be done through a two step process. The first step consists of identifying the modules of the tree and of determining the ordering of the propositional variables associated with its basic events [13]. This ordering is produced using a heuristics that has been proposed for multi-level logic networks [15]. The second step consists of traversing the tree in a depth first way and in computing its associated BDD in a bottom up way. At each gate the BDD's associated with the inputs of the gate are combined using the logical operator of this gate until the selected gate is reached. The decomposition of the tree into modules has a dramatic effect on the size of the resulting BDD since it has been shown that this size is linear with respect to the sum of the sizes of the BDD's of the different modules composing this tree [12].

5.2 Qualitative Analysis

The properties of metaproducts make it very easy to compute the number of products that are in the set P denoted by a metaproduct \mathcal{P} . The function $Size$ that, when applied on \mathcal{P} , returns the number of elements of P is defined by the following equations:

$$\begin{aligned} Size(0) &= 0 \\ Size(1) &= 1 \\ Size(\mathcal{P}) &= Size(\mathcal{P}_{\overline{o_k}}) + Size(\mathcal{P}_{o_k \overline{s_k}}) + Size(\mathcal{P}_{o_k s_k}), \quad \text{if } \mathcal{P} \text{ is different from } 0 \text{ and } 1. \end{aligned}$$

All elements of P_n have an order less or equal to n so the order distribution of the elements of P can be represented by an array of size $n+1$ whose j -th entry is the number of products of P of order j . The function *OrderDist* that, when applied on the metaproduct \mathcal{P} , returns the array representing the order distribution of the elements of P , is defined by the following equations

$$\begin{aligned}
OrderDist(0) &= [00 \dots 0] \\
OrderDist(1) &= [10 \dots 0] \\
OrderDist(\mathcal{P})[0] &= 0 \\
OrderDist(\mathcal{P})[j] &= OrderDist(\mathcal{P}_{\overline{o_k}})[j] + \\
&\quad OrderDist(\mathcal{P}_{o_k \overline{s_k}})[j-1] + \\
&\quad OrderDist(\mathcal{P}_{o_k s_k})[j-1], \text{ if } j \geq 1 \text{ and } \mathcal{P} \text{ is different from } 0 \text{ and } 1
\end{aligned}$$

It is obvious to program the recursive functions *Size* and *OrderDist* using the equations given in the preceding paragraphs. These functions both use the same recursion scheme that consists of traversing the BDD of \mathcal{P} in a depth first way, and of computing the result in a bottom up way by combining, at each vertex, the results obtained for the left and right sons of this vertex. In order to treat each vertex once during the traversal, the procedures store in each vertex of the BDD the result obtained for this vertex. Note that the recursion scheme defined above can also be used to compute in time linear with respect to the size of the BDD of \mathcal{P} the distribution of the probabilities associated with the elements of P and the distribution of the basic events in these elements.

More generally, the definition of metaproducts and the properties of BDD's allow a large class of operations on sets of products to be performed in times polynomial with respect to the sizes of the BDD's that denote these sets [9]. Selecting the elements of a set P whose order is in a given interval can be done in $O(|\mathcal{P}|)$. Selecting the elements of P that contain a given set of literals and do not contain another set of literals can also be done in $O(|\mathcal{P}|)$. Once the metaproduct of a subset P of prime implicants has been selected, the exact contribution of P to the probability of the currently selected gate of the fault tree can be computed by first generating the BDD of the function denoted by the sum of products in P , and then applying the function *Pr* described below on this BDD.

5.3 Quantitative Analysis

The probability $Pr(f)$ associated with a gate of a fault tree is defined in terms of the probabilities $Pr(x_1), \dots, Pr(x_n)$ of its basic events, and the interpretations of the Boolean function f denoted by the part of the fault tree that has this gate as root, in the following way:

$$Pr(f) = \sum_{\substack{[v_1 \dots v_n] \in \{0,1\}^n \\ f(v_1, \dots, v_n) = 1}} \left(\prod_{k=1}^n Pr(v_k) \right),$$

where $Pr(v_k) = Pr(x_k)$ if $v_k = 1$, and $Pr(v_k) = 1 - Pr(x_k)$ if $v_k = 0$. The function *Pr* that, when applied on the BDD of f , returns the probability associated with this fault tree, is defined by the equations

$$\begin{aligned}
Pr(0) &= 0 \\
Pr(1) &= 1 \\
Pr(f) &= (1 - Pr(x_k)) \times Pr(f_{\overline{x_k}}) + \\
&\quad Pr(x_k) \times Pr(f_{x_k}), \quad \text{if } f \text{ is different from } 0 \text{ and } 1,
\end{aligned}$$

which show that this probability can be evaluated in $O(|f|)$. In the same way, the conditional and partial probabilities of basic events can be computed in $O(|f|)$ [9].

6 Experimental Results

Table 1 gives the characteristics of some real life fault trees treated using METAPRIME. The trees *b5000* to *bind* come from the aircraft industry and the fault trees *elf1* to *edf2* come from the nuclear power industry. For each tree, the table gives the number of basic events of the fault tree (**#Basics**), its number of gates (**#Gates**), its number of modules (**#Modules**), and its number of prime implicants (**#Primes**). Table 2 gives, for each tree, the size of the BDD of the fault tree (**|BDD|**), the size of the metaproduct denoting its prime implicants (**|MP|**), the CPU time (**CPU time**) in seconds on a Sun SPARC 2 for loading the tree into memory, computing its BDD, its probability, and its prime implicants, and the memory space, in megabytes, needed to perform this computation (**Memory**).

Tree	#Basics	#Gates	#Modules	#Primes
<i>b5000</i>	110	254	21	5256
<i>b8060</i>	103	248	6	8060
<i>b12100</i>	170	427	28	12143
<i>b14200</i>	122	204	32	14217
<i>b16200</i>	51	81	18	16200
<i>b19500</i>	121	233	20	19518
<i>b26000</i>	276	600	70	25988
<i>bred</i>	49	85	13	27778
<i>bind</i>	109	182	63	82000000000
<i>elf1</i>	60	122	1	46188
<i>elf2</i>	32	66	1	5630
<i>elf3</i>	92	106	1	24386
<i>edf1</i>	291	122	23	579720
<i>edf2</i>	382	427	3	20807446

Table 1. Characteristics of fault trees.

Tree	BDD	MP	CPU time	Memory
<i>b5000</i>	2937	821	2.3 s	1.0 Mb
<i>b8060</i>	6792	1458	3.8 s	1.5 Mb
<i>b12100</i>	58375	2159	15.7 s	4.0 Mb
<i>b14200</i>	493	698	1.0 s	0.5 Mb
<i>b16200</i>	65	120	0.7 s	0.5 Mb
<i>b19500</i>	843	916	1.2 s	0.5 Mb
<i>b26000</i>	16427	2719	7.6 s	3.0 Mb
<i>bred</i>	63	98	0.8 s	0.5 Mb
<i>bind</i>	150	298	0.8 s	0.5 Mb
<i>elf1</i>	4488	5534	3.9 s	1.5 Mb
<i>elf2</i>	545	483	1.8 s	0.5 Mb
<i>elf3</i>	8283	4205	5.2 s	2.0 Mb
<i>edf1</i>	2876	3811	3.2 s	1.0 Mb
<i>edf2</i>	158548	39584	369 s	15.0 Mb

Table 2. Experimental results on Sun SPARC 2.

All of these trees can be treated by METAPRIME in a few seconds of CPU time and with a small memory, except for the tree *edf2* that requires 6 minutes and 15 megabytes. As far as we know, no available fault tree analysis tool has ever been able to compute all the prime implicants of the fault trees *bred* to *edf2* and to evaluate their exact associated probabilities. Note that all but one of these trees are made of a very small number of modules, which means that their analysis is not reducible into the analysis of simpler trees.

The results clearly demonstrate that $|\mathbf{MP}|$ is not related to the numbers of prime implicants of the trees. For instance $|\mathbf{MP}| = 39584$ for *edf2* that has 20807446 prime implicants. Assuming that the best extensive representation of a product is to use 2 bits per literal, the ratio between the memory spaces needed to represent the prime implicants of this tree in extension and with a metaproduct, respectively, is 2509 (a vertex of a BDD takes 20 bytes). The best ratio is obtained for *bind*, whose $8.2 \cdot 10^{10}$ prime implicants are represented by a metaproduct of size 298, which means a ratio greater than 300 millions.

Acknowledgments

The authors would like to thank Emmanuel Ledinot and Damien Alexandre from Dassault Aviation, France, Marc Bouissou from Electricité de France, and Damien Ehret from Elf Aquitaine, France, for providing us with the real life fault trees that have been used to evaluate the approach presented here.

References

- [1] S. B. Akers, “Binary Decision Diagrams”, *IEEE Trans. on Computers*, Vol C-27, N°6, pp. 509–516, 1978.
- [2] T. C. Bartee, I. L. Lebow, I. S. Reed, *Theory and Design of Digital Machines*, McGraw-Hill, 1962.
- [3] K. S. Brace, R. L. Rudell, R. E. Bryant, “Efficient Implementation of a BDD Package”, in Proc. of *27th IEEE Design Automation Conference*, pp. 40–45, Orlando FL, June 1990.
- [4] R. E. Brayton, G. D. Hachtel, C. T. McMullen, A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.
- [5] R. E. Bryant, “Graph-Based Algorithms for Boolean Functions Manipulation”, *IEEE Trans. on Computers*, Vol C35, N°8, pp. 677–692, August 1986.
- [6] O. Coudert, C. Berthet, J. C. Madre, “Verification of Synchronous Sequential Machines Based on Symbolic Execution”, in *Lecture Notes in Computer Science: Automatic Verification Methods for Finite State Systems*, Vol. 407, J. Sifakis Editor, Springer-Verlag, pp. 365–373, June 1989.
- [7] 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.
- [8] O. Coudert, J. C. Madre, “Implicit and Incremental Computation of Primes and Essential Primes of Boolean functions”, in Proc. of *29th IEEE Design Automation Conference*, Anaheim CA, June 1992.
- [9] O. Coudert, J. C. Madre, “Fault Tree Analysis: 10^{20} Prime Implicants and Beyond”, in Proc. of the *IEEE Annual Symposium on Reliability*, Atlanta NC, January 1993.
- [10] D. F. Hassl, “Advanced Concepts in Fault Tree Analysis”, in Proc. of the *System Safety Symposium*, Seattle OR, June 1965.
- [11] N. Ishiura, H. Sawada, S. Yajima, “Minimization of Binary Decision Diagrams Based on Exchanges of Variables”, in proc. of *IEEE Int’l Conference on Computer Aided Design’91*, pp. 472–475, Santa Clara CA, November 1991.
- [12] S.-W. Jeong, B. Plessier, G. D. Hachtel, F. Somenzi, “Variable Ordering for FSM Traversal, in Proc. of the *Int’l Workshop on Logic Synthesis*, Microelectronics Center of North Carolina, Research Triangle Park NC, May 1991.
- [13] T. Kohda, E. J. Henley, K. Inoue, “Finding Modules in Fault Trees”, *IEEE Trans. on Reliability*, Vol. 38, No. 2, pp. 165–176, June 1989.
- [14] J. C. Madre, O. Coudert, “A Logically Complete Reasoning Maintenance System Based on a Logical Constraint Solver”, in Proc. of *Int’l Joint Conference on Artificial Intelligence’91*, pp. 294–299, Sydney, Australia, August 1991.
- [15] S. Malik, A. R. Wang, R. K. Brayton, A. Sangiovanni-Vincentelli, “Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment”, in Proc. of *IEEE Int’l Conference on Computer Aided Design’88*, Santa Clara CA, pp. 6–9, novembre 1988.
- [16] E. L. Jr. McCluskey, “Minimization of Boolean Functions”, in *Bell System Techniques*, Vol 35, pp. 1417–1444, 1959.
- [17] F. A. patterson-Hine, B. V. Koen, “Direct Evaluation of Fault Trees Using Object Oriented Programming Techniques”, in *IEEE Trans. on Reliability*, Vol. 38, No. 2, pp. 186–192, 1989.

- [18] W. V. O. Quine, “The problem of Simplifying Truth Functions”, in *American Mathematics Monthly*, Vol. 59, pp. 521–531, 1952.
- [19] W. V. O. Quine, “On Cores and Prime Implicants of Truth Functions”, in *American Mathematics Monthly*, Vol. 66, 1959.
- [20] Reactor Safety Study. “An assessment of accident risks in US commercial nuclear power plants”, WASH 1400 (NUREG 74/014), US NRC, October 1975.
- [21] R. Reiter, J. de Kleer, “Foundation of Assumption-Based truth Maintenance Systems: Preliminary Report”, in Proc. of *6th AAAI*, pp. 183–188, 1987.
- [22] W. G. Schneeweiss, “Disjoint Boolean Products via Shannon Expansion, in *IEEE Trans. on Reliability*, Vol. 33, pp. 329–332, 1984.
- [23] W. G. Schneeweiss, “Fault Tree Analysis Using a Binary Decision Tree, in *IEEE Trans. on Reliability*, Vol. 34, pp. 353–357, 1985.
- [24] J. R. Slage, C. L. Chang, R. C. T. Lee, “A New Algorithm for Generating Prime Implicants”, in *IEEE Trans. on Computers*, Vol C-19(4), pp. 304–310, 1970.
- [25] P. Tison, “Generalized Consensus Theory and Application to the Minimization of Boolean Functions”, in *IEEE Trans. on Electronic Computers*, Vol. EC-16/4, pp. 446–456, 1967.

Authors

Dr. Olivier Coudert; Bull Corporate Research Center; Rue Jean Jaurès; 78340 Les Clayes-sous-bois FRANCE

Olivier Coudert (born in France, October 29, 1964) received his engineering degree from Ecole Centrale de Paris in 1987, and his Ph.D. in Computer Science from Ecole Nationale Supérieure des Télécommunications, Paris, France, in 1991. He joined Bull in 1988, where he has been mainly active in the areas of automated reasoning, logics, and finite state machine verification.

Dr. Jean Christophe Madre; Bull Corporate Research Center; Rue Jean Jaurès; 78340 Les Clayes-sous-bois FRANCE

Jean Christophe Madre (born in France, December 21, 1961) received his engineering degree from Ecole Nationale Supérieure d’Informatique et Mathématiques Appliquées de Grenoble in 1984, and his Ph.D. in Computer Science from Ecole Nationale Supérieure des Télécommunications, Paris, France, in 1990. He joined Bull in 1985, and Bull Research Center in 1987. His areas of interest include circuit design, formal verification of hardware, and automated reasoning.

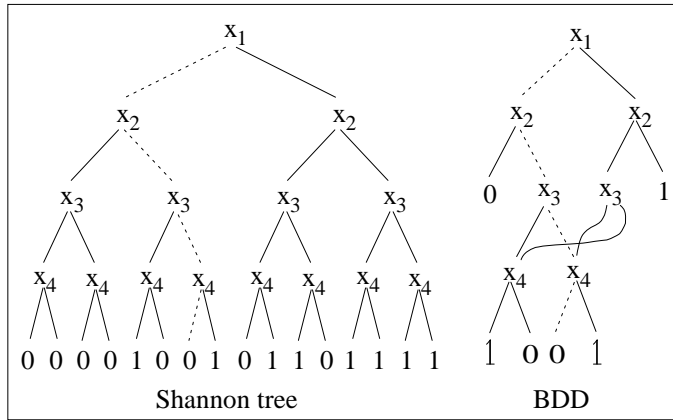


Figure 1. Shannon tree and BDD of

$$f(x_1, \dots, x_4) = (x_1 \wedge (x_3 \oplus x_4)) \vee (x_2 \wedge (x_3 \oplus \bar{x}_4)).$$

The dotted path corresponds with the assignment $x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0$.

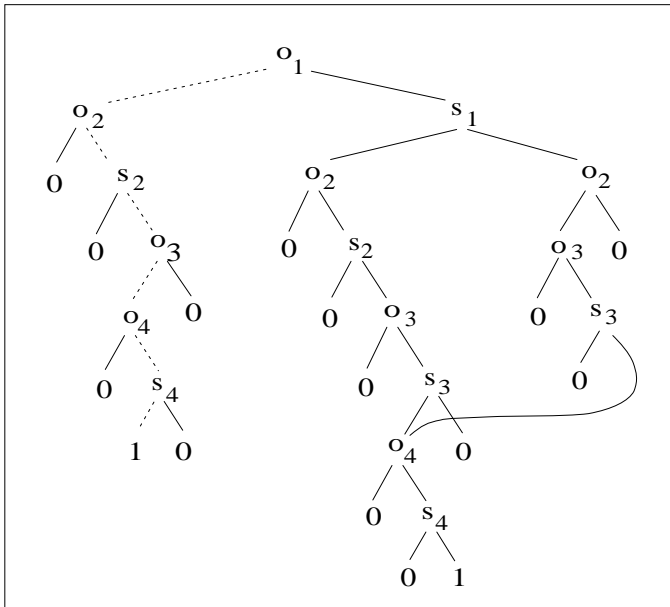


Figure 2. BDD of metaproduct of

$$\{x_2\bar{x}_4, x_1x_3x_4, \bar{x}_1x_2\bar{x}_3x_4\}.$$

The dotted path corresponds with the product $x_2\bar{x}_4$.

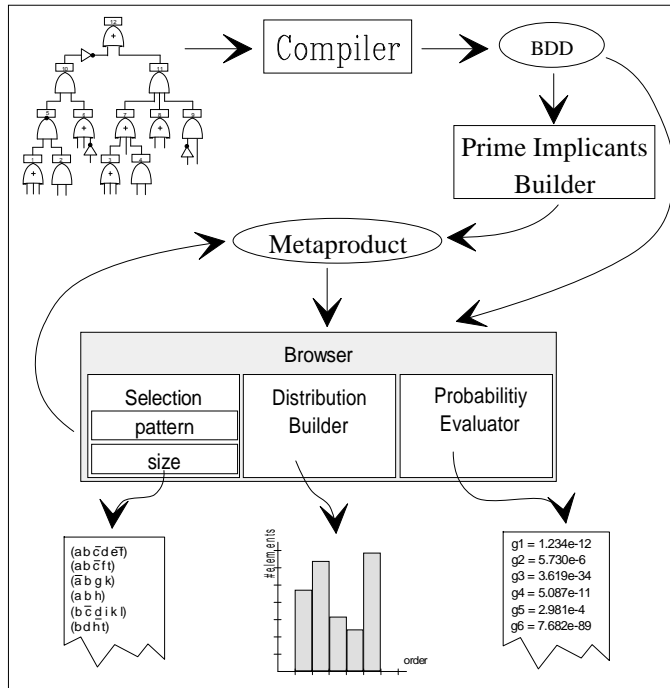


Figure 3. Architecture of the METAPRIME system.

Figure 3. Architecture of the METAPRIME system.