Bypassing BDD Construction for Reliability Analysis

Poul Frederick Williams[a], Macha Nikolskaïa[b] and Antoine Rauzy[b]

[a]*Department of Information Technology, Technical University of Denmark, Lyngby, Denmark, e-mail* pfw@it.dtu.dk

[b]*Laboratoire Bordelais de Recherche en Informatique, CNRS/Université Bordeaux-1, Bordeaux, France, e-mail* {macha,rauzy}@LaBRI.U-Bordeaux.FR

Key words: Boolean formula. Data structures. Reliability analysis.

1 Introduction

In this note, we propose a Binary Expression Diagram (BED [1])-based algorithm to compute the *minimal p-cuts* of boolean reliability models such as fault trees. BED make it possible to bypass the Binary Decision Diagram (BDD [2]) construction, which is the main cost of fault tree assessment.

We consider boolean formulas built over a set of variables $X = \{x_1, \dots, x_n\}$, the two constants $0, 1 \in \mathbb{B}$ and the usual operators $+$ (or), . (and), $^-$ (not), $x \rightarrow y, z$ (if-then-else) etc. A *literal* is either a variable x or its negation $\bar{x}$. A *product* is a set of literals that does not contain a literal and its negation. A product is assimilated with the conjunction of its elements. A minterm over X is a product that contains either positively or negatively all variables of X.

Let f be a formula and π be a product that contains only positive literals. We denote by π^c_X the minterm obtained by adding to π the negative literals formed over all of the variables occurring in f but not in π. π is a *p-cut* of f if $\pi^c_X \models f$. It is minimal if there is no product $\delta \subset \pi$ such that $\delta^c_X \models f$. We denote by $\Pi(f)$ the set of the minimal p-cuts of the formula f and $\Pi_k(f)$ the set of the minimal p-cuts with less than k literals.

Two decomposition theorems [4] allow the design of algorithms to compute the ZBDD [6] that encodes $\Pi_k(f)$ from the BDD that encodes f. Indeed, computing the former requires computing the latter. However, only part of the BDD is used, since some of the products it encodes are useless to the computation of $\Pi_k(f)$ [5]. We show that BEDs make it possible to compute only relevant parts of the BDD, therefore avoiding a potential exponential blow-up.

The remainder of this note is organized as follows. The next section introduces minimal p-cuts. Section 3 briefly reviews the BED data structure. Section 4 introduces an extension to BEDs to facilitate fault tree analysis. Section 5 gives practical results, and finally, section 6 draws conclusions.

2 Minimal P-Cuts

Minimal p-cuts play a central role in the assessment of fault trees. Boolean formulas describe the potential failures of the system under study, variables represent component failures. Minimal p-cuts represent minimal sets of component failures that induce a failure of the whole system. This notion should be preferred to the classical notion of prime implicants that also captures the idea of minimal solutions [4,5]. Minimal p-cuts approximate prime implicants by considering only positive parts of implicants, and k-truncated minimal p-cuts restrict the result to those of size at most k. The latter is of practical importance in qualitative analysis of fault trees, as it identifies sets of component with high probability of simultaneous failure that would cause the entire system to fail. There do not exist efficient (i.e. polynomial) algorithms to compute short prime implicants [3], while such algorithms do exist for minimal p-cuts [5] and are illustrated here. These algorithms are based on the following theorems. The first one establishes that minterms with more than k positive literals are useless for computing $\Pi_k(f)$. The second theorem gives a recursive principle for computing $\Pi_k(f)$ from the Shannon decomposition of f.

Theorem 1 (Dutuit & Rauzy [5]) *Let f be a boolean formula over the set of variables X and k be a integer, then the following equality holds:*

$$\Pi_k(f) = \Pi_\infty(F \cap \mathit{minterms}_k^+(X)),$$

where $\mathit{minterms}_k^+(X)$ denotes the minterms built over X that contain less than k positive literals and f is viewed as the set of minterms that satisfy it.

Theorem 2 (Dutuit & Rauzy [4]) *Let $F = x.f_1 + \bar{x}.f_0$ be a boolean formula with f_1 and f_0 not depending on x. Then, $\Pi_k(f)$ can be obtained as the union of two sets $\Pi_k(f) = v.\Pi_1 \cup \Pi_0$ where $\Pi_0 = \Pi_k(f_0)$, $\Pi_1 = \Pi_{k-1}(f_1 + f_0) \setminus \Pi_0$, $v.P = \{v.\pi; \pi \in P\}$ and $\setminus$ denotes set difference.*

We will exploit this fact not only to compute $\Pi_k(f)$ incrementally, but to expand the formula f into an OBDD incrementally. This is possible using the BED data structure [1].

3 Boolean Expression Diagrams

Definition 3 *Let X be a set of boolean variables, and let OP be a set of binary boolean operators. A* BED *over X and OP is a labelled ordered DAG $B = \langle V, E, l \rangle$ where V is a set of vertices, $E \subset V \times V$ a set of edges, $l : V \to \mathbb{B} \cup OP \cup X$ is a labelling function satisfying:*

$$\begin{array}{ll} (i) \quad l(v) \in \mathbb{B} \Rightarrow \rho(v) = 0, & \\ & \textit{where } \rho : V \to \mathbb{N} \textit{ gives the arity of a vertex.} \\ (ii) \quad l(v) \notin \mathbb{B} \Rightarrow \rho(v) = 2, & \end{array}$$

Definition 4 *The denotation of a vertex of a BED over a set of variables X and a set of operators OP is a boolean function from $\vec{x} = x_1, \ldots x_{|X|} \to \mathbb{B}$ defined by $\mathcal{D}$ as follows.*

$$\begin{aligned} \mathcal{D}[\![0]\!] &= \lambda\vec{x}.0 \\ \mathcal{D}[\![1]\!] &= \lambda\vec{x}.1 \\ \mathcal{D}[\![\odot [v_1, v_0]]\!] &= \lambda\vec{x}(\mathcal{D}[\![v_1]\!] \ \odot \ \mathcal{D}[\![v_0]\!]), \ \ \forall \odot \in OP \\ \mathcal{D}[\![x[v_1, v_0]]\!] &= \lambda\vec{x}(x \to \mathcal{D}[\![v_1]\!], \mathcal{D}[\![v_0]\!]) \end{aligned}$$

The basic step of conversion of a BED into a BDD is the *up* transformation.

Definition 5 *The* up *transformation is defined by:*

$$\odot\ ((x \to f_1, f_2), (x \to f_1', f_2')) \overset{\text{up } x}{\Longrightarrow} x \to (\odot f_1, f_1'), (\odot f_2, f_2') \ .$$

If one of the argument vertices of $\odot$ is not labelled by x but by another variable y, then a new vertex labelled by x is created : $x[y, y]$. We call *up-one* the repeated application of *up* transformation to one variable x until it is moved up to the top of the BED structure. Simultaneous application of *up-one* to all variables $x_1, \ldots, x_n$ is called *up-all* and corresponds to the standard BDD construction.

4 Minimal P-Cuts with BEDs

It is clear that minimal p-cuts can be computed using BEDs, since it is sufficient to convert the BED for f to a BDD using *up-all*, then to apply the standard algorithm from [4]. The disadvantage is that we construct the entire BDD for the f, when only part of this information is necessary for computing the p-cuts (theorem 1). The *up* transformation gives us finer control over the conversion of the BED to an BDD. We will show that minimal p-cuts can be computed by a bottom-up expansion of the formula that only converts what is necessary for the computation. The resulting algorithm does strictly less, and often *significantly* less, work than the standard algorithm.

We extend the BED data structure with a new kind of unary operator node, `PC`, which marks the frontier between a boolean formula and its p-cuts. Nodes above this frontier represent the BDD encoding the p-cuts for the formula f as the disjunction of the minterms π^c_X, where π is a k-truncated minimal p-cut of f. In each step of the new algorithm, the *up* transformation lifts the smallest variable in an order L over any boolean operators or other variable nodes, until it reaches a `PC` operator. The extended *up* transformation that is based on theorem 2 is shown in the figure 1. `PC` has two attributes: an ordered list L of the variables occurring in the formula f under study and the truncation size k (noted between brackets).

$$\begin{array}{lcll}
\texttt{PC}(0)[k;L] & \overset{up\ x}{\Longrightarrow} & 0 & \\
\texttt{PC}(1)[k;\varepsilon] & \overset{up\ x}{\Longrightarrow} & 1 & \\
\texttt{PC}(1)[k;x.L] & \overset{up\ x}{\Longrightarrow} & \bar{x}.\texttt{PC}(1)[k;L] & \\
\texttt{PC}(x \rightarrow f,g)[0;x.L] & \overset{up\ x}{\Longrightarrow} & \bar{x}.\texttt{PC}(g)[0;L] & \\
\texttt{PC}(x \rightarrow f,g)[k;x.L] & \overset{up\ x}{\Longrightarrow} & x \rightarrow S,T & (k > 0) \\
 & & T = \texttt{PC}(f)[k;L] & \\
 & & S = \texttt{PC}(f+g)[k-1;L].\bar{T} & \\
\texttt{PC}(x \rightarrow f,g)[k;y.L] & \overset{up\ x}{\Longrightarrow} & \bar{y}.\texttt{PC}(x \rightarrow f,g)[k;L] & (y < x)
\end{array}$$

Fig. 1. P-cut computation using *up* transformation

To calculate the minimal truncated p-cuts we use either *up-all* (corresponding to standard algorithm) or *up-one*. Figure 2 shows how the `PC` operators “drive” the computation, pulling BED variables up to the frontier. The process is started by seeding a `PC` operator at the root of the original formula. As long as there are variable nodes below a `PC` operator, we pull them up one by one in the order L, until either no variables remain or the `PC` nodes in the frontier exhaust their capacity ($k = 0$).

Proposition 6 *The number of BDD nodes created to encode the k-truncated p-cuts is bounded by $O(n^k)$.*

The proof is clear because the number of minterms π^c_X, where π is a k-truncated p-cut, is equal to $\sum_{i=0}^{k} \binom{n}{i} = O(n^k)$.

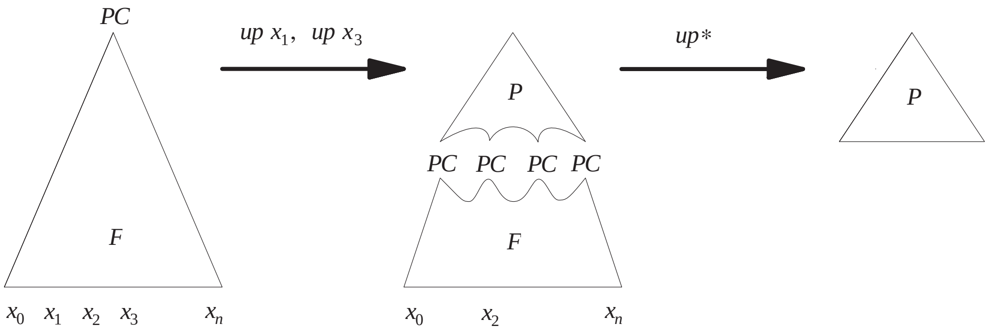

Fig. 2. Computation of p-cuts

5 Practical Results

We test our method experimentally on three fault trees, namely `cea9601`, `das9601`, and `wes9701`. They are from CEA (French Military), Dassault Aviation (French aviation company), and Westinghouse (American nuclear industry), respectively. All our experiments are run on a 500 MHz Digital Alpha.

Table 1 shows the number of p-cuts of order 1, 2, and 3 for the three fault trees as well as the runtimes in seconds to find a BDD representation for the p-cuts using *up-one*. For these calculations, the size of the BED data structure never exceeded 20 MB of memory.

	No. of p-cuts				Runtime [sec]			
Name	1	2	3	4	1	2	3	4
cea9601	0	0	1144	2024	2	27	122	683
das9601	0	47	80	446	1	2	11	75
wes9701	2	211	1079	54436	6	26	151	2000

Table 1
Number of p-cuts of order 1, 2, 3 and 4, and running times in seconds to compute them using *up-one* transformation

These results should be compared with the standard method (the *up-all* algorithm), which is unable to calculate the p-cuts for `cea9601` and `wes9701`. For `das9601` it succeeds in building the BDD for the fault tree in about 2 hours. The variable orderings used in the experiments are the ones given in the anonymous data files. The standard method depends on the variable ordering, and using improved heuristics to determine a good initial variable ordering will definitely improve the performance. However, the *up-one* method will also benefit from the use of an improved variable ordering heuristic.

6 Conclusion

In this note we proposed a new method to compute minimal truncated p-cuts. The method uses the Boolean Expression Diagram data structure instead of the standard Binary Decision Diagram data structure to represent fault trees. By including a new operator in the Boolean Expression Diagram data structure, it is possible to compute minimal trunctated p-cuts directly from the Boolean Expression Diagram without ever constructing the Binary Decision Diagram representation of the fault tree (that is often of gigabyte size).

We have shown experimental results for three industrial problems and compared them to the standard method. The results show that our method has an advantage over the Binary Decision Diagram methods.

Acknowledgements

Thanks to David James Sherman of Université Bordeaux-I for giving constructive criticism on drafts of this paper.

References

[1] H.R. Andersen and H. Hulgaard. Boolean Expression Diagrams. In *IEEE Symposium on Logic in Computer Science (LICS)*, July 1997.

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

[3] 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.

[4] Y. Dutuit and A. Rauzy. Exact and Truncated Computations of Prime Implicants of Coherent and non-Coherent Fault Trees within Aralia. *Reliability Engineering and System Safety*, 58:127–144, 1997.

[5] Y. Dutuit and A. Rauzy. Polynomial Approximations of Boolean Functions by Means of Positive Binary Decision Diagrams. In Lydersen, Hansen, and Sandtorv, editors, *Proceedings of European Safety and Reliability Association Conference, ESREL'98*, pages 1467–1472. Balkerna, Rotterdam, 1998. ISBN 90 54 10 966 1.

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