

Towards an Interactive Fault Tree Analyser

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

presented at the
IASTED International Conference on
Reliability, Quality Control and Risk Assessment
November 1992, Washington D.C., U.S.A.

May 14, 2011

Abstract

Currently available fault tree analysis techniques are not powerful enough to allow engineers to fully exploit these trees. Large fault trees require hours of CPU time on mainframe computers to be analysed, and this analysis provides designers only with *approximate* qualitative and quantitative informations. This paper presents data structures and algorithms that support the *interactive exact* analysis of very large coherent and noncoherent fault trees. The prototype tool METAPRIME that implements this method offers modeling engineers the possibility of analysing with an uncomparable finesse fault trees with billions of prime implicants.

KeyWords

Fault tree analysis, binary decision diagrams, meta-products, Shannon decomposition.

1 Introduction

Since its introduction in the 1960's [8], fault tree analysis has helped to dramatically improve the reliability of very complex systems, such as chemical and nuclear plants, aircrafts and spacecrafts, for which failures can result in very important and unacceptable damages [18, 22]. Fault tree analysis is very useful because it allows engineers to reason about the failures of the systems they design, and it provides them with *qualitative* as well as *quantitative* information about the reliability of these systems [22].

The qualitative information is obtained by analysing the set of *prime implicants* of a fault tree. The quantitative information is obtained by evaluating the probability associated with the top event of a tree using the probabilities associated with its terminal events. Computing the set of prime implicants of a fault tree with n terminal events and evaluating its associated probability are problems that are respectively exponential [17] and NP-hard with respect to n . Though much effort has been spent on these problems over the past decades [3, 10, 14, 15, 17, 19, 20, 21], currently available fault tree analysis techniques are not powerful enough to fulfill the needs of designers. Except for small fault trees, these procedures can only provide designers with *approximate* qualitative and quantitative informations. Moreover these informations are

obtained after hours of computation, which definitely prevents designers from extracting all the information that is contained in the fault trees they build.

This paper presents the data structures and algorithms supporting an interactive 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) that are a canonical representation of Boolean functions that has been used for several years in the field of digital circuit design because it supports very efficiently many procedures that are of utmost importance in this field [4, 6, 11]. As a consequence, the computational cost of the analysis of a fault tree using this method is *completely independent* of its size, of its number of interpretations, and of its number of prime implicants.

The paper is divided in 6 parts. Section 2 introduces the elementary concepts that will be used in the sequel. Section 3 presents the key properties of the binary decision diagrams and Section 4 those of the meta-product representation used to represent set of prime implicants in intension. Section 5 describes the interactive fault tree analysis method. Section 6 gives experimental results obtained with the prototype tool METAPRIME that implements this method, and discusses them.

2 Definitions

We consider here propositional formulas built out of a finite set of variables noted x_1, \dots, x_n and the constants 0 and 1, using the logical connectors $\{\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow, \oplus\}$. Assuming that these variables are ordered, such a propositional formula f denotes a unique Boolean function from $\{0, 1\}^n$ into $\{0, 1\}$. Since this Boolean function is unique, we will use the same symbol to represent the formula and the function it denotes. A formula that denotes the function 1 is called a tautology. The Shannon decomposition of a propositional formula f with respect to one of its variables x_k is the unique couple of functions $(f_{\overline{x_k}}, f_{x_k})$ denoted by the couple of formulas $(f[x_k \leftarrow 0], f[x_k \leftarrow 1])$, where the formula $f[x_k \leftarrow g]$ is obtained by substituting each occurrence of x_k in f with the formula g . This couple of functions is such that: $f = (\neg x_k \wedge f_{\overline{x_k}}) \vee (x_k \wedge f_{x_k})$.

A *literal* is a variable x_k or its negation also noted $\overline{x_k}$. The set P_n of products that can be built out of the set of variables $\{x_1, \dots, x_n\}$ is 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. A *product* is an element of this set, and it represents the conjunction of its literals. For instance, $x_1 \overline{x_2} x_4$ represents the formula $(x_1 \wedge \neg x_2 \wedge x_4)$. The product p contains the product 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 formula f if the function denoted by f evaluates to 1 on x . 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, and it is said to be a *prime implicant* of f if and only if it is an implicant of f and there is no other implicant of f that contains p [17].

A Boolean function f from $\{0, 1\}^n$ into $\{0, 1\}$ denotes a unique subset S_f of $\{0, 1\}^n$ that is defined by the equation $S_f = \{x \in \{0, 1\}^n \mid f(x) = 1\}$. Conversely a subset S of $\{0, 1\}^n$ is denoted by a unique Boolean function from $\{0, 1\}^n$ into $\{0, 1\}$, called its *characteristic function*, and noted χ_S , that evaluates to 1 on all elements of S and to 0 for all other elements of $\{0, 1\}^n$. Characteristic functions provide us with a representation in intension of the subsets of $\{0, 1\}^n$. There is an obvious correspondence between set operators and Boolean operators, for instance, union corresponds with disjunction, intersection with conjunction, and complementation with negation.

3 Binary Decision Diagrams

When iterated for all the variables of a propositional formula f , the Shannon decomposition associates this formula with a 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. 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, from the root of the tree to a leaf, that this assignment defines in the tree. At each vertex, this path follows the left branch if the associated variable is assigned the value 0, and the right branch if the variable is assigned the value 1. 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.

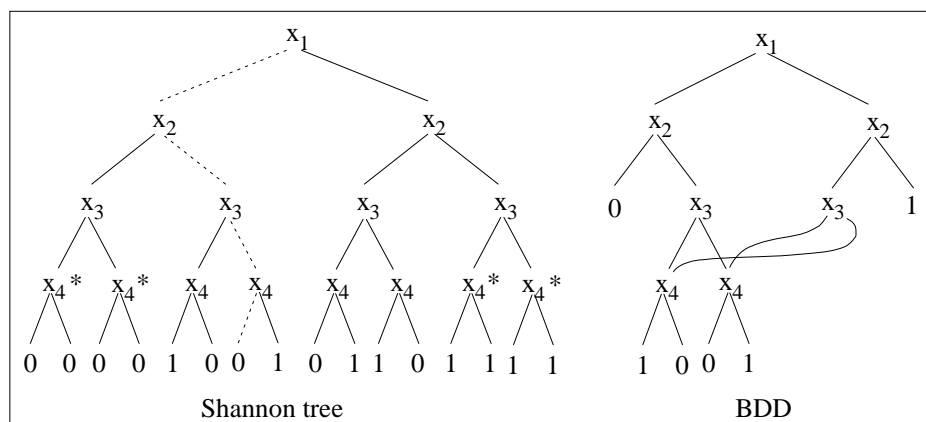


Figure 1. Shannon tree and BDD of the formula $(x_1 \wedge (x_3 \oplus x_4)) \vee (x_2 \wedge (x_3 \Leftrightarrow x_4))$.

The Binary Decision Diagram (BDD) of a formula f is the compacted representation of the Shannon tree of f obtained for a given variable ordering. This BDD is produced by applying two compaction rules on this tree [4], that consist of first eliminating all its vertices that have isomorphic sons (the vertices marked with a “*” in Figure 1 are such vertices) and then of identifying all isomorphic subtrees of the resulting tree, so that this tree becomes an acyclic directed graph. The BDD of the formula $(x_1 \wedge (x_3 \oplus x_4)) \vee (x_2 \wedge (x_3 \Leftrightarrow x_4))$ is shown by Figure 1.

The Boolean operators $\{\vee, \wedge, \Rightarrow, \Leftrightarrow, \oplus\}$ can be evaluated with a quadratic complexity on BDDs built with the same variable ordering [4]. Negation can be evaluated in linear time on BDDs and in constant time on typed decision graphs that are an improved form of BDDs [11]. 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* [7]. BDDs are a very interesting representation of Boolean functions and sets because there is *no relation at all* between the number of elements in a set and the size of the BDD that denotes this set through its characteristic function, so that huge Boolean sets can potentially be denoted by small BDDs [6]. The correspondence existing between the set operators and the Boolean operators allows us to perform operations on sets with a complexity that is not related to the number of elements of these sets but to the sizes of the BDDs that denote them.

4 The Meta-Product Representation

4.1 Definition

The many-to-one mapping σ from the set $\{0, 1\}^n \times \{0, 1\}^n$ into the set P_n is defined in the following way [7]: $\sigma([o_1 \dots o_n], [s_1 \dots s_n]) = l_1 \dots l_n$, 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 $x_2\bar{x}_3x_4$. In the sequel we will note o and s the vectors $[o_1 \dots o_n]$ and $[s_1 \dots s_n]$ respectively.

The *meta-product* \mathcal{P} of the subset of products $P = \{p_1, \dots, p_m\}$ of P_n is the characteristic function of the set $(\cup_{k=1}^m \sigma^{-1}(p_k))$. Since the set $(\cup_{p \in P_n} \{\sigma^{-1}(p)\})$ is a partition of $\{0, 1\}^n \times \{0, 1\}^n$, meta-products are a canonical functional representation of subsets of P_n [7]. For instance, the meta-product of the subset $\{x_2\bar{x}_4, x_1x_3x_4, x_1\bar{x}_2x_3\bar{x}_4\}$ of P_4 is the characteristic function of the set $\{([0101, 0100]), ([0101], [0110]), ([0101], [1100]), ([010, 1110]), ([1011], [1011]), ([1011], [111]), ([1111], [0101])\}$.

4.2 Properties

Meta-products have properties that are direct consequences of their definition. The usual operations on set of products can be performed on their meta-products using the corresponding Boolean operations. For instance, the function $(\mathcal{P} \vee \mathcal{P}')$ is the meta-product of the union of the sets of products denoted by \mathcal{P} and \mathcal{P}' respectively.

Consider a subset P of P_n and its meta-product \mathcal{P} . The Shannon decomposition of the function \mathcal{P} with respect to the variable o_k provides us with the couple of functions $(\mathcal{P}_{\bar{o}_k}, \mathcal{P}_{o_k})$. The meta-product $(\bar{o}_k \wedge \mathcal{P}_{\bar{o}_k})$ denotes the subset of elements of P in which neither the literal (\bar{x}_k) nor the literal (x_k) occur. The meta-product $(o_k \wedge \mathcal{P}_{o_k})$ 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 (\bar{x}_k) occurs and the set of products in which the literal (x_k) occurs, respectively. These sets are denoted by the functions $(o_k \wedge \bar{s}_k \wedge \mathcal{P}_{o_k\bar{s}_k})$ and $(o_k \wedge s_k \wedge \mathcal{P}_{o_k s_k})$ respectively.

The following theorem, which will be used to evaluate the exact contribution of a subset of the set of prime implicants of a fault tree to its associated probability, establishes the relation existing between a meta-product \mathcal{P} and the Boolean function denoted by the sum of all the products denoted by \mathcal{P} [7].

Theorem 1 *Let P be a set of products, \mathcal{P} its meta-product, and f be the Boolean function denoted by the sum of the products of P . Then: $f = \lambda s. (\exists o \mathcal{P}(o, s))$.*

The properties given above make many operations on sets of products linear with respect to the size of the BDD of the meta-products 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 $\{1, \dots, n\}$. In the following we will only consider the case where this permutation is the same as the one defining the variable ordering used to build the BDD of the formula under treatment.

5 Fault Tree Analysis

5.1 Architecture of the system

Figure 3 describes the architecture of the interactive fault tree analysis tool METAPRIME. This tool gives engineers interactive access, through a command shell, to a unique set of fault tree analysis procedures. After loading a fault tree into memory, the user can choose an event of this tree, compute the BDD of the Boolean function of this event, then the meta-product of its set of prime implicants, and then enter an analysis loop in which the basic step consists of one of the following operations:

- Selecting a subset S of this set of prime implicants.

- Computing the number of elements of S .
- Computing the distribution of the elements of S by size.
- Evaluating the contribution of the elements of S to the probability associated with the current event.
- Pretty printing the elements of S .

Figure 2. Architecture of the METAPRIME system.

5.2 Computing the BDD of a Fault Tree

Building the BDD representing a fault tree can be done through a two step process. First step consists of identifying the modules of the tree and of determining the ordering of the propositional variables associated with its terminal events. Second step consists of traversing this tree in a depth first way and of computing its associated BDD in a bottom up way. This is done by combining the BDDs associated with the inputs of each gate using the logical operator of this gate. Fault trees are multi-level logic networks [13] so the variable ordering heuristics that have been proposed for such networks can be applied on fault trees. In METAPRIME we use the heuristics proposed in [13] that orders the variables according to their appearance during a traversal of the network guided by depth and fanout information associated with the gates of this network.

The efficiency of the heuristics referenced above is very much improved by first decomposing the network to be treated into modules [10]. Decomposing a network into modules consists in expressing the Boolean function f associated with this network as a composition of functions f_1, \dots, f_m whose supports of variables are disjoint. This decomposition, that can be performed recursively, has a dramatic effect on the size of the BDD of f since it has been shown that this size is linear with respect to the sum of the sizes of the BDDs of f_1, \dots, f_m [9]. It also allows subsequent operations on this BDD to have reduced complexities.

5.3 Qualitative Analysis

5.3.1 Prime Implicant Computation

The computation of the meta-product $\mathcal{P}rime(f)$ of the set of prime implicants of the Boolean function f is based on the following theorem which establishes the relation between the meta-product $\mathcal{P}rime(f)$ and the meta-products $\mathcal{P}rime(f_{\bar{x}_k})$, $\mathcal{P}rime(f_{x_k})$, and $\mathcal{P}rime(f_{\bar{x}_k} \wedge f_{x_k})$.

Theorem 2 Consider a Boolean function f from $\{0, 1\}^n$ into $\{0, 1\}$ and one of its variable x_k . The meta-product $\mathcal{P}rime(f)$ of the set of prime implicants of the function f is equal to

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

This theorem shows that the cost of computing the BDD of the meta-product $\mathcal{P}rime(f)$ is polynomial with respect to the sizes of the BDDs of the meta-products $\mathcal{P}rime(f_{\overline{x_k}})$, $\mathcal{P}rime(f_{x_k})$, and $\mathcal{P}rime(f_{\overline{x_k}} \wedge f_{x_k})$. It also gives the structure of the 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 treat is exponential with respect to the BDD of f because for each vertex of this BDD the algorithm has to treat the BDD of $(f_{\overline{x_k}} \wedge f_{x_k})$ that can contain vertices that are neither in the BDD of $f_{\overline{x_k}}$ nor in the one of f_{x_k} . However note that in the case where the function f is unate, the function $(f_{\overline{x_k}} \wedge f_{x_k})$ is equal to $f_{\overline{x_k}}$ so the number of recursions that are necessary to compute $\mathcal{P}rime(f)$ is exactly equal to the size of the BDD of f .

5.3.2 Prime Implicant Distribution

The property of meta-products given in Section 4.2 makes very easy to compute the number of products that are in a set P of products by working on the BDD of its meta-product \mathcal{P} . The function $Size$ that, when applied on \mathcal{P} , returns the number of elements of P is defined by the following equations: $Size(0) = 0$, $Size(1) = 1$, and

$$Size(\mathcal{P}) = Size(\mathcal{P}_{\overline{o_k}}) + Size(\mathcal{P}_{o_k \overline{s_k}}) + Size(\mathcal{P}_{o_k s_k}), \text{ if } \mathcal{P} \notin \{0, 1\}.$$

All elements of P_n have a size less or equal to n so the distribution of 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 size j . The function $OrderDist$ that, when applied on the meta-product \mathcal{P} , returns the array representing the distribution of the elements of P , is defined by the following equations: $OrderDist(0) = [00 \dots 0]$, $OrderDist(1) = [10 \dots 0]$, and

$$\begin{aligned} OrderDist(\mathcal{P})[0] &= 0 \\ OrderDist(\mathcal{P})[j] &= OrderDist(\mathcal{P}_{\overline{o_k}})[j] + \\ & OrderDist(\mathcal{P}_{o_k \overline{s_k}})[j - 1] + \\ & OrderDist(\mathcal{P}_{o_k s_k})[j - 1], \text{ if } j \geq 1, \mathcal{P} \notin \{0, 1\}. \end{aligned}$$

It is immediate, using the equations given in the preceding paragraphs, 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. In order for treating each vertex once during the traversal, the procedures store in each vertex of the BDD the result obtained for this vertex. The complexity of these functions is thus in $O(|\mathcal{P}|)$.

5.4 Quantitative Analysis

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

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

where $prob(v_k) = p(x_k)$ if $v_k = 1$, and $prob(v_k) = 1 - p(x_k)$ if $v_k = 0$. The function *Prob* that, when applied on the BDD of f , returns the probability associated with this fault tree, is defined by the following equations which show that this probability can be evaluated in $O(|f|)$: $Prob(0) = 0$, $Prob(1) = 1$, and

$$Prob(f) = (1 - p(x_k))Prob(f_{\overline{x_k}}) + p(x_k)Prob(f_{x_k}), \text{ if } f \notin \{0, 1\}.$$

5.5 The Browser

The definition of meta-products 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 size is in a given interval can be done in $O(|\mathcal{P}|)$. Building the BDD of the meta-product of the subset of elements of P that contain a given set of literals and do not contain another set of literals can also be done in $O(|\mathcal{P}|)$. More generally, selecting the elements of P that satisfy a property that can be expressed by a propositional formula f can be done in $O(|\mathcal{P}| \times |f|)$. Once the meta-product of a subset P of prime implicants 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 g denoted by the sum of products of P using Theorem 1, and then in applying the function *Prob* on this BDD. Since experience shows that these BDDs are rather small this means that such operations can be performed interactively by the modeling engineer who can thus perform very precise qualitative and quantitative analysis.

6 Experimental Results

Table 1 gives the CPU times that have been obtained with the prototype tool METAPRIME, written in C, using the concepts presented here. The fault trees b^* are real life fault trees coming from the aircraft industry, and the fault trees e^* come from the nuclear power industry. For each of these examples, the column **#I#G** gives the number of terminal events and the number of gates of the fault tree, column **|BDD|** the size of its BDD, column **#Prime** its number of prime implicants, column **|MP|** the size of the meta-product denoting these prime implicants, **Time** the CPU time in seconds on a Sun SPARC Station 2 workstation[†] to analyse this tree, and **Memory** the memory space, in megabytes, needed to perform this analysis.

The CPU times given in Table 1 are dedicated to the followings tasks. The first task consists of parsing the fault tree, of identifying its modules, and of determining the variable ordering. The second step consists of computing the different BDDs that represent the Boolean functions represented by each module of the fault tree, of composing them to get the complete BDD of the tree, and of evaluating the exact probability associated with the tree. The third step consists of computing the BDD of the meta-product of the set of prime implicants of the top event of the tree, and of building the order distribution of these prime implicants.

As far as we know, there is no currently known technique that can be used to compute all the prime implicants of the fault trees *elf1*, *elf2*, *elf3*, *edf1*, *edf2*, and to evaluate their exact associated probabilities. Each of these trees can be treated by METAPRIME in less than 1 minute of CPU time and with a memory space less or equal than 2 megabytes, except for the tree *edf2* that requires 15 megabytes and about 12 minutes. Note that none of these trees can be decomposed into modules which means that their analysis is not reducible into the analysis of simpler trees.

The performances obtained for the faults trees *b2500* to *bred* have been compared with those obtained with a prime implicant computation method [12] that manipulates sets of products in extension and uses an algorithm similar to the one used in [2, 3]. The tree *bind* cannot obviously

[†]SUN SPARCStation 2 is a trademark of Sun Microsystems, Inc.

be treated with such a method because of the much too large number of prime implicants to be computed. With METAPRIME, this tree is analysed in less than 3 seconds, and the resulting BDD occupies 6000 bytes (each vertex of a BDD takes 20 bytes). 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 in intension with a meta-product respectively is greater than 400 millions. This illustrates the efficiency of the meta-product representation. Moreover this tree cannot even be partially analysed by any previously known technique because its smallest prime implicants have 10 literals, and there are 10077696 such smallest prime implicants. For the other trees, METAPRIME was found to be about 1000 times more efficient than the method referenced above.

Tree	#I/#G	BDD	#Prime	[MP]	Time	Memory
<i>b560</i>	32/13	28	560	56	0.4 s	0.5 Mb
<i>b2500</i>	50/72	50	2376	100	1.1 s	0.5 Mb
<i>b5000</i>	110/254	2937	5256	821	3.7 s	0.5 Mb
<i>b9000</i>	50/72	50	9216	100	1.1 s	0.5 Mb
<i>b26000</i>	276/600	16427	25988	2719	20 s	2.0 Mb
<i>b8060</i>	103/248	6792	8060	1458	13 s	1.5 Mb
<i>bind</i>	109/182	150	8210 ⁹	298	2.1 s	0.5 Mb
<i>b14200</i>	122/204	493	14217	698	1.8 s	0.5 Mb
<i>b16700</i>	53/83	100	16707	177	1.0 s	0.5 Mb
<i>b16200</i>	51/81	65	16200	120	1.0 s	0.5 Mb
<i>b17300</i>	51/71	51	17280	102	1.0 s	0.1 Mb
<i>b19500</i>	121/233	842	19518	916	2.2 s	0.5 Mb
<i>b12100</i>	170/427	58375	12143	2159	88 s	4.0 Mb
<i>bred</i>	49/85	63	27778	98	1.0 s	0.5 Mb
<i>elf1</i>	60/122	4488	46188	5534	37 s	1.5 Mb
<i>elf2</i>	32/66	545	5630	483	2.1 s	1.0 Mb
<i>elf3</i>	92/106	8283	24386	4205	38 s	2.0 Mb
<i>edf1</i>	291/122	2876	579720	3811	2.6 s	1.0 Mb
<i>edf2</i>	382/427	158548	20807446	39584	700 s	15.0 Mb

Table 1. Experimental results on Sun SPARC Station 2.

7 Conclusion

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

Though for most of the trees treated here the set of prime implicants has a smaller BDD than the BDD of the tree, it can happen that METAPRIME is able to build the BDD of a fault tree but is not able to build the BDD of its set of prime implicants. In this case it is possible to make the approximation used in all previously available tools that consists in computing only the implicants whose order is less than a user defined bound or the implicants whose associated probability is greater than a user defined bound.

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 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 use of the technology underlying METAPRIME to these problems is under study.

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 Transactions on Computers*, Vol C-27, N°6, 1978.
- [2] T. C. Bartee, I. L. Lebow, I. S. Reed, *Theory and Design of Digital Machines*, McGraw-Hill, 1962.
- [3] R. E. Brayton, G. D. Hachtel, C. T. McMullen, A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.
- [4] R. E. Bryant, “Graph-Based Algorithms for Boolean Functions Manipulation”, *IEEE Transactions on Computers*, Vol C35, N°8, pp. 677–692, August 1986.
- [5] 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.
- [6] O. Coudert, J. C. Madre, “A Unified Framework for the Formal Verification of Sequential Circuits”, in Proc. of *ICCAD’90*, Santa Clara CA, USA, Novembre 1990.
- [7] 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.
- [8] D. F. Hassl, “Advanced Concepts in Fault Tree Analysis”, in Proc. of the *System Safety Symposium*, Seattle, Oregon, June 1965.
- [9] S.-W. Jeong, B. Plessier, G. D. Hachtel, F. Somenzi, “Variable Ordering for FSM Traversal”, in Proc. of the *International Workshop on Logic Synthesis*, Microelectronics Center of North Carolina, Research Triangle Park, North Carolina, May 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, J. P. Billon, “Proving Circuit Correctness using Formal Comparison Between Expected and Extracted Behaviour”, in Proc. of *25th Design Automation Conference*, July 1988.
- [12] 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.
- [13] 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.
- [14] E. L. Jr. McCluskey, “Minimization of Boolean Functions”, in *Bell System Techniques*, Vol 35, pp. 1417–1444, 1959.
- [15] F. A. patterson-Hine, B. V. Koen, “Direct Evaluation of Fault Trees Using Object Oriented Programming Techniques”, in *IEEE Transactions on Reliability*, Vol. 38, No. 2, pp. 186–192, 1989.
- [16] W. V. O. Quine, “The problem of Simplifying Truth Functions”, in *American Mathematics Monthly*, Vol. 59, pp. 521–531, 1952.

- [17] W. V. O. Quine, “On Cores and Prime Implicants of Truth Functions”, in *American Mathematics Monthly*, Vol. 66, 1959.
- [18] Reactor Safety Study. “An assessment of accident risks in US commercial nuclear power plants. WASH 1400 (NUREG 74/014), US NRC, October 1975.
- [19] W. G. Schneeweiss, “Fast Fault Tree Evaluation for Many Sets of Input Data”, in *IEEE Transactions on Reliability*, Vol. 39, No. 3, pp. 296–300, 1990.
- [20] J. R. Slagle, C. L. Chang, R. C. T. Lee, “A New Algorithm for Generating Prime Implicants”, in *IEEE Transactions on Computers*, Vol C-19(4), pp. 304–310, 1970.
- [21] P. Tison, “Generalized Consensus Theory and Application to the Minimization of Boolean Functions”, in *IEEE Transactions on Electronic Computers*, Vol. EC-16/4, pp. 446-456, 1967.
- [22] A. Villemeur, *Sûreté de fonctionnement des systèmes industriels*, Eyrolles, Paris, France 1988.