A Brief Introduction to Binary Decision Diagrams

Antoine Rauzy

LaBRI - URA CNRS 1304 - Université Bordeaux I
351, cours de la Libération,
33405 Talence Cedex (France)
`rauzy@labri.u-bordeaux.fr`

ABSTRACT. *This paper aims to be a brief introduction to Binary Decision Diagrams and some of their uses in the reliability analysis framework. It is a computer scientist look on that question, i.e. it focuses on data structures and algorithms. It tries to give an overview of implementation issues at an abstract level. Basic algorithms are reviewed while recent developments are only mentioned.*
KEY WORDS : *Binary Decision Diagrams.*

1. Introduction

Automated resolution of many practical problems requires to store very large collections of objects. This is especially the case for those expressed in terms of Boolean formulae. To solve them one has to memorize and to manipulate the truth tables of the formulae under study (or equivalently sum-of-products forms for these formulae), i.e. *a priori* at least a significant part of the 2^n possible assignments of Boolean values to their n variables. Both qualitative and quantitative analyses of fault trees raise this kind of problems [VES 81, LEE 85, LIM 91].

If the studied Boolean formulae were generated at random, there would be no hope to get a compact representation for them [SHA 49]. Fortunately, real-life fault trees or combinatorial circuits are not, as far as we know, generated at random. It is thus possible to design data structures that capture, at least to a certain extent, their regularities. I.e. that encode their truth tables within an amount of memory that is not directly related to the number of variable assignments satisfying them.

Binary Decision Diagrams (BDD's for short) are such a data structure. BDD's are the state-of-the-art data structure to encode and manipulate Boolean functions. They have been introduced by B. Akers [AKE 78] and improved by R. Bryant [BRY 86, BRA 90]. They are nowadays used in a wide range of areas, including hardware synthesis and verification, protocol validation and automated deduction (see [BRY 92] for a survey on BDD's and their applications). Their use in the reliability analysis framework has been initiated by J.-C. Madre and O. Coudert on the one hand [COU 92a, COU 92b], and the author on the other hand [RAU 93]. BDD's are sometimes called branching programs in the theoretical computer science literature [WEG 88].

This paper aims to be a brief introduction to BDD's and some of their uses for reliability analyses. First, some definitions and vocabulary about propositional calculus are recalled section 2. The data structure and its basic properties are presented section 3. Data structures have no intrisic meaning. So, functions must be defined that associates to each data structure its semantics, i.e. a mathematical object. Different interesting semantics can be defined for BDD's. Four of them are presented section 4. BDD's are always built in a compositional way, i.e. starting from basic BDD's and combining them to obtain more complex ones. The most important functions to do so are sketched section 5. Worst case sizes of ROBDD's and heuristics to get small size ROBDD's are discussed section 6. Section 7 shows how ROBDD's are used to compute the probability of the root event of a fault tree given the probabilities of its terminal events and how to compute and to encode the set of its prime implicants in a compact way.

2. Background

This section recalls basic vocabulary of Boolean algebra and graph theory we need in this paper. A more detailed presentation could be found in [SCH 89].

2.1. *Boolean Algebra*

Boolean formulae are terms inductively built over the two (Boolean) constants 0 (False) and 1 (True), a denumerable set of variables $\mathcal{V}$, and the usual logical connectives $\wedge$ (and), $\vee$ (or), $\neg$ (not). For instance, $(a \wedge \neg b) \vee (\neg a \wedge c)$ is a Boolean formula built over the variables a, b and c. Other connectives are sometimes used. For instance, $\oplus$ denotes the connective 'exclusive or'. Recall that $f \oplus g$ is equivalent to $(f \wedge \neg g) \vee (\neg f \wedge g)$.

A *literal* is either a variable x or its negation $\neg x$. x and $\neg x$ are said *opposite.* A *product* is a set of literals that does not contain both a literal and its opposite. A product is assimilated with the conjunction of its elements. A set of products is assimilated with the disjunction of its elements. Sets of products are also called *sum-of-products* or formulae in *disjunctive normal form* (DNF).

We denote by $Var[f]$ the set of variables occurring in a Boolean formula f.

Let f be a Boolean formula. An *assignment* of $Var[f]$ is any mapping from $Var[f]$ to $\{0,1\}$. For instance, $[a \leftarrow 1, b \leftarrow 0, c \leftarrow 1]$ is an assignment of $Var[(a \vee \neg b) \wedge (\neg a \vee c)]$. An assignment σ *satisfies* (resp. *falsifies*) a formula f if $\sigma(f) = 1$ (resp. $\sigma(f) = 0$). For instance, $[a \leftarrow 1, b \leftarrow 0, c \leftarrow 1]$ satisfies $(a \vee \neg b) \wedge (\neg a \vee c)$, while $[a \leftarrow 1, b \leftarrow 1, c \leftarrow 1]$ falsifies it.

Let f and g be Boolean formulae. g *implies logically* f if any assignment satisfying g satisfies f. We denote that by $g \models f$. If both $f \models g$ and $g \models f$, f and g are said equivalent. We denote that by $f \equiv g$.

A product can be seen as a *partial assignment* of the variables of a formula: the value 1 (resp. 0) is assigned to each variable occurring positively (resp. negatively) in the product, while the other variables remain unassigned. We say that a product σ *satisfies* (resp. *falsifies*) a given Boolean formula f if f is satisfied (resp. falsified) by any (total) assignment of $Var[f]$ that agrees with σ. For instance, $\{a, \neg b\}$ satisfies $(a \vee \neg b) \wedge (\neg a \vee c)$. The set of assignments satisfying a formula can thus be seen as a sum-of-products.

A product σ that satisfies a function f is also called an *implicant* of f. Let f be a Boolean formula and σ be an implicant of f. σ is said *prime* if there is no implicant γ of f strictly included in σ. In what follows, we denote by $Prime[f]$ the set of prime implicants of a formula f. For instance, $Prime[(a \wedge \neg b) \vee (\neg a \wedge c)] = \{\{a, \neg b\}, \{\neg a, c\}, \{\neg b, c\}\}$.

When they contain no negative literal, prime implicants are often called *minimal cuts* in the reliability analysis literature.

2.2. *Graphs*

A *graph* is given by a set of nodes (also called *vertices*) U together with a set of edges $E \subseteq U \times U$ (edges are thus pairs of nodes). In what follows, we consider only *directed* graphs which means that pairs are ordered.

An edge (u, v) is an *out-edge* for the node u and an *in-edge* for the node v. We also say that the edge (u, v) points to the node v. A node without in-edge is called a *root* node, while a node without out-edge is called a *sink* node. Sink nodes are also called *leaves*.

A *path* in the graph is a sequence of nodes $u_1, \ldots, u_k$ such that $(u_i, u_{i+1}) \in E$ for $i = 1, \ldots, k-1$. The *length* of a path is the number of edges it traverses. A node v is said to be *reachable* from a node u if there exists a path from u to v.

A graph is said acyclic if for every node u, u is not reachable from itself through a positive length path. A *tree* is a uniquely rooted directed acyclic graph in which each node but the root has exactly one in-edge.

The *subgraph* rooted by the node u of a graph $G = (U, E)$ is the graph $G' = (U', E')$, where U' is the subset of nodes of U that are reachable from u and E' is the set of edges $(v, w) \in E$ such that both v and w belong to U'.

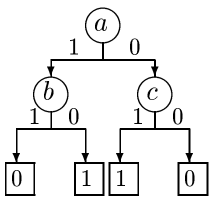

Figure 1. *A BDD over the variables a, b and c.*

Throughout this paper we deal with *labeled graphs*, i.e. with graphs that embbed data (labels) on their vertices and edges. Two edges with the same source and target nodes but different labels are considered as distinct.

Three labeled graphs are pictured Fig. 2. The leftmost one is a tree, the two others are simply acyclic.

3. Data Structures

3.1. *Reduced Ordered Binary Decision Diagrams*

Let $x_1,\ldots,x_n$ be n Boolean variables. A *binary decision diagram* over x_1,...,x_n is a directed acyclic graph α verifying conditions *(i)* and *(ii)*.

(i) Sink nodes of α are labeled either with 0 or with 1.

(ii) Each internal node of α is labeled with a variable x_i and has 2 out-edges labeled respectively with 0 and 1.

Let $<$ be a total order over $x_1,\ldots,x_n$ (say $x_1 < x_2 < \ldots < x_n$). A binary decision diagram α is said to be *ordered* if condition *(iii)* holds.

(iii) For any pair of nodes (α,β) labeled respectively with the variables x_i and x_j, if β is reachable from α then $x_i < x_j$.

As a consequence, in an ordered BDD (OBDD) with a single root node, each variable is encountered at most once onto any path from the root node to a sink node (moreover, variables are encountered according to the order $<$). The BDD pictured Fig. 1 is ordered (for the lexicographic order).

For the sake of convenience, we denote by $\Delta(x,\alpha_1,\alpha_0)$ the BDD rooted by a node labeled with the variable x and whose 1- and 0-out-edges point respectively to BDD's α_1 and α_0.

This definition of BDD's is purely syntactical, which means that a BDD is seen as data structure, just as data structure. In order to give a meaning to this construction we need a function that associates a mathematical object with each BDD. The semantics of a node $\Delta(x,\alpha_1,\alpha_0)$ should depend only on the variable x and the semantics of the nodes α_1 and α_0. It follows that two isomorphic BDD's should have the same semantics. As a consequence, it is useless to maintain copies of isomorphic BDD's. One copy suffices, which is interesting to limit memory consumption.

An ordered binary decision diagram α is said *reduced* if condition *(iv)* holds.

(iv) α *contains no isomorphic sub-graphs.*

In particular, it means that a reduced ordered binary decision diagram (ROBDD) has only two sink nodes, labeled respectively with 0 and 1.

Fig. 2 shows three equivalent OBDD's. The left most is a tree, while the right most is reduced.

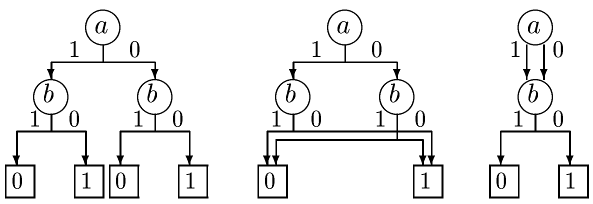

Figure 2. *Three equivalent OBDD's.*

Condition *(iv)* can be seen as a rewriting rule that "folds" OBDD's to get more compact equivalent ones. A completely unfolded OBDD is a binary tree. The following property holds.

Theorem 1 (Canonicity of ROBDD's) *For any OBDD α, the ROBDD obtained from α by sharing isomorphic subgraphs is unique up to an isomorphism.*

This theorem is very important because it allows to establish a one-to-one correspondence between ROBDD's and mathematical objects.

3.2. *Management of ROBDD's nodes*

As we will see, ROBDD's are always created in a bottom-up way, i.e. that a node $\alpha = \Delta(x, \alpha_1, \alpha_0)$ is always created after the creation of the nodes α_0 and α_1. Furthermore, once created a node is never modified.

Nodes are kept into a table and they are created only through a function *find_or_add_node*. This function takes x, α_1 and α_0 as parameters. It looks up the table and it creates actually a new node only if necessary, i.e. if the table does not already contain the node $\Delta(x, \alpha_1, \alpha_0)$. Technically, this table is a hashtable. This management of nodes has two important consequences:

1. Reduction *(iv)* is automatically performed.
2. Provided the hashtable is well dimensioned, the complexity of the *find_or_add_node* is in $\mathcal{O}(1)$, i.e. constant.

A more detailed discussion about implementation can be found in [BRA 90].

4. Four Semantics for ROBDD's

4.1. *Standard ROBDD's*

Usually, ROBDD's are interpreted as Boolean formulae under Shannon Normal Form. This semantics for ROBDD's is the standard one and usually it is implicitely understood in [AKE 78, BRY 86, BRA 90, BRY 92]. The Shannon Normal Form is based on the *ite* (for *I*f-*T*hen-*E*lse) connective which is defined as follows. Let f, g and h three Boolean functions, then

$$ite(f,g,h) \quad \overset{\text{def}}{=} \quad (f \wedge g) \vee (\neg f \wedge h)$$

Any binary connective can be expressed by means of an *ite* and possibly a negation. For instance, $f \vee g \equiv ite(f,1,g)$ and $f \oplus g \equiv ite(f,g,\neg g)$. Moreover, *ite* is orthogonal with the usual connectives.

Let Op be an n-ary operation, let $f^1,\ldots,f^n$ be n formulae and let x be a variable, Op is said *orthogonal* with *ite* if the following equivalence holds.

$$Op(f^1,\ldots,f^n) \equiv ite(x, Op(f^1_{x=1},\ldots,f^n_{x=1}), Op(f^1_{x=0},\ldots,f^n_{x=0}))$$

where $f_{x=v}$ denotes the formula f in which the constant v has been substituted for all occurrences of x.

Property 2 (Orthorgonality of usual connectives) *Connectives* $\neg$, $\vee$, $\wedge$, $\oplus$ *are orthogonal with ite.*

A formula is said to be *Shannon Normal Form* if either it is a constant or it is of the form $ite(x,f,g)$, where x is a variable and f and g are formulae in Shannon Normal Form in which x does not occur. It is clear that, thanks to orthogonality, any formula can be rewritten into an equivalent one that is in Shannon Normal Form.

For instance, $(a \wedge \neg b) \vee (\neg a \wedge c)$ is equivalent to $ite(a, ite(b,0,1), ite(c,1,0))$ which is in Shannon Normal Form.

The function $Shannon[\![.]\!]$ that associates a formula in Shannon Normal Form to each BDD is defined by the following recursive equations.

$$\begin{aligned}
Shannon[\![0]\!] &\overset{\text{def}}{=} 0 \\
Shannon[\![1]\!] &\overset{\text{def}}{=} 1 \\
Shannon[\![\Delta(x,\alpha_1,\alpha_0)]\!] &\overset{\text{def}}{=} ite(x, Shannon[\![\alpha_1]\!], Shannon[\![\alpha_0]\!])
\end{aligned}$$

For instance, the semantics of the OBDD pictured Fig. 1 is $ite(a, ite(b,0,1), ite(c,1,0))$ and the one of OBDD's pictured Fig. 2 is $ite(a, ite(b,0,1), ite(b,0,1))$.

This semantics for ROBDD's induces another reduction rule. It is actually easy to verify that, for any two Boolean functions f and g, $ite(f,g,g) \equiv g$. If a node encodes such an *ite* connective, it is useless. For instance, the ROBDD

$\Delta(b, 0, 1)$ can used instead of OBDD's pictured Fig. 2. We thus slightly change the definition of ROBDD's.

An OBDD α encoding a Boolean formula in Shannon Normal Form is said *reduced* if conditions *(iv)* and *(v)* hold.

(v) α contains no node of the form $\Delta(x, \beta, \beta)$.

The following theorem holds.

Theorem 3 (Canonicity of Shannon ROBDD's)*Let f be a Boolean function that depends on variables $x_1, \ldots, x_n$ and let $<$ be a total order over these variables. Then there exists an unique (up to an isomorphism) ROBDD encoding f w.r.t. $<$.*

This theorem has many interesting consequences. Let f and g be two Boolean formulae encoded respectively by ROBDD's α and β, then the following propositions hold.

- f is a tautology (resp. and antilogy) if and only if α is the leaf 1 (resp. 0).
- f is satisfiable if and only if α is not the leaf 0.
- $f \equiv g$ if and only if α and β are isomorphic.

This last point is interesting because if we manage ROBDD's in such way that there is never two copies of isomorphic ROBDD's (as shown section **3.2**) then testing equivalence between two functions is reduced to testing equality of the addresses of their ROBDD's.

4.2. *Functional ROBDD's*

The Reed-Muller Normal Form is another interesting normal form for Boolean formulae [BRO 90]. It is based on the following theorem.

Theorem 4 (Reed-Muller decomposition)*Let f be a Boolean function that depends on the variable x. Then there exists two Boolean functions g and h that do not depend on x such that $f \equiv (x \wedge g) \oplus h$.*

Theorem 4 induces a new possible semantics $\mathit{ReedMuller}[\![.]\!]$ for ROBDD's.

$$\begin{aligned}
\mathit{ReedMuller}[\![0]\!] &\stackrel{\text{def}}{=} 0 \\
\mathit{ReedMuller}[\![1]\!] &\stackrel{\text{def}}{=} 1 \\
\mathit{ReedMuller}[\![\Delta(x, \alpha_1, \alpha_0)]\!] &\stackrel{\text{def}}{=} (x \wedge \mathit{ReedMuller}[\![\alpha_1]\!]) \oplus \mathit{ReedMuller}[\![\alpha_0]\!]
\end{aligned}$$

For the same reasons as previously, ROBDD's provide a canonical encoding of Boolean formulae under Reed-Muller normal form. We do not develop further this semantics because it has no application to fault tree analyses (at least at the moment). It should be mentioned that it has applications in the logical circuit synthesis framework [BEC 93].

4.3. *Zero-suppressed ROBDD's*

In the reliability analysis framework, it is often the case that one has to manipulate subsets of a given set. For instance, minimal cuts of a fault tree are subsets of the set of its terminal events. Let $E = \{e_1, \ldots, e_n\}$ be a set. A subset F of E can be described by means of a Boolean function χ, so-called characteristic function, which is defined over variables $x_1, \ldots, x_n$ as follows.

$$\chi(F) \overset{\text{def}}{=} \bigwedge_{e_i \in F} x_i$$

In the same way, a set $\mathcal{F} = \{F_1, \ldots, F_m\}$ of subsets of E can be described as the disjunction of characteristic functions of the F_i's.

$$\chi(\mathcal{F}) \overset{\text{def}}{=} \bigvee_{F_i \in \mathcal{F}} \chi(F_i)$$

This gives raise to a new semantics for ROBDD's that has been formalized by S. Minato under the name Zero-suppressed BDD's (ZBDD's) in [MIN 93, MIN 94] (and implicitely used in [RAU 93]).

$$\begin{aligned}
ZBDD[\![0]\!] &\overset{\text{def}}{=} \emptyset \\
ZBDD[\![1]\!] &\overset{\text{def}}{=} \{\emptyset\} \\
ZBDD[\![\Delta(x_i, \alpha_1, \alpha_0)]\!] &\overset{\text{def}}{=} \{\{e_i\} \cup \sigma; \sigma \in ZBDD[\![\alpha_1]\!]\} \cup ZBDD[\![\alpha_0]\!]
\end{aligned}$$

For instance, the OBDD pictured Fig. 1 encodes the sets $\{a\}$ and $\{c\}$, and OBDD's pictured Fig. 2 encode the sets $\{a\}$ and $\emptyset$.

Reduction rule *(v)* does not hold for ZBDD's. However, there is a a new reduction rule, specific to ZBDD's, that is based on the equality: $ZBDD[\![\Delta(x_i, 0, \alpha_0)]\!] = ZBDD[\![\alpha_0]\!]$. For instance, in ZBDD's of Fig. 1 and 2, nodes $\Delta(b, 0, 1)$ are useless and can be replaced by leaf 1.

An ZBDD α is said *reduced* if conditions *(iv)* and *(vi)* hold.

(vi) α contains no node of the form $\Delta(x, 0, \beta)$.

The following theorem holds.

Theorem 5 (Canonicity of ZBDD's) *Let $\mathcal{F}$ be a set of subsets of a set E let $<$ be a total order over elements of E. Then, there exists an unique (up to an isomorphism) ZBDD encoding $\chi(\mathcal{F})$ w.r.t. $<$.*

4.4. *ROBDD's to encode Meta-Products*

As shown above, ZBDD's can be used to encode minimal cuts sets of fault trees in a quite simple way. Prime implicants are more tedious to encode because they may contain both positive and negative literals. A way to tackle this problem (suggested in [MIN 93]), is to create, for each variable x of the original formula, two variables, say x^+ and x^-, that denote respectively the presence of the positive and the negative literal built over x in the considered products. Another way to encode prime implicants by means of ROBDD's has been proposed by J.-C. Madre and O. Coudert in [COU 92a]. It works as follows. Two variables, p_x and s_x, are associated with each variable x of the original formula. p_x is used to encode the presence of the variable x in the products. s_x is used to encode the polarity of x in the products, if it is actually present.

The *meta-product* encoding π, denoted by $MP(\pi)$, is the conjunction, over the variables x occurring in the formula, of $mp(\pi, x)$, where $mp(\pi, x)$ is defined as follows.

$$mp(\pi, x) \stackrel{\text{def}}{=} \begin{cases} (p_x \wedge s_x) \text{ if } x \in \pi, \\ (p_x \wedge \neg s_x) \text{ if } \neg x \in \pi, \\ \neg p_x \text{ if neither } x \text{ nor } \neg x \text{ belong to } \pi. \end{cases}$$

A set of products is encoded by the disjunct of the corresponding meta-products. This gives the following equations that define the semantics of ROBDD's encoding meta-products.

$$\begin{aligned} MP[\![0]\!] &\stackrel{\text{def}}{=} \emptyset \\ MP[\![1]\!] &\stackrel{\text{def}}{=} \{\emptyset\} \\ MP[\![\Delta(p_x, \alpha_1, \alpha_0)]\!] &\stackrel{\text{def}}{=} MP[\![\alpha_1]\!] \cup MP[\![\alpha_0]\!] \\ MP[\![\Delta(s_x, \alpha_1, \alpha_0)]\!] &\stackrel{\text{def}}{=} \{\{x\} \cup \sigma; \sigma \in MP[\![\alpha_1]\!]\} \cup \{\{\neg x\} \cup \sigma; \sigma \in MP[\![\alpha_0]\!]\} \end{aligned}$$

Note that the canonicity of the representation is ensured by the canonicity of standard ROBDD's.

4.5. *Attributed Edges*

Another scheme of reduction rules consists in defining attributes (or flags) for edges. Let f be a Boolean formula in Shannon Normal Form. The Boolean formula in Shannon Normal Form equivalent to $\neg f$ is obtained from f by substituting 1's for 0's and vice-versa. From a ROBDD point of view, this operation consists in exchanging 0 and 1 leaves.

A programming trick (due to S. Minato & al [MIN 90]) makes possible a negation in constant time and reduces memory consumption. It consists in putting a flag on each edge. This flag indicates whether the pointed BDD is to be considered positively or negatively. In other words, $Shannon[\![\bullet\alpha]\!] = \neg Shannon[\![\alpha]\!]$. As a consequence, only one leaf remains necessary, say for instance the leaf 1, since $0 \equiv \neg 1$. The canonicity of the representation is maintained by storing only nodes with a positive 1-out-edge. The orthogonality

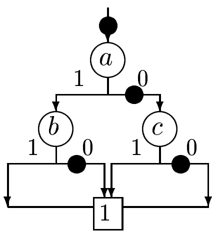

Figure 3. *The BDD encoding* $\neg ite(a, ite(b, 1, \neg 1), \neg ite(c, 1, \neg 1))$ *(negated edges are denoted with black dots)*

of *ite* and $\neg$ is used to keep only such nodes: $\Delta(x, \bullet\alpha_1, \alpha_0)$ is replaced by $\bullet\Delta(x, \alpha_1, \bullet\alpha_0)$ and $\Delta(x, \bullet\alpha_1, \bullet\alpha_0)$ is replaced by $\bullet\Delta(x, \alpha_1, \alpha_0)$. For instance, $ite(a, ite(b, 0, 1), ite(c, 1, 0))$ is rewritten into $\neg ite(a, ite(b, 1, \neg 1), \neg ite(c, 1, \neg 1))$ which is encoded by the ROBDD pictured Fig. 3.

In [MIN 93], S. Minato suggest to apply this technique to ZBDD's in the following way. A ZBDD is flagged if and only if it encodes a set containing the empty set. More complex attributes (or flags) have proposed in the literature that allow further reductions [MIN 90], but it is not clear whether these reductions improve the method in practice.

5. Operations for ROBDD's combination

ROBDD's are built in a compositional way, i.e. that basic ROBDD's are combined to get more complex ones which are in turn combined and so on. In this section, the main operations to combine ROBDD's are sketched.

5.1. *Computation of the ROBDD encoding a Boolean formula*

Let f be a formula. If f is a constant, a variable x or of the form $\neg g$, the ROBDD encoding f easy to obtain. It is the leaf eventually negated in the first case, the ROBDD $\Delta(x, 1, \bullet 1)$ in the second and the negation of the ROBDD encoding g in the third one (see section **4.5**). Otherwise, we can assume without a loss of generality that $f = g \odot h$, where $\odot$ is any usual binary connective and g and h are two formulae. In order to compute the ROBDD α encoding f, one proceeds as follows. First, one computes the ROBDD's β and γ encoding respectively g and h. Second, one combines these two ROBDD's to get α. Recursive rules for ROBDD's combination are deduced from equations giving their semantics described section **4.1**. Basically, three cases are possible:
case 1. β and γ are such that α can be immediately deduced. For instance, if $\odot$ is $\wedge$ and $\beta = \gamma$, then α is β ($g \wedge g \equiv g$ for any g). Other cases of immediate deduction occur when either β or γ (or both) are leaves. For instance, if β is the leaf 1 and $\odot$ is $\wedge$, then α is γ ($1 \wedge h \equiv h$ for any h).

case 2. $\beta = \Delta(x, \beta_1, \beta_0)$ and $\gamma = \Delta(x, \gamma_1, \gamma_0)$. Let g_1, g_0, h_1 and h_0 be respectively the formulae that are the standard semantics of respectively β_1, β_0, γ_1 and γ_0. Then, the following equivalence holds.

$$ite(x, g_1, g_0) \odot ite(x, h_1, h_0) \quad \equiv \quad ite(x, g_1 \odot h_1, g_0 \odot h_0)$$

Thus, in order to compute α, one first combines β_1 and γ_1 with $\odot$ to get a ROBDD α_1, then second one combines β_0 and γ_0 with $\odot$ to get a ROBDD α_0, and third one builds α as $\Delta(x, \alpha_1, \alpha_0)$ by means of the function *find_or_add_node* (excepted if $\alpha_1 = \alpha_0$ in which case $\alpha = \alpha_1$).
case 3. The third possibility is a degenerated case of case 2: $\beta = \Delta(x, \beta_1, \beta_0)$ and $\gamma = \Delta(y, \gamma_1, \gamma_0)$ with say $x < y$. In this case, the combination rule is the same as previously excepted that β_1 and β_0 are combined with γ itself.

The reference [BRA 90] contains more details about these operations.

5.2. *Zero-suppressed BDD's*

As for ROBDD's encoding formulae in Shannon Normal Form, Zero-suppressed BDD's are built from smaller ZBDD's that are combined through set operations. The leaves 0 ($\neg 1$) and 1 encode respectively the sets $\emptyset$ and $\{\emptyset\}$. Singletons $\{e_i\}$ are encoded by ZBDD's of the form $\Delta(x_i, 1, \bullet 1)$. Recursive rules that perform set operations on ZBDD's are derived from equations giving their semantics and are very similar to those used to combine standard ROBDD's.

For instance, let $\beta = \Delta(x_i, \beta_1, \beta_0)$ and $\gamma = \Delta(x_i, \gamma_1, \gamma_0)$ be two ZBDD's. Let G and H be the two sets encoded by resp. β and γ and assume we want to compute a ZBDD α encoding $G \cap H$. We have:

$$\begin{array}{rcl} G \cap H & = & ZBDD[\![\beta]\!] \cap ZBDD[\![\gamma]\!] \\ & = & (\{\{e_i\} \cup \sigma ; \sigma \in ZBDD[\![\beta_1]\!]\} \cup ZBDD[\![\beta_0]\!]) \\ & \cap & (\{\{e_i\} \cup \sigma ; \sigma \in ZBDD[\![\gamma_1]\!]\} \cup ZBDD[\![\gamma_0]\!]) \\ & = & \{\{e_i\} \cup \sigma ; \sigma \in ZBDD[\![\beta_1]\!] \cap ZBDD[\![\gamma_1]\!]\} \\ & \cup & (ZBDD[\![\beta_0]\!] \cap ZBDD[\![\gamma_0]\!]) \end{array}$$

Thus, in order to get the ZBDD's α encoding F, one first combines β_1 and γ_1 with $\cap$ to get a ZBDD α_1, then second one combines β_0 and γ_0 with $\cap$ to get a ZBDD α_0, and third one builds α as $\Delta(x_i, \alpha_1, \alpha_0)$ by means of the function *find_or_add_node* (excepted if $\alpha_1 = 0$ in which case $\alpha = \alpha_0$).

Similar recursive rules are defined for all the classical operations on sets such as union, difference and complementation [MIN 93].

5.3. *Operations on ROBDD's encoding Meta-Products*

Operations on sets of products can be performed through logical operations on the corresponding meta-products. This is the main advantage of this way of encoding products.

Let Σ_1 and Σ_2 be two sets of products built over the variables $x_1, \ldots, x_n$. Let $MP(\Sigma_1)$ and $MP(\Sigma_2)$ be the two corresponding meta-products. Then,

- The empty set of products is encoded by the boolean constant 0.
- The set of all of the possible products is encoded by the boolean constant 1.
- The empty product is encoded by the function $(\neg p_{x_1} \wedge \ldots \wedge \neg p_{x_n})$.
- $\Sigma_1 \cup \Sigma_2$ is encoded by the function $MP(\Sigma_1) \vee MP(\Sigma_2)$.
- $\Sigma_1 \cap \Sigma_2$ is encoded by the function $MP(\Sigma_1) \wedge MP(\Sigma_2)$.
- The complement $\overline{\Sigma_1}$ to Σ_1 in the set of all of the possible products is encoded by the function $\neg MP(\Sigma_1)$.

5.4. *Memorization of Intermediate Results*

Canonicity of ROBDD's is used to reduce the computational cost of basic operations in the following way. A table is used in which results of already performed operations are stored. For instance, this table contains 4-tuples $\{\odot, \beta, \gamma, \alpha\}$, where $\odot$ is a Boolean operation and α, β and γ are ROBDD's such that the combination of β and γ with $\odot$ gives α as result.

Before any operation, one first looks up the table to see whether its result is not already computed. If it is not the case, the operation is actually performed and its result is added to the table. This second table is also managed as a hashtable which ensures a fast access to stored tuples (see [BRA 90] for more details).

From a practical point of view, memorization of intermediate results increases dramatically the performances of operations on ROBDD's. From a theoretical point of view, it ensures that the worst case complexity of the combination of two ROBDD's α and β is in $\mathcal{O}(|\alpha| \times |\beta|)$, i.e. proportional to the product of the numbers of nodes of α and β [BRA 90] (this holds both for logical operations between ROBDD's and set operations between ZBDD's).

6. Complexity Issues

The size of sum-of-products a representation of Boolean function depends on the number of assignments satisfying the represented function. This is the reason why these representations are quickly too expensive. Many functions encountered in practice admit not too large ROBDD representations, even if they have a very large number of implicants. However, from a theoretical point of view, ROBDD's do not really improve the situation since most of Boolean functions cannot admit polynomial size ROBDD representations [WEG 88, LIA 92] ([LIA 92] is very interesting because it also studies the respective powers of reduction rules *(iv)* and *(v)*). Moreover, except for the special class of symmetric functions, the size of the ROBDD heavily depends on the chosen variable ordering. For instance, R. Bryant shows in [BRY 86] that the ROBDD encoding the function:

$$f(x_1,\ldots,x_n,y_1,\ldots,y_n) = (x_1 \wedge y_1) \otimes \ldots \otimes (x_n \wedge y_n)$$

has a linear size for the ordering $x_1 < y_1 < \ldots < x_n < y_n$ and an exponential size for the ordering $x_1 < \ldots < x_n < y_1 < \ldots < y_n$. There exists natural functions that are encoded by exponential size BDD's whatever the variable ordering is (e.g. n bits multipliers [BRY 91]).

Finding the best variable ordering is a very difficult problem. The best known algorithms are in $\mathcal{O}(3^n)$ [FRI 90, ISH 91], where n is the number of variables, which makes them impracticable.

Thus, heuristics are used to find good orderings. Many such heuristics have been proposed in the literature (see [FUJ 88, MAL 88, BER 89, CHO 90, BUT 91, FUJ 91, FUJ 93] to cite some of them). There is, as far as we know, only one common sense rule to design such a heuristics: variables that are semantically close must be close in the ordering as well. The above example helps us to understand that rule. It is clear that variables x_i and y_i are strongly linked together. With the ordering $x_1 < \ldots < x_n < y_1 < \ldots < y_n$ they are separated by n variables. Before taking a decision on the value of the sub-formula $(x_1 \wedge y_1)$ one must examine all the possible values for the x_i's. It follows that the higher part of the BDD must be a complete binary tree. Conversely, with the ordering $x_1 < y_1 < \ldots < x_n < y_n$ the sub-formulae $(x_i \wedge y_i)$ are treated one after another.

The cost of the computation of the ROBDD encoding a function is not only related to its size but also to the sizes of each ROBDD used during the computation. This is examplified by the formula $f = g \vee \neg g$ that is is encoded by the leaf 1, even g is encoded by an exponential size ROBDD. Moreover, the way a function is written influences the complexity of the computation (see papers by M. Bouissou [BOU 94] and [BOU 96] for interesting discussions on that subject). Note that, as noted by several authors, techniques used by classical algorithms to simplify fault trees, such as modularization, can be adapted successfully for ROBDD's computations [SIN 96a, PUL 96, DUT 96a].

7. Application to Fault Trees Analysis

7.1. *Quantitative Analyses*

A central problem of quantitative fault tree analyses is to determine the probability of failure of the system under study knowing the probabilities of failures of its elementary components. This problem is known to be #P-complete [VAL 79], i.e. as hard as finding the number of satisfying assignments of a Boolean formula. Once built the ROBDD associated with a formula, it is easy to compute the *exact* probability of the formula, given the probability of each variable. This is performed by means of a ROBDD traversal, the Shannon's decomposition being applied on each node of the ROBDD.

Let $p(x)$ denote the probability of the variable x. The function $p[\![.]\!]$ that associates a probability to each ROBDD can be seen as a new semantics for ROBDD's. It is computed by applying the following recursive equations.

$$\begin{aligned} p[\![0]\!] &\overset{\text{def}}{=} 0 \\ p[\![1]\!] &\overset{\text{def}}{=} 1 \\ p[\![\Delta(x,\alpha_1,\alpha_0)]\!] &\overset{\text{def}}{=} p(x).p[\![\alpha_1]\!] + (1-p(x)).p[\![\alpha_0]\!] \end{aligned}$$

If intermediate results are memorized, the complexity of this computation is linear in the size of the ROBDD. This is a major advantage of ROBDD's. Once computed the ROBDD, the cost of the computation of the probability is very cheap and can thus be reiterated many times, for instance in order to sample its evolution through the time.

ROBDD's can be used to compute other quantities of interest in the reliability analysis framework as shown in [SIN 95, SIN 96a, SIN 96b, BON 96].

7.2. *Qualitative Analyses*

Qualitative analyses of fault trees (and some quantitative analyses as well) require to compute prime implicants of the corresponding formulae [LEE 85]. ROBDD based algorithms (proposed in [COU 92a, COU 92b] and [RAU 93]) to do so use the following theorem as an inductive principle.

Theorem 6 (Decomposition Theorem)*Let $f(x_1,\ldots,x_n)$ be a Boolean function. Then, the set of prime implicants of $f(x_1,\ldots,x_n)$ is the union of the three following sets.*

- *The set $Prime[f(1,\ldots,x_n) \wedge f(0,\ldots,x_n)]$.*
- *The set of products $\{x_1\} \cup \sigma$ where σ is a product in $Prime[f(1,\ldots,x_n)]$ that doesn't belong to $Prime[f(1,\ldots,x_n) \wedge f(0,\ldots,x_n)]$.*
- *The set of products $\{\neg x_1\} \cup \sigma$ where σ is a product in $Prime[f(0,\ldots,x_n)]$ that doesn't belong to $Prime[f(1,\ldots,x_n) \wedge f(0,\ldots,x_n)]$.*

Intuitively, the above theorem is justified as follows. A prime implicant σ of $f(x_1,\ldots,x_n)$ may contain either x_1 or $\neg x_1$ or none of these two literals. In this latter case, σ must still be a prime implicant of f whatever constant is substituted for x_1. Thus, σ is a prime implicant of $f(x_1,\ldots,x_n)$ that doesn't contain x_1 nor $\neg x_1$ if and only if it is a prime implicant of $\forall x_1 f(x_1,\ldots,x_n) = f(1,\ldots,x_n) \wedge f(0,\ldots,x_n)$. Now, a product $\{x_1\} \cup \sigma$ is a prime implicant of $f(x_1,\ldots,x_n)$ if it is a prime implicant of $f(1,\ldots,x_n)$ and σ is not already a prime implicant of $f(x_1,\ldots,x_n)$, i.e. if σ doesn't belong to $Prime(f(1,\ldots,x_n) \wedge f(0,\ldots,x_n))$.

The decomposition theorem gives an inductive principle to compute prime implicants (no matter how prime these implicants are stored, i.e. either by means of ZBDD's or by means of ROBDD's encoding meta-products). See the cited articles for more details.

8. Perspectives

Several experiments on large real-life fault trees (realized with tools such as METAPRIME [COU 93, COU 94, MAD 94a] or Aralia [ARA 94, ARA 95]) have demonstrated that ROBDD's outperform by orders of magnitude classical techniques such as MOCUS [FUS 72]. Trees with several hundred of gates and (repeated) terminal events can now be handled on personal computers. Despite of these first successes, it remains many things to do.

Treatment of very large fault trees. Practitioners, knowing that they have efficient tools at their disposal, are now studying physical systems at a more detailed level. The trees they are working on are in general automatically generated (by tools such as FIGARO [BOU 91]) and sometimes so large (or complex) that they cannot be handled directly. Several techniques can be explored to tackle this complexity.
– The design of new and more powerful heuristics for variable ordering. First experiments we did in that direction are promising.
– Modularization and rewriting of formulae in such way that they become easier to handle. First works such as those presented in [BOU 96, DUT 96a] indicate that great improvements can be obtained in this way.
– Application of ROBDD's techniques such as dynamic variable re-ordering [ISH 91]. These techniques have been shown powerful in the circuit verification framework. Perhaps they can be used for fault tree analysis as well.
– The design of approximated algorithms. At least for what concerns the determination of minimal cuts, we succeeded in this way in dealing with otherwise untractable trees [ARA 95].

Applications to other risk assessment techniques. Works by J. Dugan & al. show that ROBDD's can successfully applied to many risk assessment problems, such as fault coverage [DOY 96] and analysis of dynamic fault trees [PUL 96]. In particular, one of their interesting applications is the reliability networks analysis for which first promising results have been obtained by Madre & al. [MAD 94b] (and improved in [DUT 96b]).

9. References

[AKE 78] B. Akers. "Binary Decision Diagrams". *IEEE Transactions on Computers*, 27(6):509–516, 1978.

[ARA 94] Groupe Aralia. "Arbres de Défaillances et Diagrammes Binaires de Décision". in *Actes du* 1^{er} *congrès interdisciplinaire sur la Qualité et la Sûreté de Fonctionnement*, pp 47–56. Université Technologique de Compiègne, 1994. The "Groupe Aralia" is constitued by A. Rauzy (LaBRI – Université Bordeaux I), Y. Dutuit (LADS – Université Bordeaux I), J.P. Signoret (Elf-Aquitaine – Pau), M. Chevalier (Schneider Electric – Grenoble), I. Morlaes (SGN – Saint Quentin

en Yvelines), A.M. Lapassat (CEA-LETI – Saclay), S. Combacon (CEA-IPSN – Valduc), F. Brugère (Technicatome – Aix-Les Milles), M. Bouissou (EDF-DER – Clamart).

[ARA 95] Groupe Aralia. "Computation of Prime Implicants of a Fault Tree within Aralia". in *Proceedings of the European Safety and Reliability Association Conference, ESREL'95*, pp 190–202, Bournemouth – England, June 1995. European Safety and Reliability Association.

[BEC 93] B. Becker, R. Drechsler, and M. Theobald. "On the implementation of a package for efficient representation and manipulation of functional decision diagrams". in *IFIP WG 10.5 Workshop on Applications of Reed-Muller Expansion in Circuit Design, Hamburg*, 1993.

[BER 89] C.L. Berman. "Ordered Binary Decision Diagrams and Circuit Structure". in *Proceedings of the IEEE International Conference on Computer Aided Design, ICCAD'89*, September 1989. Cambridge MA, USA.

[BON 96] J.-L. Bon and J. Collet. "L'utilisation des diagrammes de décision binaire pour le calcul de fiabilité". *RAIRO-APII-JESA*, 30(8):1103–1114, 1996. Special Issue on Binary Decision Diagrams, ISSN 0296-1598.

[BOU 91] M. Bouissou, H. Bouhadana, M. Bannelier, and N. Villatte. "Knowledge modelling and reliability processing: presentation of the FIGARO language and of associated tools". in *Proceedings of SAFECOMP'91*, Trondheim, Norway, 1991.

[BOU 94] M. Bouissou. "Une heuristique d'ordonnancement des variables pour la construction des diagrammes de décision binaires à partir d'arbre de défaillance". in *Actes du congrès $\lambda - \mu$*, pp 547–556, 1994.

[BOU 96] M. Bouissou. "An Ordering Heuristics for Building Binary Decision Diagrams from Fault-Trees". in *Proceedings of the Annual Reliability and Maintenability Symposium, ARMS'96*, 1996.

[BRA 90] K. Brace, R. Rudell, and R. Bryant. "Efficient Implementation of a BDD Package". in *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pp 40–45. IEEE 0738, 1990.

[BRO 90] F.M. Brown. *Boolean Reasoning.* Kluwer Academic Publishers, 1990.

[BRY 86] R. Bryant. "Graph Based Algorithms for Boolean Fonction Manipulation". *IEEE Transactions on Computers*, 35(8):677–691, August 1986.

[BRY 91] R. Bryant. "On the Complexity of VLSI Implementations and Graphs Representations of Boolean Functions with Application to Integer Multiplication". *IEEE Transactions on Computers*, 40(2):205–213, 1991.

[BRY 92] R. Bryant. "Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams". *ACM Computing Surveys*, 24:293–318, September 1992.

[BUT 91] K.M. Butler, D.E. Ross, R. Kapur, and M.R. Mercer. "Heuristics to Compute Variable Orderings for Efficient Manipulation of Ordered BDDs". in *Proceedings of the 28th Design Automation Conference, DAC'91*, June 1991. San Francisco, California.

[CHO 90] H. Cho, G. Hatchel, S.W. Jeong, B. Plessier, E. Swharz, and F. Somenzi. "ATPG Aspect of FSM Verification". in *Proceedings of IEEE International Conference on Computer Aided Design, ICCAD'90*, November 1990.

[COU 92a] O. COUDERT and J.-C. MADRE. "A New Method to Compute Prime and Essential Prime Implicants of Boolean Functions". in T. KNIGHT and J. SAVAGE, editors, *Advanced Research in VLSI and Parallel Systems*, pp 113–128, March 1992.

[COU 92b] O. COUDERT and J.-C. MADRE. "Implicit and Incremental Computation of Primes and Essential Primes of Boolean Functions". in *Proceedings of the 29th ACM/IEEE Design Automation Conference, DAC'92*, June 1992.

[COU 93] O. COUDERT and J.-C. MADRE. "Fault Tree Analysis: 10^{20} Prime Implicants and Beyond". in *Proceedings of the Annual Reliability and Maintainability Symposium, ARMS'93*, January 1993. Atlanta NC, USA.

[COU 94] O. COUDERT and J.-C. MADRE. "MetaPRIME: an Iteractive Fault Tree Analyser". *IEEE Transactions on Reliability*, 43(1):121–127, March 1994.

[DOY 96] S.A. DOYLE, J.B. DUGAN, and M. BOYD. "Combinatorial-Models and Coverage: A Binary Decision Diagram (BDD) Approach". in *Proceedings of the IEEE Annual Reliability and Maintainability Symposium, ARMS'95*, pp 82–89. IEEE, 1996.

[DUT 96a] Y. DUTUIT and A. RAUZY. "A Linear Time Algorithm to Find Modules of Fault Trees". *IEEE Transactions on Reliability*, 45(3):422–425, 1996.

[DUT 96b] Y. DUTUIT, A. RAUZY, and J.-P. SIGNORET. "Réséda: a Reliability Network Analyser". in C. CACCIABUE and I.A. PAPAZOGLOU, editors, *Proceedings of European Safety and Reliability Association Conference, ESREL'96*, volume 3, pp 1947–1952. Springer Verlag, 1996. ISBN 3-540-76051-2.

[FRI 90] S.J. FRIEDMAN and K.J. SUPOWIT. "Finding the Optimal Variable Ordering for Binary Decision Diagrams". *IEEE Transactions on Computers*, 39(5):710–713, May 1990.

[FUJ 88] M. FUJITA, H. FUJISAWA, and N. KAWATO. "Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagrams". in *Proceedings of IEEE International Conference on Computer Aided Design, ICCAD'88*, pp 2–5, 1988.

[FUJ 91] M. FUJITA, Y. MATSUNAGA, and N. KAKUDA. "On the Variable Ordering of Binary Decision Diagrams for the Application of Multilevel Logic Synthesis". in *Proceedings of European Conference on Design Automation, EDAC'91*, pp 50–54, 1991.

[FUJ 93] M. FUJITA, H. FUJISAWA, and Y. MATSUGANA. "Variable Ordering Algorithm for Ordered Binary Decision Diagrams and Their Evalutation". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(1):6–12, January 1993.

[FUS 72] J.B. FUSSEL and W.E. VESELY. "A New Methodology for Obtaining Cut Sets for Fault Trees". *Trans. Am. Nucl. Soc.*, 15:262–263, June 1972.

[ISH 91] N. ISHUIRA, H. SAWADA, and S. YAJIMA. "Minimization of Binary Decision Diagrams Based on Exchanges of Variables". in *Proceedings of the IEEE International Conference on Computer Aided Design, ICCAD'91*, pp 472–475, November 1991. Santa Clara CA, USA.

[LEE 85] W.S. LEE, D.L. GROSH, F.A. TILLMAN, and C.H. LIE. "Fault tree analysis, methods and applications : a review". *IEEE Transactions on Reliability*, 34:194–303, 1985.

[LIA 92] H.-T. LIAW and C.-S. LIN. "On the OBDD-Representation of General Boolean Functions". *IEEE Transactions on Computers*, 41(46):661–664, June 1992.

[LIM 91] N. LIMNIOS. *Arbres de défaillance.* Traité des Nouvelles Technologies. Hermes, 1991. in french.

[MAD 94a] J.-C. MADRE, O. COUDERT, and H. FRAÏSSÉ. "New Qualitative Analysis Strategies in METAPRIME". in *Proceedings of the Annual Reliability and Maintainability Symposium, ARMS'94*, pp 298–303, 1994. Annaheim, California.

[MAD 94b] J.-C. MADRE, O. COUDERT, H. FRAÏSSÉ, and M. BOUISSOU. "Application of a New Logically Complete ATMS to Digraph and Network-Connectivity Analysis". in *Proceedings of the Annual Reliability and Maintainability Symposium, ARMS'94*, pp 118–123, 1994. Annaheim, California.

[MAL 88] S. MALIK, A.R. WANG, R.K BRAYTON, and A. SANGIOVANNI-VINCENTELLI. "Logic Verification using Binary Decision Diagrams in Logic Synthesis Environment". in *Proceedings of the IEEE International Conference on Computer Aided Design, ICCAD'88*, pp 6–9, November 1988. Santa Clara CA, USA.

[MIN 90] S. MINATO, N. ISHIURA, and S. YAJIMA. "Shared Binary Decision Diagrams with Attributed Edges for Efficient Boolean Function Manipulation". in L.J.M CLAESEN, editor, *Proceedings of the 27th ACM/IEEE Design Automation Conference, DAC'90*, pp 52–57, June 1990.

[MIN 93] S. MINATO. "Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems". in *Proceedings of the 30th ACM/IEEE Design Automation Conference, DAC'93*, pp 272–277, 1993.

[MIN 94] S. MINATO. "Calcultation of Unate Cube Set Using Zero-Suppressed BDDs". in *Proceedings of the* 31*th ACM/IEEE Design Automation Conference, DAC'94*, pp 420–424, 1994.

[PUL 96] L.L. PULLUM and J.B. DUGAN. "Fault Tree Models for the Analysis of Complex Computer-Based Systems". in *Proceedings of the IEEE Annual Reliability and Maintainability Symposium*, pp 200–207, 1996.

[RAU 93] A. RAUZY. "New Algorithms for Fault Trees Analysis". *Reliability Engineering & System Safety*, 05(59):203–211, 1993.

[SCH 89] W.G. SCHNEEWEISS. *Boolean Functions with Engineering Applications and Computer Programs.* Springer Verlag, 1989. ISBN 3-540-18892-4.

[SHA 49] C.E. SHANNON. "The synthesis of two-terminal switching circuits". *Bell Syst. Tech. J.*, 28:59–98, 1949.

[SIN 95] R.M. SINNAMON and J.D. ANDREWS. "New Approaches to Evaluating Fault Trees". in *Proceedings of European Safety and Reliability Association Conference, ESREL'95*, pp 241–254, June 1995.

[SIN 96a] R.M. SINNAMON and J.D. ANDREWS. "Fault Trees Analysis and Binary Decision Diagrams". in *Proceedings of the Annual Reliability and Maintainability Symposium, ARMS'96*, pp 215–222, 1996.

[SIN 96b] R.M. SINNAMON and J.D. ANDREWS. "Quantitative Fault Tree Analysis

Using Binary Decision Diagrams". *Journal Européen des Systèmes Automatisés, RAIRO-APII-JESA*, 30:1051–1072, 1996. Special Issue on Binary Decision Diagrams.

[VAL 79] L.G. Valiant. "On the complexity of computing the permanent". *Theoretical Computer Science*, 8:189–201, 1979.

[VES 81] W.E. Vesely, F.F. Goldberg, N.H. Robert, and D.F. Haasl. "Fault Tree Handbook". TR NUREG 0492, U.S. Nuclear Regulatory Commission, 1981.

[WEG 88] I. Wegener. "On the Complexity of Branching Programs and Decision Trees for Clique Functions". *J. ACM*, 35(2):461–471, 1988.