

Fault Tree Analysis: 10²⁰ Prime Implicants and Beyond

Olivier Coudert • Bull Corporate Research Center • Les Clayes sous Bois France
Jean Christophe Madre • Bull Corporate Research Center • Les Clayes sous Bois France

Key Words: Fault tree analysis, binary decision diagrams, metaproducts

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 *explicitly*, so that their complexities are *directly related* to the number of prime implicants to be generated. This paper presents a new analysis method of coherent as well as noncoherent fault trees that overcomes this limitation because its computational cost is *not related* to either the number of variables or the number of gates or the number of prime implicants of these trees. The interactive fault tree analyser METAPRIME that is based on this new method has been shown by experience to be able to perform in seconds the complete analysis of noncoherent fault trees with more than 10²⁰ prime implicants.

1 INTRODUCTION

Fault tree analysis has two complementary aspects. The *qualitative* aspect of the fault tree analysis consists of computing the set of *prime implicants* or *primes* of the Boolean function denoted by the tree under analysis, that are called the *minimal cut sets* in the particular case where the tree is coherent [10]. The *quantitative* aspect of the problem consists of evaluating, using the laws of probabilities associated with the terminal events of the fault tree, the probability associated with the top event of this tree.

Computing the set of primes of a fault tree with n terminal events and evaluating its associated probability are problems that are both *exponential* with respect to n . Though many efforts have been made on these problems over the past decades, currently available fault tree analysis tools, when dealing with a tree that has a complex structure, can only compute a subset of the primes of this tree, for instance those with either the smallest order or the largest probability, and these primes are used to produce a sometimes rough approximation of the probability of the tree.

This paper presents a fault tree analysis procedure that dramatically overcomes the limitations of all previously

known analysis procedures. This procedure is based on two new technologies, the *binary decision diagram* and the *implicit prime computation* technologies. The combined properties of these technologies make the computational complexity of this new analysis procedure independent from the number of terminal events, the number of gates, and the number of prime implicants of the fault tree to be analysed.

The paper is divided in 6 parts. Section 2 presents the BDD representation and gives its main characteristics. Section 3 presents the canonical *metaproduct* representation of sets of products built out of a finite set of Boolean variables, and explains how the metaproduct of the set of primes of any Boolean function can be computed. Section 4 explains how these concepts are used in fault tree analysis. Section 5 gives experimental results obtained with the procedure presented here, and Section 6 discusses them.

2 BINARY DECISION DIAGRAMS

The Shannon decomposition of a propositional formula f with respect to one of its variables x is the couple of formulas $(f_{\overline{x}}, f_{x_k})$ [1], such that: $f \Leftrightarrow (\neg x \wedge f_{\overline{x}}) \vee (x \wedge f_{x_k})$. When iterated for all the variables of f , the decomposition process defined above associates the formula f with its *Shannon tree* which is unique modulo the variable ordering used during the decomposition [1]. Figure 1 shows the Shannon tree of the formula $(x_1 \wedge (x_3 \oplus x_4)) \vee (x_2 \wedge (x_3 \Leftrightarrow x_4))$. The value of the formula for any assignment of its variables can be found by following the path that this assignment defines in the tree, for instance the path defined by the assignment $x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0$ is marked with a dotted line in Figure 1.

The binary decision diagram (BDD) of the formula f is the compacted representation of its Shannon tree obtained for a given variable ordering. This BDD is obtained after having applied two compaction rules on this tree [3]. The first compaction rule consists of eliminating all the vertices of the tree that have isomorphic sons because these vertices do not contain any valuable information about the formula. For instance the vertices marked with a “*” in Figure 1 are useless. The second compaction rule consists of identifying

all remaining isomorphic subtrees, so that the tree becomes an acyclic directed graph called the BDD of f . The BDD of the formula $(x_1 \wedge (x_3 \oplus x_4)) \vee (x_2 \wedge (x_3 \Leftrightarrow x_4))$ is shown in Figure 1.

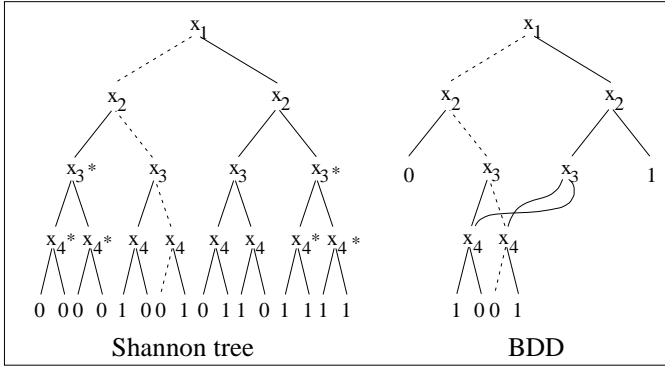


Figure 1. Shannon tree and BDD of the formula $(x_1 \wedge (x_3 \oplus x_4)) \vee (x_2 \wedge (x_3 \Leftrightarrow x_4))$.

The binary Boolean operators can be evaluated with a quadratic complexity, and the negation in linear time, on BDDs built with the same variable ordering [3]. This polynomial complexity is a remarkable property that makes BDDs very different from previously used representations of propositional formulas, for instance the disjunctive normal form for which the operators have different complexities, most of them being at least *exponential* [6].

Any subset of $\{0, 1\}^n$ can be represented with a unique Boolean function called its characteristic function. Since there is no relation between the number of elements in this set and the size of the BDD that denotes its characteristic function, huge Boolean sets can be represented with small BDDs [5], and the binary set operations can be performed *implicitly* on these sets with complexities related not to their sizes but to the sizes of the BDDs that denote them.

3 PRIME COMPUTATION

3.1 DEFINITIONS

The set P_n of products that can be built out of the set of variables $\{x_1, \dots, x_n\}$ is isomorphic to the set of strings $\{x_1, \overline{x_1}, \varepsilon\} \times \dots \times \{x_n, \overline{x_n}, \varepsilon\}$, where ε is the empty string, and the literals x_k and $\overline{x_k}$ are the variable x_k and its negation respectively. For any products p and p' , p contains p' , which is noted $p \succeq p'$, if and only if the formula $(p' \Rightarrow p)$ is a tautology.

An element x of $\{0, 1\}^n$ is an *interpretation* of the Boolean function f from $\{0, 1\}^n$ into $\{0, 1\}$, if and only if $f(x) = 1$. The product p of P_n is said to be an *implicant* of f if and only if the formula $(p \Rightarrow f)$ is a tautology. The product p is said to be a *prime implicant* or *prime* of f if and only if it is an implicant of f and there is no other implicant of f that contains p [13]. For instance, the product x_1 is a prime

of the function $f = x_1 \vee x_1x_2$, and x_1x_2 is an implicant of f but it is not prime.

3.2 METAPRODUCTS

There are 3^n elements in the set of products P_n , so $\lceil n \log_2(3) \rceil$ Boolean variables are sufficient to represent any subset of P_n . The implicit prime computation method presented here makes use of a Boolean space of dimension $2n$, that is thus larger than necessary, but that allows the encoding of elements of P_n to be strongly related to Shannon decomposition.

The many-to-one mapping σ from the set $\{0, 1\}^n \times \{0, 1\}^n$ onto the set P_n is defined in the following way [6]:

$$\sigma([o_1 \dots o_n], [s_1 \dots s_n]) = l_1 \dots l_n,$$

where $l_k = \varepsilon$ if $o_k = 0$, l_k is $\overline{x_k}$ 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 $x_2\overline{x_3}x_4$. In the sequel we will note o and s the vectors $[o_1 \dots o_n]$ and $[s_1 \dots s_n]$ respectively.

The *metaproduct* \mathcal{P} of a subset of products P of P_n is the characteristic function of the set:

$$\left(\bigcup_{p \in P} \sigma^{-1}(p) \right).$$

Since the set $(\bigcup_{p \in P_n} \{\sigma^{-1}(p)\})$ is a partition of $\{0, 1\}^n \times \{0, 1\}^n$, metaproducts are a canonical functional representation of subsets of P_n [6]. For instance the metaproduct of the subset $\{\overline{x_1}x_2x_3, x_1\overline{x_2}x_4, x_1x_2\overline{x_3}x_4\}$ of P_4 is the characteristic function of the set $\{([1110], [0110]), ([1110], [0111]), ([1101], [1001]), ([1101], [1011]), ([1111], [1100])\}$. The direct correspondence between the set and the logical operations allows the manipulations on sets of products to be performed implicitly on their metaproducts. For instance, the function $(\mathcal{P} \vee \mathcal{P}')$ is the metaproduct of the union of the sets of products denoted by the metaproducts \mathcal{P} and \mathcal{P}' respectively.

The relation between the metaproduct representation and Shannon decomposition is made explicit in the following way. Consider a subset P of P_n and its metaproduct \mathcal{P} . The Shannon decomposition of the function \mathcal{P} with respect to the variable o_k provides us with the couple of functions $(\mathcal{P}_{\overline{o_k}}, \mathcal{P}_{o_k})$. The metaproduct $(\overline{o_k} \wedge \mathcal{P}_{\overline{o_k}})$ denotes the subset of elements of P in which neither the literal $\overline{x_k}$ nor the literal x_k occur. The metaproduct $(o_k \wedge \mathcal{P}_{o_k})$ denotes the subset of elements of P in which at least one of these literals occurs. This subset is the disjoint union of two sets which are the set of products in which the literal $\overline{x_k}$ occurs and the set of products in which the literal x_k occurs, respectively. These sets are denoted by the functions $(o_k \wedge \overline{s_k} \wedge \mathcal{P}_{o_k \overline{s_k}})$ and $(o_k \wedge s_k \wedge \mathcal{P}_{o_k s_k})$ respectively, where the couple of functions $(\mathcal{P}_{o_k \overline{s_k}}, \mathcal{P}_{o_k s_k})$ is the Shannon decomposition of the function \mathcal{P}_{o_k} with respect to the variable s_k .

The relation defined above makes possible to perform many operations on sets of products in an implicit way with complexities polynomial with respect to the sizes of the BDDs of their metaproducts, 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 $\{1, \dots, n\}$. In the sequel 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.

3.3 IMPLICIT PRIME COMPUTATION

The computation of the metaproduct of the set of primes of a Boolean function is based on the following theorem which shows that this set can be computed recursively using Shannon decomposition [2].

Theorem 1 *Consider a Boolean function f from $\{0, 1\}^n$ into $\{0, 1\}$ and the variable x_k . The set of primes $\text{Prime}(f)$ of the function f is equal to*

$$\begin{aligned} & \{\bar{x}_k\} \times (\text{Prime}(f_{\bar{x}_k}) \setminus \text{Prime}(f_{\bar{x}_k} \wedge f_{x_k})) \cup \\ & \{x_k\} \times (\text{Prime}(f_{x_k}) \setminus \text{Prime}(f_{\bar{x}_k} \wedge f_{x_k})) \cup \\ & \text{Prime}(f_{\bar{x}_k} \wedge f_{x_k}). \end{aligned}$$

This theorem has an immediate corollary from which it is immediate to build the algorithm that produces the BDD of the metaproduct of the set of primes of f from the BDD of f [7].

Theorem 2 *Consider a Boolean function f from $\{0, 1\}^n$ into $\{0, 1\}$ and the variable x_k . The metaproduct $\mathcal{P}\text{rime}(f)$ of the set of primes of the function f is equal to*

$$\begin{aligned} & \bar{o}_k \wedge \mathcal{P}\text{rime}(f_{\bar{x}_k} \wedge f_{x_k}) \vee \\ & o_k \wedge \bar{s}_k \wedge (\text{Prime}(f_{\bar{x}_k}) \wedge \neg \mathcal{P}\text{rime}(f_{\bar{x}_k} \wedge f_{x_k})) \vee \\ & o_k \wedge s_k \wedge (\mathcal{P}\text{rime}(f_{x_k}) \wedge \neg \mathcal{P}\text{rime}(f_{\bar{x}_k} \wedge f_{x_k})). \end{aligned}$$

4 FAULT TREE ANALYSIS

The theorems given in the preceding section state that the BDD representing the metaproduct $\mathcal{P}\text{rime}(f)$ of the set of primes of the Boolean function f can be obtained from the BDD of f using the standard Boolean operators. This section shows that these two BDDs are sufficient to compute, in times that are linear with respect to the sizes of these BDDs, the qualitative as well as the quantitative informations that designers are interested in.

4.1 FROM FAULT TREES TO BDDS

Building the BDD of a fault tree is a straightforward operation that is performed by traversing this tree in a depth

first manner and by building the resulting BDD while going up [3]. The only problem here is to determine the variable ordering used to build this BDD. The size of the BDD of a Boolean function can heavily depend on the variable ordering that has been chosen to build this graph [3, 8]. Since determining the best variable ordering is a NP-hard problem, many efforts have been made to develop ordering heuristics that exploit the structure of formulas to produce a good variable ordering, that is an ordering for which the size of the generated BDD is tractable. Very good heuristics have been proposed to deal with multi-level logic networks [4, 12], which can be used here because fault trees are such networks. The experimental results given in section 5 have been obtained with the heuristics proposed in [12].

4.2 QUALITATIVE ANALYSIS

The relation between metaproducts and Shannon decomposition makes very easy to compute the number of products that are in the set of products P denoted by \mathcal{P} . The function Size that, when applied on \mathcal{P} , returns the number of elements of P is defined by the equations

$$\begin{aligned} \text{Size}(0) &= 0 \\ \text{Size}(1) &= 1 \\ \text{Size}(\mathcal{P}) &= \text{Size}(\mathcal{P}_{\bar{o}_k}) + \text{Size}(\mathcal{P}_{o_k \bar{s}_k}) + \text{Size}(\mathcal{P}_{o_k s_k}), \\ &\text{if } \mathcal{P} \notin \{0, 1\}. \end{aligned}$$

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 with 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 equations

$$\begin{aligned} \text{OrderDist}(0) &= [00 \dots 0] \\ \text{OrderDist}(1) &= [10 \dots 0] \\ \text{OrderDist}(\mathcal{P})[0] &= 0 \\ \text{OrderDist}(\mathcal{P})[j] &= \text{OrderDist}(\mathcal{P}_{\bar{o}_k})[j] + \\ &\text{OrderDist}(\mathcal{P}_{o_k \bar{s}_k})[j - 1] + \\ &\text{OrderDist}(\mathcal{P}_{o_k s_k})[j - 1], \\ &\text{if } j \geq 1 \text{ and } \mathcal{P} \notin \{0, 1\}. \end{aligned}$$

It is immediate, using the equations given above, to program the recursive functions Size and OrderDist . 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. This result is stored in the vertex, so these procedures have linear complexities with respect to

the number of vertices in the traversed BDDs.

4.3 QUANTITATIVE ANALYSIS

The probability $Probability(f)$ associated with the top event of a fault tree is defined in terms of the probabilities $prob(x_1), \dots, prob(x_n)$ of its terminal events and the interpretations of the Boolean function f denoted by this fault tree, in the following way:

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

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

$$\begin{aligned} Probability(0) &= 0 \\ Probability(1) &= 1 \\ Probability(f) &= (1 - prob(x_k)) * Probability(f_{\bar{x}_k}) + \\ &\quad prob(x_k) * Probability(f_{x_k}), \\ &\quad \text{if } f \notin \{0, 1\}, \end{aligned}$$

which show that this probability can be evaluated with a cost linear with respect to the size of the BDD of f .

4.4 THE BROWSER

The definition of metaproducts and the properties of BDDs actually allow a large class of operations on sets of products to be performed in times polynomial with respect to the sizes of the BDDs that denote these sets. Selecting the elements of P whose order is in a given interval can be done in $O(|P|)$. 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 $O(|P|)$. Once the metaproduct of a subset P' of primes has been selected, the exact contribution of P' to the probability of the top event of the fault tree can be computed by first generating the BDD of the function f' denoted by the sum of products in P' using the equation $f'(x) = (\exists o \mathcal{P}'(o, x))$ [6], and then in applying the function $Probability$ on this BDD.

The browser of the prototype tool METAPRIME takes advantage of the very low cost of the functions described in the previous sections to allow the reliability engineer to interact with the analysis tool in order for fully exploiting the information contained in the fault trees of the system under design. A typical interactive session with METAPRIME thus begins with loading a fault tree into memory and computing its binary decision diagram and the BDD that represents its primes, and then the user enters an analysis loop whose elementary step consists of selecting, from either the set of primes of the tree or from the result of a previous

selection, the primes that have precise characteristics and then of refining the analysis of this subset.

5 EXPERIMENTAL RESULTS

Table 1 and Table 2 give the experimental results obtained with the prototype tool METAPRIME developed at Bull using the concepts presented here. All the fault trees treated here come from real life applications. The trees b^* come from the aircraft industry and the fault trees e^* come from the nuclear power industry. For each of these examples, column **#A** of Table 1 gives the number of terminal events of the fault tree, column **#G** its number of gates, column **#Primes** its number of primes, and column **[..]** the interval of orders of these primes. Column **|BDD|** of Table 2 gives the size of the BDD of the fault tree, column **|MP|** the size of the metaproduct denoting its set of primes, column **CPU Time** the CPU time in seconds needed to compute these primes on a Sun SPARC Station 2 with 30 Mbytes of main memory, and finally column **Memory** the memory space, in megabytes, needed to perform this analysis.

Tree	#A	#G	#Primes	[..]
<i>b26000</i>	276	600	25988	[1 .. 4]
<i>bind</i>	109	182	82000000000	[10 .. 22]
<i>b14200</i>	122	204	14217	[2 .. 7]
<i>b16700</i>	53	83	16707	[7 .. 15]
<i>b16200</i>	51	81	16200	[2 .. 5]
<i>b17300</i>	51	71	17280	[6 .. 6]
<i>b19500</i>	121	233	19518	[1 .. 6]
<i>b12100</i>	170	427	12143	[1 .. 8]
<i>bred</i>	49	85	27778	[1 .. 11]
<i>elf1</i>	60	122	46188	[2 .. 11]
<i>elf3</i>	92	106	24386	[2 .. 11]
<i>edf1</i>	291	122	579720	[1 .. 6]
<i>edf3</i>	382	427	20807446	[1 .. 12]

Table 1. Characteristics of fault trees.

The CPU times given in Table 2 are dedicated to the following tasks. The first task consists of parsing the fault tree, identifying its modules using a technique similar to the one presented in [10], and determining the variable ordering. The second step consists of computing the BDD of the Boolean function represented by the top event of the fault tree, and the probability associated with this event. The third step consists of computing the BDD of the metaproduct of the set of primes of the top event of the tree, and the last step consists of building the order distribution of these primes.

The performances obtained for the faults trees b^* have been compared with those obtained with a prime computation method [11] that manipulates sets of products explicitly and uses an algorithm similar to the one used in [2]. The tree *bind* cannot obviously be treated with such a method because of the much too large number of primes to be computed. With METAPRIME, this tree is analysed in 1 second, and the resulting metaproduct uses 6000 bytes

(each vertex of a BDD takes 20 bytes). Assuming that the best explicit representation of a product is to use 2 bits per literal, the ratio between the memory spaces needed to represent the primes of this tree explicitly and implicitly respectively is greater than 400 millions. This illustrates the compactness of the metaproduct representation. Moreover this tree cannot even be partially analysed by any previously known technique because all its primes have at least 10 literals, and there are 10077696 primes with this order. For the other trees, METAPRIME was found to be between 100 and 1000 times more efficient than the method referenced above. As far as we know, no available fault tree analysis tool has ever been able to compute all the primes of the fault trees *edf**, or to evaluate their exact associated probabilities. Note that neither *edf1* nor *edf3* can be decomposed into modules, which means that their analysis is not reducible into the analysis of simpler trees.

Tree	BDD	MP	CPU Time	Memory
<i>b26000</i>	16427	2719	8.2 s	3.0 Mb
<i>bind</i>	150	298	1.0 s	0.5 Mb
<i>b14200</i>	493	698	1.0 s	0.5 Mb
<i>b16700</i>	100	177	0.8 s	0.5 Mb
<i>b16200</i>	65	120	0.8 s	0.5 Mb
<i>b17300</i>	51	102	0.8 s	0.1 Mb
<i>b19500</i>	843	916	1.2 s	0.5 Mb
<i>b12100</i>	58375	2159	18.0 s	4.0 Mb
<i>bred</i>	63	98	0.8 s	0.5 Mb
<i>elf1</i>	4488	5534	4.2 s	1.5 Mb
<i>elf3</i>	8283	4205	5.7 s	2.0 Mb
<i>edf1</i>	2876	3811	3.4 s	1.0 Mb
<i>edf3</i>	158548	39584	385.0 s	18.0 Mb

Table 2. Analysis times on SUN SPARC Station 2.

The diagram shown in Figure 2 is a compilation of the results presented here as well as other prime computations performed on a set of benchmark problems [14]. On

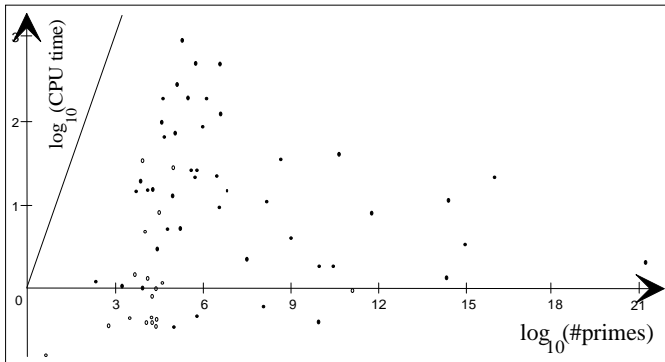


Figure 2. Summary diagram of experimental results.

the *x*-axis of this diagram are the logarithms of the numbers of primes to be computed, and on the *y*-axis are the logarithms of the CPU times in seconds, obtained on a Sun SPARC Station 2 computer, needed to compute

the metaproducts representing these sets of primes. The straight line on the left of the diagram corresponds to the CPU times that would be obtained using a procedure with a cost linear with respect to the numbers of primes to be computed. The drawing shows that all computation times for the method presented here are sub-linear with respect to the number of primes to be computed. Moreover, the regression coefficient for these results is almost equal to 0, which corroborates the theoretical independence between the number of primes of the function and the time needed to compute these primes.

6 CONCLUSION

In this paper we have presented the concepts that underly the prototype tool METAPRIME, and the experimental results obtained with METAPRIME on real life fault trees showing that these concepts allow us to analyse in seconds fault trees that could not be analysed by any previously known technique.

These performances are made possible by the dramatically different computation paradigm that is used in METAPRIME. Instead of operating on elements of $\{0,1\}^n$ and products represented explicitly as all previously known techniques do, METAPRIME uses characteristic functions to represent sets of such elements implicitly. The properties of these representations, combined with the efficiency of Boolean manipulations on binary decision diagrams, make the computational cost of the noncoherent fault tree analysis presented in this paper *independent* of the number of interpretations as well as the number of primes of the treated fault trees.

There certainly exist fault trees that the technique presented here would not be able to handle. Several causes could be responsible for such a failure, and research work is being done to deal with them. It could happen that METAPRIME would not be able to build the BDD of the fault tree under analysis. This problem can be tackled in different ways. The first way consists of determining whether there exists a variable ordering that would allow this building to be completed. Recent research efforts have demonstrated that variable ordering heuristics such as the one used in METAPRIME can be very much enhanced by using a special kind of binary decision diagrams call partial binary decision diagrams [4]. Another technique, called variable reordering [9], has been shown by experience to help BDD based tools to handle more complex problems. Partial binary decision diagrams have also been shown to be very good for making approximations in BDD computations, which would allow METAPRIME, in the case where no good variable ordering can be found, to make approximations in the fault tree analysis.

Though for most of the trees treated here the set of primes has a smaller BDD than the BDD of the tree, it could hap-

pen that METAPRIME is able to build the BDD of a fault tree but would not be able to build the BDD of its set of primes. In this case it is possible to make the approximation used in all previously available tools, which consists of computing only the primes whose order is less than a user defined bound or the primes whose associated probability is greater than a user defined bound. These procedures are under development.

The concepts that have been presented here can be used for the analysis of event trees because such trees denote Boolean functions on which these concepts can be applied. Prime 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 [11]. The applications of the technology underlying METAPRIME to these problems is under study.

References REFERENCES

- [1] S. B. Akers, "Binary Decision Diagrams", *IEEE Transactions on Computers*, Vol C-27, N°6, 1978.
- [2] R. E. Brayton, G. D. Hachtel, C. T. McMullen, A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.
- [3] R. E. Bryant, "Graph-Based Algorithms for Boolean Functions Manipulation", *IEEE Transactions on Computers*, Vol C35, N°8, pp. 677–692, August 1986.
- [4] K. M. Butler, D. E. Ross, R. Kapur, M. R. Mercer, "Heuristics to Compute Variable Orderings for Efficient Manipulations of Ordered Binary Decision Diagrams", in Proc. of *28th Design Automation Conference*, pp. 417–420, San Francisco, California, June 1991.
- [5] 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*, Volume 407, J. Sifakis Editor, Springer-Verlag, pp. 365–373, June 1989.
- [6] O. Coudert, J. C. Madre, "A New Method to Compute Prime and Essential Prime Implicants of Boolean Functions", in Proc. of *Brown/MIT Conference on Advanced Research in VLSI and Parallel Systems*, Cambridge MA, USA, March 1992.
- [7] O. Coudert, J. C. Madre, "Implicit and Incremental Computation of Primes and Essential Primes of Boolean functions", in Proc. of the *Design Automation Conference*, Anaheim, California, June 1992.
- [8] S. J. Friedman, K. J. Supowit, "Finding the Optimal Variable Ordering for Binary Decision Diagrams", *IEEE Transactions on Computer*, Vol. C-39, N° 5, pp. 710–713, May 1990.

- [9] N. Ishiura, H. Sawada, S. Yajima, "Minimization of Binary Decision Diagrams Based on Exchanges of Variables", in proc. of *IEEE International Conference on Computer Aided Design'91*, Santa Clara, California, November 1991.
- [10] 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.
- [11] J. C. Madre, O. Coudert, "A Logically Complete Reasoning Maintenance System Based on a Logical Constraint Solver", in Proc. of *IJCAI'91*, Sydney, Australia, August 1991.
- [12] 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 *ICCAD'88*, Santa Clara, USA, pp. 6–9, novembre 1988.
- [13] W. V. O. Quine, "The problem of Simplifying Truth Functions", in *American Mathematics Monthly*, Vol. 59, pp. 521–531, 1952.
- [14] S. Yang, *Logic Synthesis and Optimization Benchmarks User Guide*, Microelectronics Center of North Carolina, January 1991.

BIOGRAPHIES

Olivier Coudert
 Jean Christophe Madre
 Bull Research Center
 Rue Jean Jaurès
 78340 Les Clayes sous Bois France

Olivier Coudert 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, finite state machine verification, and logic synthesis.

Jean Christophe Madre 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.