

Application of a New Logically Complete ATMS to Digraph and Network Connectivity Analysis

Olivier Coudert • Digital Paris Research Laboratory • Rueil Malmaison France
Jean Christophe Madre • Digital Paris Research Laboratory • Rueil Malmaison France
Henri Fraisse • Bull Research Center • Les Clayes sous Bois France
Marc Bouissou • Direction des Etudes et Recherches EDF • Clamart France

Key Words: Fault Tree Analysis, Digraph Analysis, Network Connectivity Analysis,
Assumption-based Truth Maintenance System

SUMMARY & CONCLUSIONS

The binary decision diagram (BDD) and the metaproduct technologies have made possible to perform interactively exact qualitative and quantitative analysis of fault trees that could not previously be exactly analysed. This paper shows that these technologies can also be used to build an interactive tool to perform exact analysis of digraphs and of network connectivity problems. The paper also shows that the basic functionalities needed to perform these analysis are exactly those of an assumption-based truth maintenance system (ATMS), and it describes a new logically complete ATMS built using BDDs and metaproducts. The usefulness and the efficiency of this approach is demonstrated through the analysis of the feedwater system of a nuclear plant.

1 INTRODUCTION

The recently developed *metaproduct* technology, combined with the *binary decision diagram* (BDD) technology, has made possible the development of a new fault tree analysis tool called Metaprime that allows engineers to perform exact *qualitative* and *quantitative* analysis of very complex *fault trees* that could not be treated by any previously available analysis tool [6].

This paper shows that it is possible, using these technologies, to solve several other reliability analysis problems, including digraph analysis and network connectivity analysis, and that the basic functionalities that are necessary to perform these analysis are exactly those of an assumption-based truth maintenance system (ATMS) [7].

Digraphs are a fault propagation model very often used in conjunction with fault trees [10, 2, 8]. However, though both models are directed graphs, digraphs are much more difficult to analyse than fault trees, because of the cycles that can be found in them. Some techniques have been proposed to analyse digraphs, for instance digraph matrix

analysis [10] is a very efficient method to compute the sets of single component and of double component failures of the system under study. However, due to its combinatorial nature, this method cannot be used to compute all the failure sets (cut sets) of complex digraphs [10]. It has more recently been proposed to translate in an automated way digraphs into fault trees in order to use available fault tree analysis techniques [8]. However this method is very costly because it requires a minimal cut set computation for each cycle of the treated digraph, and a duplication of large parts of this digraph, thus producing a fault tree that can be much larger than the corresponding digraph.

The functionalities of an ATMS include those of a *theorem prover* and of a *diagnosis system*. For instance, the ATMS has to check whether propositional formulas are satisfiable or valid, which are both tasks of a theorem prover, and it also has to find minimal explanations of facts, which is the task of a diagnosis system. Different ATMSs have been implemented in the past [7], all of limited interest because they are not *logically complete* and so are difficult to use to solve real life problems. We have shown in [9] that it was possible, using BDDs, to implement a logically complete and efficient ATMS. However this ATMS was still of limited power because its diagnosis functionalities, that consist of computing *prime implicants* and *prime covers* of Boolean functions, were still implemented using a classical method.

The first part of the paper presents a much more powerful ATMS in which the diagnosis features are implemented using the metaproduct technology. Section 2 briefly presents BDDs and metaproducts, and Section 3 gives the functional specifications of the ATMS and briefly describes its implementation. The second part of the paper presents two applications of this new ATMS. Section 4 is dedicated to digraph analysis, it shows that the ATMS can be used to perform exact qualitative and quantitative analysis of digraphs without transforming them into fault trees. Section 5 is dedicated to network connectivity analysis.

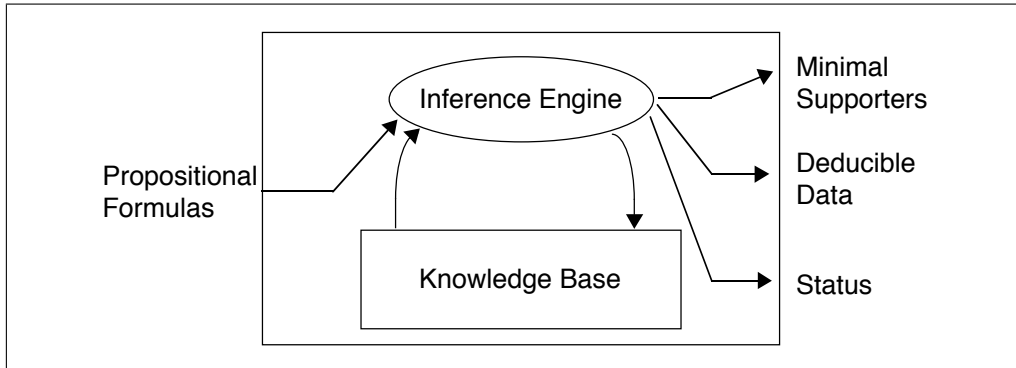


Figure 1: An ATMS.

2 BDDS AND METAPRODUCTS

This section presents the binary decision diagram (BDDs) and the metaproduct representations that are two graph based representations with very similar characteristics though they were developed to solve different problems. The reader is invited to refer to [3, 4, 5] for more detailed descriptions of these representations. We assume the reader to be familiar with Propositional Logic.

Binary decision diagrams are a representation of formulas built using a finite set of propositional variables $\{x_1, \dots, x_n\}$, and of subsets of the set $\{0, 1\}^n$ [3]. This representation is canonical, which means that for a given ordering of the variables x_1, \dots, x_n , two formulas are equivalent if and only if (iff) their BDDs are identical. In addition to this remarkable property that makes quite easy to implement a propositional theorem prover, the most interesting property of BDDs is the complete independence of the size of a BDD, i.e. its number of nodes, and of the size of the formula or of the subset of $\{0, 1\}^n$ it represents, which makes possible to manipulate these formulas and these subsets with costs not related to their sizes but to the sizes of their BDDs.

The metaproduct representation is a canonical representation of sets of products built from the set $\{x_1, \dots, x_n\}$, a product being a conjunction of positive literals (noted x_k) and negative literals (noted \bar{x}_k) [4]. Because there is a complete independence of the size of a metaproduct and of the number of products in the set it represents [4], metaproducts make possible to perform operations on sets of products with costs related to their sizes and not to the number of manipulated products. For instance the prime implicant computation algorithm presented in [4, 5] has a cost independent of the number of prime implicants to be computed, which allows Metaprime to analyse exhaustively in very short times fault trees with millions of prime implicants [6].

3 THE ATMS

This section gives the functional specifications of the

assumption-based truth maintenance system (ATMS) and presents a logically complete implementation of this ATMS based on BDDs and metaproducts.

3.1 FUNCTIONAL SPECIFICATIONS

The ATMS takes as inputs propositional formulas. We note by B the set of propositional formulas that the ATMS maintains. Since from an inconsistent set of formulas it is possible to deduce any formula, the first task of the ATMS is to keep its knowledge base *consistent*, which means that the set of formulas in B must always be satisfiable, i.e., have at least one interpretation.

The second task of the ATMS is to make proofs. The ATMS is given a set $S = \{l_1, \dots, l_m\}$ of literals, and another literal d , and is asked whether its knowledge base $B = \{f_1, \dots, f_p\}$ and S support d , which means that d is a logical consequence of the formulas in B and S . The problem here thus is to determine whether the formula $(f_1 \wedge \dots \wedge f_p \wedge l_1 \wedge \dots \wedge l_m) \Rightarrow d$ is a *tautology*. In order for this proof to make sense, the ATMS must check that the set of literals S is consistent and that it is also consistent with B . An associated and complementary task that the ATMS can be asked to perform is to compute all the literals that B and S support.

Building supporters and minimal supporters is a task that is the converse of the one described above and can be considered as a diagnosis task. The ATMS is now given a literal d and a set of variables S , and it is asked to compute the set of *supporters* of d that are all the sets of literals built from the variables in S that are consistent with B , and that support the literal d . A complementary and more difficult problem consists in computing all the *minimal supporters* of d , that are the supporters that are made of a minimal number of literals.

Finally the ATMS can be asked to evaluate the probability associated with a set P of supporters or of minimal supporters of a literal d . Each variable x_i occurring in these supporters is assigned a probability $\Pr(x_i)$, and the ATMS must compute the probability associated with the

subset of elements of $\{0, 1\}^n$ that are interpretations of these products. This set is represented by the disjunction $f = \bigvee_{p \in P} p$ and its associated probability $\Pr(P) = \Pr(f = 1) = \sum_{v \in \text{Sat}(f)} \prod_{k=1}^n \Pr(x_k = v_k)$, where $\text{Sat}(f)$ is the set of interpretations of f , $\Pr(x_k = 1) = \Pr(x_k)$, and $\Pr(x_k = 0) = 1 - \Pr(x_k)$.

3.2 IMPLEMENTATION

The use of BDDs to represent propositional formulas makes easy to keep the knowledge base of the ATMS consistent. Each time the ATMS is given a formula f , it first computes its BDD, which is different from the constant 0 if and only if f is consistent. The ATMS then computes the BDD B of the conjunction of all the formulas that are in its base and of f and checks that this BDD is different from 0, which guarantees that introducing f in the knowledge base does not make it become inconsistent. In order to reduce the cost of this operation, the BDD B of this conjunction is kept in memory for future use.

In order to determine whether the literal d is a logical consequence of the consistent base of formulas $B = \{f_1, \dots, f_p\}$ and the set of literals $S = \{l_1, \dots, l_m\}$, the ATMS first computes the BDD S of the conjunction of the literals in S , and checks that this BDD is not the constant 0. The ATMS then computes the BDD $(B \wedge S)$ which must also be different from 0. Making the proof finally comes down to checking that the formula $((B \wedge S) \Rightarrow d)$ is a tautology, which is done by building its BDD and testing whether it is the constant 1.

The supporter and minimal supporter computation algorithms used in the ATMS are based on the manipulation of metaproducts. Given a subset S of the variables $\{x_1, \dots, x_n\}$ and a literal d , the supporters of d are all the products p made of literals built using some of the variables in S , and that are such that the formula $f_1 \wedge \dots \wedge f_p \wedge p$ is satisfiable and the formula $(f_1 \wedge \dots \wedge f_p \wedge p) \Rightarrow d$ is a tautology. The metaproduct of this set of products can be computed directly from this definition using the algorithms presented in [4], as well as the metaproduct of the set of minimal supporters of d .

Finally, to compute the probability associated with the set P of supporters or of minimal supporters of a literal, the ATMS first computes, from the metaproduct of P , the BDD f of the set of interpretations of the products of P [4], and then it makes use of the probability evaluation algorithm presented in [6].

4 DIGRAPH ANALYSIS

The digraph model considered here is a more general form of the model commonly used in reliability analysis, because

it allows the description of more complex fault propagation rules than this model. We consider a digraph to be a directed graph made of a finite set of vertices V and a finite set of edges E . The vertices are associated with the components of the system under study, and there exists an edge e_{ij} from the vertex v_i to the vertex v_j iff the failure of the component C_i associated with v_i has some influence on the failure of the component associated with v_j .

4.1 LOGICAL REPRESENTATION

A component can fail for two reasons, either it can fail by itself, or it can fail because some other components have failed. In order to take these two reasons into account, we associate with each vertex v_j two propositional variables s_j and d_j , and a propositional formula f_j . The variable d_j models the failure of the component C_j , it is thus equal to 1 iff this component is out of order. The variable s_j is equal to 1 iff this component has failed by itself. The formula f_j , which describes how C_j can fail because of the failures of some other components, is built from the failure variables d_i of the vertices v_i such that there exists an edge e_{ij} .

From these definitions, it is immediate to build the knowledge base B that will be used by the ATMS to reason about the digraph, and that is made of the formulas $(d_j \Leftrightarrow (s_j \vee f_j))$. Note that in the commonly used digraph model, the formula f_i can be built with the connectors AND and OR only, which means that it is only possible to build *coherent* digraphs. Note also that in the commonly used graphical representation of the digraphs, the OR gates are not drawn, several edges pointing to a vertex being implicitly considered as the inputs of an OR gate.

The qualitative analysis of a digraph consists in determining under which conditions the components of the system are in the failure state. The most simple analysis task that the ATMS can perform is to simulate the propagation of the failures of some components. The ATMS is given a subset S of the variables s_j that represents a set of components that have failed by themselves, and it is asked either to determine whether these failures cause the failure of the component C_i , which is done by determining whether S is a support of the literal d_i , or to compute the complete set of components that are in the failure state, which is done by computing the set of literals d_i that are logical consequences of B and S .

A more precise analysis of the digraph can be done by computing the *failure sets* or *minimal failure sets* of a component C_j of the system. A failure set for C_j is a set of components, which, when failing by themselves, cause the failure of C_j , and a minimal failure set is a failure set of minimal size. The (minimal) failure set computation problem thus consists in computing all the (minimal) sets of positive literals built from the variables s_i that support the literal d_j , and thus are all the (minimal) supporters of d_j built from the literals s_i .



Figure 2: Digraph of Feedwater System.

The quantitative aspect of the digraph analysis consists, given the probabilities that the components of the system fail by themselves, in computing the probabilities of failure of some components the user is interested in. This is done by first computing the failure sets of this component and then by using the probabilistic evaluation procedure described in Section 2.

4.2 THE FEEDWATER SYSTEM

The ATMS has been used to analyse the reliability of the auxiliary feedwater system of a French 1300MW nuclear power plant. The task of this system is to supply water to at least one of the 4 steam generators of the plant, and of course this system must work in various incidental and

accidental situations. There are several loops in the system, for instance the pumps are cooled using a small part of their output flow, and the turbines of the pumps receive their steam from the steam generators. The designer is interested here in understanding under which conditions the flows in the system can become incorrect. Figure 2 shows the digraph that models this problem, drawn using the Figaro workbench [2]. It is made of 73 vertices, thus the knowledge base to reason about is made of 146 propositional variables and 73 formulas. The time needed to compute the BDD of this knowledge base on a DEC 3000-400 is 1.6s, and it is made of 5249 nodes.

We start the analysis of this digraph by first making some simulations. We assume that the components VD01, VD02, VD121, VD122, VD123 and VD124 have failed by themselves, and we determine whether these failures cause the

failure of the system. This is proved to be true in 0.1s, by checking that the knowledge base and this set of literals support the literal FAIL. We then determine whether all these failures are really necessary to cause the failure of the system, by computing the set of minimal failure sets that are contained in the set of components given above. The ATMS computes these minimal failure sets in 0.2s, there are 4 of them, the set {VD01, VD02} being of order 2, the sets {VD02, VD121, VD122} and {VD01, VD123, VD124} of order 3, and finally {VD121, VD122, VD123, VD124} of order 4. The complete set of minimal failure sets of the vertex FAIL is computed using the ATMS in 0.4s. There are 665857 minimal failures sets, and the metaproduct that represents them has 2719 vertices. The distribution of these minimal failure sets with respect to their size is shown in Figure 3.

An attempt has been made to manually break the cycles in this digraph in order to transform it into a fault tree. The resulting fault tree is completely analysed in 4.5s using Metaprime [6], it has 367237 minimal cut sets whose distribution with respect to their size is given in Figure 3. It is quite understandable that the number of minimal failure sets of this tree is different from the number of minimal failure sets of the digraph because some edges of the digraph have been deleted, thus breaking some logical links between the components' failures. A rough analysis of the tree would conclude that the tree and the digraph are equivalent, differences beginning to appear only with minimal failure sets of size 4, but the tree is not a correct model of the problem to be studied. Note that the reachability analysis technique presented in [10] would produce only the 2 smallest minimal failure sets of the digraph, which represent a very small part of all these minimal failure sets.

5 NETWORK CONNECTIVITY ANALYSIS

This section addresses the problem of network connectivity analysis [1]. A network is an oriented graph (V, E) , where V is a set of *vertices*, and E is a set of *edges*, i.e., a subset of $V \times V$. Among the vertices of the network, we distinguish a vertex called the *emitter*. The emitter sends a signal that is propagated in the network along the edges. An edge can be either working or out of work, in which case it is unable to transmit the signal. A *connecting path* from the emitter to a vertex of the network is a set of edges linking these two vertices that is made of working edges. For instance the set of edges $\{e_1, e_6, e_{11}, e_{16}\}$ of the network in Figure 4 is a path connecting the emitter v_1 to v_9 .

The connectivity analysis of the network consists in determining under which conditions the different vertices of the network receive the signal from the emitter. This problem has qualitative and quantitative aspects. The first aspect of the qualitative analysis consists in determining all the

Figure 4: Example of Network.

connecting paths from the emitter to a vertex, or the minimal ones. The second aspect of the qualitative analysis, which is complementary to the preceding one, consists in determining all the combinations of failure of the edges that can prevent this vertex from receiving the signal from the emitter, and more interestingly, all the minimal ones. The quantitative aspect of the problem consists, given failure probabilities associated with the edges of the network, in computing the probability that the transmitted signal does not reach this vertex.

5.1 LOGICAL REPRESENTATION

We associate a Boolean variable v_i with each vertex of the network, and a Boolean variable e_k with each edge. The state of a vertex will be defined by the value of its associated Boolean variable. A vertex v_i will not emit the signal iff $v_i = 0$. In the same way, an edge e_k will be working iff $e_k = 1$. The equation chosen to express that the edge e_k connects the vertices v_i and v_j is $(e_k \Rightarrow (v_i \Rightarrow v_j))$.

Let V be a set of vertices $\{v_1, \dots, v_n\}$ constituting a total graph. We associate with each of the n edges of this total graph that connect the vertex v_i to all the n vertices of the graph, a variable e_{ij} , and we note $K(e, v)$ the following formula:

$$K(e, v) = \bigwedge_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}} (e_{ij} \Rightarrow (v_i \Rightarrow v_j))$$

We say that an interpretation of K defines a path from the vertex v_i to the vertex v_j iff there exists a sequence $e_{i,j_2}, e_{j_2,j_3}, \dots, e_{k,j}$, such that $e_{i,j_2} = e_{j_2,j_3} = \dots = e_{j_k,j} =$

Order	Digraph	Fault Tree
1	1	1
2	1	1
3	64	64
4	1670	1566
5	21200	17580
6	121545	81193
7	180856	70524
8	175360	73500
9	107736	69128
10	44608	29824
11	11520	16800
12	1296	7056

Figure 3: Distribution of Minimal Failure Sets of Digraph and Fault Tree.

1. The following theorem shows that it is possible to build from the formula K , a formula that represents all the paths from the vertex v_i to the vertex v_j .

Theorem 1 *The formula $P_{ij}(e)$ built from the variables e_{ij} which describes the set of paths going from the vertex v_i to the vertex v_j is equivalent to the formula:*

$$(\forall v_1 \forall v_2 \dots \forall v_n (K \Rightarrow (v_i \Rightarrow v_j))) \quad (1)$$

Proof. An assignment of the variables e_{ij} is an interpretation of the formula $P_{ij}(e)$ iff it defines a path from v_i to v_j . The length of this path is necessarily smaller than $n - 1$. Thus the formula $P_{ij}(e)$ is the disjunction of the formulas $L_{ij}^k(e)$, $k = 1, \dots, n - 1$ that express that there exists a path of length k from v_i to v_j :

$$P_{ij}(e) = \bigvee_{k=1}^{n-1} L_{ij}^k(e), \quad \text{where} \quad (2)$$

$$L_{ij}^k(e) = \bigwedge_{\{j_2, \dots, j_k\} \subseteq \{1, \dots, n\}} e_{i,j_2} \wedge e_{j_2,j_3} \wedge \dots \wedge e_{j_k,j} \quad (3)$$

By eliminating the quantified variables from equation 1 using the rewriting rule associated with the quantifier “ \forall ” [4], one can check that we obtain equation 2. \square

The knowledge base B that describes any network made of p vertices can be obtained from K by conjuncting it with formulas of the form $\neg e_{ij}$, for any vertices v_i and v_j such that there does not exist an edge from v_i to v_j .

5.2 A Sample Session

We will study here the network drawn in Figure 4. Its knowledge base is made of 17 formulas and 26 variables, its BDD is built in 0.1s, and it has 88 vertices. We first suppose that the edges $e_1, e_6, e_8, e_5, e_{10}, e_{14}, e_{17}, e_{15}, e_{16}, e_{11}$ are all functioning and we check whether the vertex v_9 is connected to the emitter v_1 . This is done in 0.1s. We then compute all the subpaths of this path that connect v_1 to v_9 . This computation, which is done in 0.1s, produces the 3 following minimal subpaths $\{e_1, e_6, e_{11}, e_{16}\}$, $\{e_1, e_6, e_{11}, e_{15}, e_{17}\}$, and $\{e_1, e_5, e_6, e_8, e_{10}, e_{14}, e_{17}\}$.

We then compute all the minimal paths from the vertex v_1 to the vertex v_9 . These paths are computed in 0.2s, and there are 20 such paths. There are 5 paths of length 4, 5 paths of length 5, 4 paths of length 6, 5 paths of length 7, and 1 path of length 8. For instance, the only minimal path of length 8 is $\{e_3, e_4, e_6, e_9, e_{10}, e_{11}, e_{15}, e_{17}\}$.

References

REFERENCES

- [1] R. Billinton, R. N. Allan, *Reliability Evaluation of Engineering Systems*, Plenum Press, New York, 1992.
- [2] M. Bouissou, *The FIGARO Dependability Evaluation Workbench in Use: Case Studies for Fault-Tolerant Computer Systems*, in Proc. of the *International Conference on Fault-Tolerant Computing Systems*, Toulouse, France, 1993.
- [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] O. Coudert, J. C. Madre, "A New Method to Compute Prime and Essential Prime Implicants of Boolean Functions", in *Advanced research in VLSI and Parallel Systems*, T. Knight and J. Savage Editors, The MIT Press, pp. 113–128, March 1992.
- [5] O. Coudert, J. C. Madre, "Implicit and Incremental Computation of Primes and Essential Primes of Boolean Functions", in Proc. of *29th DAC*, Anaheim CA, USA, June 1992.
- [6] O. Coudert, J. C. Madre, "Fault Tree Analysis: 10^{20} Prime Implicants and Beyond", in Proc. of the *ARMS*, pp. 240–245, Atlanta GA, USA, January 1993.
- [7] J. de Kleer, "An Assumption-based TMS". in *Artificial Intelligence*, N° 28, pp. 127–162, 1986.
- [8] D. L. Iverson, "Automatic Translation of Digraph to Fault-Tree Models", in Proc. of 1992 ARMS, pp. 354–362, 1992.
- [9] J. C. Madre, O. Coudert, "A Logically Complete Reasoning Maintenance System Based on a Logical Constraint Solver", in Proc. of the *International Joint Conference on Artificial Intelligence*, pp. 294–299, Sydney, Australia, August 1991.
- [10] I. J. Sacks, "Digraph Matrix Analysis", in *IEEE Trans. on Reliability*, Vol R-34, NO. 5, pp. 437–446, 1985.

BIOGRAPHIES

Dr. Olivier Coudert; Digital Paris Research Laboratory; 85 av. Victor Hugo, 92500 Rueil Malmaison, France. Olivier Coudert received his engineering degree from Ecole Centrale de Paris in 1987, and his Ph.D. in Computer Science from ENST, Paris, France, in 1991. After 4 years at Bull Research Center, he joined Digital in 1993. He has been mainly active in the areas of automated reasoning, logics, and finite state machine verification.

Dr. Jean Christophe Madre; Digital PRL. Jean Christophe Madre received his engineering degree from ENSIMAG,

Grenoble, France, in 1984, and his Ph.D. in Computer Science from ENST, in 1990. After 5 years at Bull Research Center, he joined Digital in 1993. His areas of interest include circuit design, formal verification of hardware, and automated reasoning.

Henri Fraise; Bull Corporate Research Center; Rue Jean Jaurès; 78340 Les Clayes-sous-bois, FRANCE. Henri Fraise received his engineering degree from ESE, Paris, France in 1991, and is a Ph.D. student at ENST, since 1992. He joined Bull in 1992, where he has been mainly active in the areas of automated reasoning and logics.

Marc Bouissou; Direction des Etudes et Recherches; 1, avenue du Général de Gaulle, 92141 Clamart, France. Marc Bouissou received his engineering degree from Ecole des Mines de Paris. He joined EDF in 1982, and has been active since in the field of reliability analysis.