A Linear-Time Algorithm to Find Modules of Fault Trees

Yves Dutuit
Université Bordeaux I, Talence
Antoine Rauzy
Université Bordeaux I, Talence

Key Words — **Fault tree, Module.**

Abstract — **A module of a fault tree is a subtree whose terminal events do not occur elsewhere in the tree. Modules, which are independent subtrees, can be used to reduce the computational cost of basic operations on fault trees, such as the computation of the probability of the root event or the computation of the minimal cut sets. This paper presents a linear time algorithm to detect modules of a fault tree, coherent or not, that is derived from the Tarjan algorithm to find strongly connected components of a graph. We show, on a benchmark of real fault trees, that our method detects modules of trees with several hundred gates and events within few milliseconds on a personal computer.**

1. INTRODUCTION

Acronyms[1]

BDD (Bryant) binary decision diagram
LTA/DR linear-time algorithm (Dutuit & Rauzy).

We assume that the reader is familiar with fault trees and their applications; see [9] for a review of them.

Analysis of large fault tree can be computationally expensive, despite recent works on the use of BDD [3], as proposed by Madre & Coudert [6] and one of the authors [11] (see also [7]). It is sometimes difficult to understand the physical importance of a large number of minimal cuts. A way to tackle both difficulties is to detect modules and to treat them separately. A module of a fault tree is a subtree whose terminal events do not occur elsewhere in the tree, *ie*, independent subtrees. Chatterjee [5] and Birnbaum & Esary [2] developed the properties of modules, and demonstrated their use in fault-tree analysis. Locks [10] expanded the concept to non-coherent fault trees and showed its effectiveness in obtaining cut sets.

Several methods have been proposed to modularize fault trees, *eg*, [4, 8, 12, 14]. These methods are based on rewriting the original formula into an equivalent one, which contains more modules. LTA/DR achieves a less ambitious goal since it detects only modules already existing in the fault tree. However, its efficiency makes it of a special interest as a subprogram for rewriting methods or to design heuristics for variable ordering in BDD [6].

[1]The singular & plural of an acronym are always spelled the same.

Rigorous complexity analysis of 'fault-tree analysis algorithms' becomes more important as the size of the fault tree increases. Fault trees with several hundred gates & events are not uncommon, due to the generalization of tools that automatically generate them. For such trees, differences of complexity (*eg*, linear *vs* quadratic complexity) are important in practice, at least for operations that are often repeated. General techniques to achieve a linear complexity are thus of a great interest. LTA/DR is derived from the Tarjan algorithm [13] to detect strongly connected components of a graph. We believe that this kind of graph technique can be useful to perform other operations on fault trees or reliability networks.

Section 2 recalls the basics about fault trees. Section 3 presents the algorithm. Section 4 reports the experimental results.

Notation

$\mathcal{V}$	$\{e_1, e_2, \ldots\}$
$\wedge$	and
$\vee$	or
$\neg$	not
U	set of nodes (or vertices)
E	set of edges: $E \subseteq U \times U$
r	root-event of the tree
g_i	internal event #i
e_i	terminal event #i.

Other, standard notation is given in "Information for Readers & Authors" at the rear of each issue.

Nomenclature (for Fault Trees)

- Boolean formula. 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 $\wedge$, $\vee$, $\neg$.
- Graph. A set of *nodes* (or *vertices*) U, together with a set of *edges* E (edges are pairs of nodes). This paper considers only *directed* graphs: the edges are ordered.
- Edges & Nodes. Let (u,v) be an edge. u is a *parent* node of v; v is a *child* node of u. A node without parent node is a *root* node. A node without child node is a *sink* node (or leaf).
- Path. A sequence of nodes $u_1,\ldots,u_k$ such that $(u_i, u_{i+1}) \in E$ for $i=1,\ldots,k\text{-}1$.
- Path length. The number of edges the path traverses.
- Reachable node. Node v is reachable from a node u if there exists a path from u to v.
- Acyclic graph. A graph is acyclic, if for every node u, u is not reachable from itself through a positive length path.
- Strongly-connected component. For graph (U,E), a component is strongly connected if there is a set of nodes $U' \subseteq U$ such that for all u, v in U' there exists a path from u to v and vice-versa.

- Link. An edge links the vertex v to the vertex w iff v takes w as input.
- Events(v). A set of events reachable from the vertex v; eg, on the tree in figure 1,

$$\text{Events}(g_2) = \{g_4, e_1, e_2, g_5, e_3, e_4\}.$$

- Module. A module is an internal event whose terminal events do not occur elsewhere in the fault tree. Formally, an event v is a module iff for any other event w,

either $w \in \text{Events}(v)$,

or $\text{Events}(v) \cap \text{Events}(w) = \emptyset$.

2. FAULT TREES

For the purpose of this paper, fault trees are essentially Boolean formulae. Figure 1 depicts a small fault tree that is used throughout this paper. Its Boolean formula is:

$$r = (((e_1 \wedge e_2) \vee (e_3 \wedge e_4)) \wedge ((e_3 \wedge e_4) \vee (e_5 \wedge e_6))) \vee e_7. \quad (1)$$

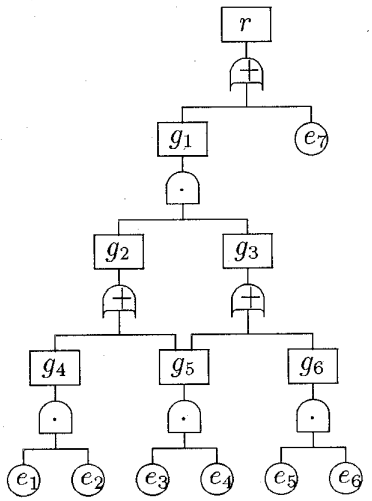

Figure 1. A Coherent Fault Tree

Assumptions

1. The physical system whose failures are described by such a fault tree, as well as each of its parts (elementary or not), are either good or failed.

2. Terminal events that describe failures of elementary components are mutually s-independent. ◄

Even small formulae, like (1), can be quite hard to read. This is the reason why one prefers, in general, to write fault trees as sets of Boolean equations. Moreover, equations reflect the graphical nature of fault trees. The set of equations describing r is:

$$\begin{aligned} r &= (g_1 \vee e_7) \\ g_1 &= (g_2 \wedge g_3) \\ g_2 &= (g_4 \vee g_5) \\ g_3 &= (g_5 \vee g_6) \\ g_4 &= (e_1 \wedge e_2) \\ g_5 &= (e_3 \wedge e_4) \\ g_6 &= (e_5 \wedge e_6). \end{aligned} \quad (2)$$

A fault tree can be considered as a directed acyclic graph whose nodes are either events or logical gates. From the definitions, the root event and the terminal events are always modules. In following sections, for the sake of conciseness, we do not mention them.

Modules of the tree in figure 1 are:

g_1, g_4, g_5, g_6.

3. DETECTING MODULES IN LINEAR TIME

3.1 Depth-First Left-Most Traversals

LTA/DR works "in the spirit" of the Tarjan algorithm [13] to compute strongly connected components of (directed) graphs in that it performs two depth-first left-most traversals of the formula under study, updating counters at each traversal.

A depth-first left-most traversal of the tree in figure 1 visits the events in the order reported in table 1.

Table 1. Depth-First Left-Most Traversal
For fault tree in figure 1
[S = step; V = visited node]

S	1	2	3	4	5	6	7	8
V	r	g_1	g_2	g_4	e_1	e_2	g_4	g_5
S	9	10	11	12	13	14	15	16
V	e_3	e_4	g_5	g_2	g_3	g_5	g_6	e_5
S	17	18	19	20	21	22		
V	e_6	g_6	g_3	g_1	e_7	r		

Notes (about table 1)

1. Each internal event is visited at least twice: the first time when descending from its first parent, the second time when going back from its right-most child.

2. The graph under a vertex is never traversed twice. For instance, the graph under g_5 is traversed when coming from g_2 but not the third time g_5 is encountered (*viz*, when coming from g_3). ◄

From notes #1 & #2, it follows that each edge (link between a node and one of its children) is traversed exactly twice; thus a depth-first left-most traversal of the graph is linear in the number of edges it contains.

3.2 Algorithm

Notation

v an internal event

t_i date of visit i of v in a depth-first left-most traversal of the graph; i=1,2.

The principle of the algorithm is: v is a module iff none of its descendants is visited before t_1 or after t_2 during the traversal. For instance,

- g_5 is a module because both e_3 & e_4 are visited after 8 (date of visit #1 to g_5) and before 11 (date of visit #2 to g_5);
- g_2 is not a module because g_5 is visited at 14, which is after 12 (date of visit #2 to g_2);
- g_3 is not a module because g_5 is visited at 8 which is before 13 (date of visit #1 to g_3. ◄

In order to implement the algorithm, 3 counters/node are needed, to contain respectively the dates of visits #1, #2. #last. The algorithm has 3 steps:

1. Initialize the counters: Traverse the list of nodes.

2. Perform the first 'depth-first left-most' traversal of the graph to set counters; for leaves, dates #1 & #2 are identical.

3. Perform the second 'depth-first left-most' traversal of the graph in which it collects, for each internal event v, the minimum of the first dates and the maximum of the last dates of its children. v is a module iff: a) the collected minimum is greater than the first date of v, AND b) the collected maximum is less than the second date of v. ◄

The second traversal has the same complexity as the first one. Thus LTA/DR is linear in the size (number-of-nodes + number-of-edges) of the tree.

For the example, the results are summarized in table 2.

Table 2. Results for the Tree of Figure 1

	r	g_1	g_2	g_3	g_4	g_5	g_6	e_1	e_2	e_3	e_4	e_5	e_6	e_7
#1	1	2	3	13	4	8	15	5	6	9	10	16	17	21
#2	22	20	12	19	7	11	18	5	6	9	10	16	17	21
last	22	20	12	19	7	14	18	5	6	9	10	16	17	21
min	2	3	4	8	5	9	16							
max	21	19	14	18	6	10	17							
mod	yes	yes	no	no	yes	yes	yes							

Connectives do not influence the algorithm that works consequently on any kind of formulae (including s-coherent and non s-coherent fault trees).

4. EXPERIMENTAL RESULTS AND CONCLUSION

The results in table 3 were obtained on a set of real fault trees from Dassault Aviation and Électricité de France. The rows give —

Row #1: numbers of internal events of the tested trees;
Row #2: numbers of terminal events;
Row #3: number of modules they contain;
Row #4: running times in milliseconds on a PC 486 66MHz (this computer does not provide a measure of time finer than 16ms, which explains the regularity of reported times).

Table 3. Experimental Results
[bs.ev = basic events
l.m = largest module
m.d = module detection
t.p - m.d = tree processing without module detection
t.p + m.d = tree processing with module detection]

	1	2	3	4	5	6	7	8
#gates	248	254	323	337	289	439	484	1050
bs.ev	283	278	276	306	311	530	548	362
#modules	25	25	70	32	32	30	29	70
#gates l.m	224	230	230	306	258	410	456	981
#bs.ev l.m	251	246	216	276	272	490	509	240
Running times (milliseconds) for:								
m.d	16	25	33	33	16	16	33	50
t.p - m.d	9016	11093	1038	95028	39045	1035	2053	69025
t.p + m.d	8067	10045	1006	88095	37014	1027	2047	44070

Table 3 shows that LTA/DR is very efficient. It is integrated satisfactorily in several tools of the Aralia toolbox [1], to design heuristics for variable ordering (for BDD), and as a part of formula simplifiers.

Table 3 reports running times to compute BDD encoding of fault trees with & without module detection. The running times are not very different because all of the trees, except the last one, are decomposed into a large module just under the root event and a number of modules that gather very few terminal events. Despite that, running times are always improved by module detection. The largest module of the last tree is appreciably smaller than the tree itself. As a consequence, module detection improves running times appreciably as well.

REFERENCES

[1] Groupe Aralia, "Computation of prime implicants of a fault tree within Aralia", *Proc. European Safety & Reliability Assoc. Conf*, ESREL'95, 1995 Jun, pp 190-202; ESREL.

[2] Z.W. Birnbaum, J.P. Esary, "Modules of coherent binary systems", *SIAM J. Applied Mathematics*, vol 13, 1965, pp 442-462.

[3] K. Brace, R. Rudell, R. Bryant, "Efficient implementation of a BDD package", *Proc. 27th ACM/IEEE Design Automation Conf*, 1990; IEEE.

[4] L. Camarinopoulos, J. Yllera, "An improved top-down algorithm combined with modularization as highly efficient method for fault tree analysis", *Reliability Engineering*, vol 11, 1985, pp 93-108.

[5] P. Chatterjee, "Modularization of fault trees: A method to reduce the cost of analysis", *Reliability and Fault Tree Analysis*, 1975, pp 101-137; SIAM.
[6] O. Coudert, J.C. Madre, "MetaPrime: An interactive fault tree analyzer", *IEEE Trans. Reliability*, vol 43, 1994 Mar, pp 121-127.
[7] S.A. Doyle, J.B. Dugan, M. Boyd, "Combinatorial-models and coverage: A binary decision diagram (BDD) approach", *Proc. Ann. Reliability and Maintainability Symp*, 1995, pp 82-89.
[8] T. Kohda, E.J. Henley, K. Inoue, "Finding modules in fault trees", *IEEE Trans. Reliability*, vol 38, 1989 Jun, pp 165-176.
[9] W.S. Lee, D.L. Grosh, F.A. Tillman, C.H. Lie, "Fault tree analysis, methods and applications: A review", *IEEE Trans. Reliability*, vol R-34, 1985 Aug, pp 194-203.
[10] M.O. Locks, "Modularizing, minimizing and interpreting the K&H fault tree", *IEEE Trans. Reliability*, vol R-30, 1981 Dec, pp 411-415.
[11] A. Rauzy, "New algorithms for fault tree analysis", *Reliability Engineering & System Safety*, vol 5, 1993, pp 203-211.
[12] A. Rosenthal, "Decomposition methods for fault tree analysis", *IEEE Trans. Reliability*, vol R-29, 1980 Jun, pp 136-138.
[13] R.E. Tarjan, "Depth first search and linear graph algorithms", *SIAM J. Comput.*, vol 1, 1972, pp 146-160.
[14] J. Yllera, "Modularization methods for evaluating fault trees of complex technical systems", *Engineering Risk and Hazard Assessment* (A. Kandel, E. Avni, *Eds*), vol 2, 1988, chapter 5; CRC Press.

AUTHORS

Pr. Yves Dutuit; Dept. Hygiène et Sécurité, LADS; IUTA; Université Bordeaux I; 33405 Talence cedex, FRANCE.
Internet (e-mail): dutuit@iuta.u-bordeaux.fr

Yves Dutuit was born in 1943 in Morocco. After having received his Doctorat-ès-Sciences in time-domain spectroscopy from Bordeaux University, he devoted himself to teaching safety & reliability at the Bordeaux Institute of Technology. His research interests lie in qualitative & quantitative assessments methods in reliability. He is on the editorial board of *Reliability Engineering & System Safety*, and is a referee for this *Transactions* and *Performance Evaluation Journal*.

Antoine Rauzy; LaBRI, Université Bordeaux I; 351, Cours de la Libération; 33405 Talence cedex, FRANCE.
Internet (e-mail): rauzy@labri.u-bordeaux.fr

Antoine Rauzy received his PhD (1989) in Computer Science from the University of Marseilles. He joined the Computer Science Laboratory of the University of Bordeaux in 1989 and the French National Center for Scientific Research in 1991. His research topics are constraint logic programming, automated deduction, validation of distributed system, and (since 1993) reliability analysis. His activity is focused on Boolean reasoning and he leads the French group RESSAC on practical resolution of NP-complete problems.

Manuscript received 1996 March 3

Publisher Item Identifier S 0018-9529(96)07355-1 ◄TR►

TUTORIAL PAPERS *Physics & Chemistry of Failure* *TUTORIAL PAPERS* *Physics & Chemistry of Failure* *TUTORIAL PAPERS*

You Can Help

TUTORIAL PAPERS *Physics & Chemistry of Failure* *TUTORIAL PAPERS* *Physics & Chemistry of Failure* *TUTORIAL PAPERS*

We are interested in receiving tutorial manuscripts on the specifics of failure mechanisms of materials and of fabricated parts. (These manuscripts will be refereed.)

The manuscript should discuss the physics, the chemistry, the metallurgy, *etc* of the way things fail. We are NOT inviting tutorials on probability, statistics, and/or other mathematical procedures & concepts.

It is important to designers, especially at the part (non-repairable) level, to know:

- What kind of "bad things" to watch out for and to avoid.
- How to eliminate or to mitigate the effects of those "bad things".

We want papers with both of the following attributes:

- Non-mathematical discussion of the concepts (Albert Einstein said that if you can't explain it without the math, then you don't really understand it). The purpose of this part is to help engineers know what questions to ask.
- Quantitative discussions of the failure mechanisms (their causes & results). Any quantitative estimates of effects should give some measure of the uncertainty involved. The purpose of this part is to help engineers make accurate, real predictions about reliability.

For the past several years we have been publishing one such paper in each issue; look at those to get an idea of what we are after. ◄TR►