BDD BASED FAULT-TREE PROCESSING: A COMPARISON OF VARIABLE ORDERING HEURISTICS

M. Bouissou [1]   F. Bruyère [1]   A. Rauzy [2]

[1] Electricité de France (DER/ESF Section),
1, av. du Général de Gaulle, 92141 Clamart FRANCE
(tel) 33 1 47 65 55 07, (fax) 33 1 47 65 51 73,
(email) Marc.Bouissou@der.edfgdf.fr, Franck.Bruyere@der.edfgdf.fr

[2] LaBRI – CNRS – Université Bordeaux I,
351, cours de la Libération, 33405 Talence Cedex, FRANCE,
(tel) 33 5 56 84 60 83, (fax) 33 5 56 84 66 69,
(email) rauzy@labri.u-bordeaux.fr

Abstract

In this paper, we set up a method to compare and evaluate variable ordering heuristics for BDD based analyses of fault-trees, and give the results obtained with this method on 6 different heuristics, tried on a large benchmark of real-life fault-trees. Some of these heuristics were already proposed in the literature, some are original. As a final synthesis of all these results, we give the two strategies one can choose to process a new fault-tree, depending on the objective of this processing. This objective may be either to obtain a first BDD as soon as possible, or to minimize the size of the BDD, for further intensive use.

Keywords

Binary Decision Diagram (BDD), fault-tree, heuristic, optimization, benchmark.

1 Introduction

Binary Decision Diagrams (BDDs for short) are the state-of-the-art data structure to handle boolean functions [Bryant 92]. Since their introduction in the reliability field they have proved to be in many cases a very powerful tool. They made possible the assessment of complex fault-trees both qualitatively (computation of minimal cutsets) and quantitatively (exact calculation of the top event). Tools such as Aralia[1] can in many cases give more accurate results than conventional tools, while running 1000 times faster [Groupe Aralia 95].

Any boolean function, coherent or not, can be encoded by a BDD. This encoding requires to select a total order over the variables the function depends on. For a given ordering, the representation is unique, up to an isomorphism. The smaller the BDD, the more efficient subsequent operations using it (probability assessment, computation of minimal cutsets) are likely to be. So it is of a great interest to get BDDs as small as possible. From a theoretical point of view, this is hopeless: almost all the functions cannot admit polynomial size representations (w.r.t. their number of variables). This holds indeed for the particular case of BDDs whatever the chosen ordering is.

In many practical cases however, BDDs encoding fault-trees remain quite small, and their size heavily depends on the chosen ordering. Actually, there is often a variation of several orders of magnitude between the sizes of two BDDs built over reasonable variable orderings. Finding the best variable ordering is an intractable task. So, one uses heuristics to get good ones.

Several such heuristics have been proposed in the literature. However, they have been designed for combinational or sequential circuits, which concern boolean functions of a very different nature than those of boolean reliability models. It is not clear *a priori* whether they are also efficient for fault-trees.

The aim of this paper is to compare 6 interesting heuristics (three of them were proposed in the cited papers, the others are original), using Aralia. We also study the effect of restructuring formulae, according to the principles described in [Bouissou 96], before applying heuristics. We present a benchmark created for the purpose of this study. It is made of fault-trees from different origins presenting a wide range of specificities (size, structure, manual or automated trees, ...). We define an experimental protocol, based on Monte-Carlo simulations, that ensures a fair comparison. Finally we provide experimental results we got and we give some conclusions on advantages and drawbacks of the presented heuristics.

The rest of this paper is organized as follows. We recall basics on fault-trees and BDDs in the next section. We describe the heuristics we selected in section 3. Finally, we present the benchmark, the experimental protocol we used and the results we got in section 4.

2 Fault-Trees, Binary Decision Diagrams

Fault-Trees For the purpose of this paper, *fault-trees* are essentially considered as *boolean formulae*, i.e. terms inductively built over the two constants 0 and 1, a set of variables $\mathcal{X}$, and usual logical connectives $\wedge$ (and), $\vee$ (or), $\neg$ (not), k-out-of-n, ...

[1]Aralia is a boolean and stochastic model processing tool developed at the computer science lab. of the university of Bordeaux and financed by a pool of French companies including EdF.

Fault-trees are in general presented as sets of equations in the form $x = f$, where x is a variable and f is a formula. Sets of equations are a good representation of fault-trees because they reveal their actual structure, which is a directed acyclic graph and not a tree in the usual sense. In other words, a variable may have several fathers (also called fanouts) in the tree, which has numerous consequences from an algorithmic point of view.

Binary Decision Diagrams The BDD associated with a function is a compact encoding of the truth table of this function. This representation is based on the Shannon decomposition. Let f be a boolean function that depends on the variable x. Then, there exists two functions f_1 and f_0 not depending on x such that $f \equiv (x \wedge f_1) \vee (\neg x \wedge f_0)$. By choosing an ordering among the variables and applying recursively the Shannon decomposition, the truth table of any function can be graphically represented as a binary tree whose leaves are labeled with constants, whose internal nodes are labeled with variables and have outedges labeled with values assigned to variables. The value of the function for a particular variable assignment is obtained by descending along the corresponding branch of the tree. Such a representation is not compact at all since it has $2^n - 1$ internal nodes for a function of n variables. It is however possible to shrink it by means of the two following reduction rules.
– *Isomorphic subtrees merging.* Since two isomorphic subtrees encode the same function, at least one is superfluous.
– *Deletion of useless nodes.* A node encoding a formula of the form $(x \wedge f) \vee (\neg x \wedge f)$ is superfluous since it is equivalent to f.

By applying these two rules as far as possible, one gets the binary decision diagram associated with the formula. It is unique, up to an isomorphism. In practice, the size of a BDD heavily depends on the chosen variable ordering. Finding the best one is a computationaly intractable. The best known algorithms are in $\mathcal{O}(3^n)$, where n is the number of variables. So, one uses heuristics to get good ones. Several such heuristics have been proposed in the literature [Fujita *et al* 88, Malik *et al* 88, Berman 89, Minato *et al* 90, Butler *et al* 91, Fujita *et al* 93].

3 Heuristics for Variable Ordering

This section presents the 6 best heuristics among a dozen we tried. All of these heuristics are implemented in the Aralia toolbox [Rauzy 95]. We keep here the names they have in that report. The above heuristics share a number of common features. First, they are easy to compute (roughly linear in the size of the trees). Second, except for H1, they perform some reordering of gates' inputs. Third and more importantly, they respect modules. This is very important because, as shown in [Berman 89], any optimal ordering has this property.

Heuristic H1: This heuristic is probably the simplest one can imagine. It consists in ordering variables according to a depth-first left-most traversal of the formula considered as a directed acyclic graph. Consider for instance the following set of equations.

$$r = g_1 \wedge g_2 \qquad g_1 = a \vee g_3 \qquad g_2 = d \vee g_3 \vee e \qquad g_3 = b \wedge c$$

Variables are visited in the following order.

1	2	3	4	5	6	7	8	9	10
r	g_1	a	g_3	b	c	g_2	d	g_3	e

Which induces the ordering $a \prec b \prec c \prec d \prec e$. Note that the definition of g_3 is visited only once. It follows that this heuristic is of linear complexity w.r.t. the size of the formula. Despite its simplicity, this heuristic seems to be quite efficient with hand-written fault-trees.

Heuristic A3: This heuristic has been proposed in [Minato *et al* 90]. It works in three steps.
– First, it associates the weight 1 to each terminal variable and it propagates bottom-up these weights through the formula by associating to each intermediate variable the sum of the weights of the variables occurring in its definition.
– Second, it sorts arguments of connectives in increasing order of their weights.
– Third, it applies heuristic H1 on the resulting formula.

Consider for instance the above set of equations. First, the weight 1 is associated to a, b, c, d and e. Then, the weights of r, g_1, g_2 and g_3 are computed as follows.

$$\begin{array}{rclrcl} \omega(g_3) & = & \omega(b) + \omega(c) = 2 & \omega(g_2) & = & \omega(d) + \omega(g_3) + \omega(e) = 4 \\ \omega(g_1) & = & \omega(a) + \omega(g_3) = 3 & \omega(r) & = & \omega(g_1) + \omega(g_2) = 7 \end{array}$$

Second, it rewrites g_2 as $g_2 = d \vee e \vee g_3$ (definitions of r, g_1 and g_3 are kept unchanged). Finally, it perfoms a depth-first left-most traversal of the formula to get the ordering $a \prec b \prec c \prec d \prec e$. Since a list of length l can be sorted in $\mathcal{O}(l.log(l))$, this heuristic is in $\mathcal{O}(e.k.log(k))$, where e denotes the number of equations and k denotes the maximum, over the equations, of the number of arguments.

Heuristic H7: The principle of this heuristic is to sort the arguments of each connective according to their number of references (i.e. number of fanouts). The indexing resulting from a depth-first left-most traversal of the formula after this rewriting has been suggested by M. Fujita and H. Fujisawa and N. Kawato in [Fujita *et al* 88] (and also in [Fujita *et al* 93]).

Consider, for instance, the above set of equations. g_1 and g_2 have only one fanout, thus the definition of r remains unchanged. g_3 has two fanouts (g_1 and g_2) while a, d and e have only one, so g_1 is rewritten as $g_1 = g_3 \vee a$ and g_2 is rewritten as $g_2 = g_3 \vee d \vee e$. b and c have only one fanout. Thus the definition of g_3 remains unchanged.

After a depth-first left-most traversal of the formula, one gets the ordering $b \prec c \prec a \prec d \prec e$. As A3, this heuristic is in $\mathcal{O}(e.k.log(k))$.

Heuristic H4: This heuristic has been proposed by A. Rauzy (and is yet unpublished). It works by reordering arguments of connectives while performing a depth-first left-most traversal of the formula. More precisely, when it processes an equation in the form: $g = op(g_1, ..., g_n)$, where op is any connective, it first chooses the g_i $1 \leq i \leq n$ such that:
1) g_i contains the minimum number of leaves not already indexed. And, in case of tie:
2) the g_i whose sum of indices of already indexed leaves is minimum.

Then, it calls the algorithm on g_i, extracts g_i from the list $g_1, \ldots, g_n$ and continues with the rest of the list. When all of the g_i's are processed, the list of arguments of g is recomposed in the order in which the g_i's have been treated.

Consider, for instance, the above set of equations. g_1 has less leaves than g_2 (3 vs. 4); so it is processed first. a has less leaves than g_3; so it is processed first and it is put in the first position. Then g_3 is processed and we get the partial order $a \prec b \prec c$. g_1 is thus kept unchanged. Now g_2 is processed. Since g_3 is entirely processed, it is put before g_2. Thereafter it just remains to complete the partial order with $d \prec e$. We get $a \prec b \prec c \prec d \prec e$, and g_2 is rewritten as $g_2 = g_3 \vee d \vee e$.

It can be shown that this heuristic is in $\mathcal{O}(e.n.k.log(k))$, where n denotes the number of leaves and e and k keep the same meaning as above.

Heuristics H7(A3) and H7(H4): In most practical cases, the above heuristics are not deterministic, i.e. a large number of arbitrary choices still have to be made in sorting procedures because of ties. There are mainly two ways to deal with ties between gates' inputs: either one keeps them in the order they were given or one selects a new order at random. For the sake of repeatability, we adopted the first solution.

It was thus of interest to study how heuristics A3, H7 and H4 can be combined. For this purpose, we selected the two best combinations (of 2 heuristics): H7(A3), i.e. H7 applied on the result of A3, and H7(H4).

4 Benchmark, Experimental Protocol and Results

Benchmark In order to test the heuristics presented in the previous section, we selected 13 real life coherent fault-trees among those constituting the Aralia benchmark[2]. These trees are from different origins and present a wide range of specificities (size, structure, manual or automated trees, ...). For each tree we considered 3 versions: the initial one, plus two optimized versions (O1, O2) that were obtained with the optimizer described in [Bouissou 96].

Experimental Protocol A given heuristic may lead to significantly different results when applied to different rewritings of the fault-tree. By simply changing the orders of gates' inputs one may observe variations in the BDD sizes up to a factor 1500 (see Figure 1).

[2]The Aralia benchmark (including more detailed characteristics of the fault-trees than those given here) will be ftp/http available at the date of the conference, at `http://www.labri.fr`.

					#nodes			
name	#var	#gat	#min-cuts	fail%	min	max	mean	std.dev
baboab1	61	81	46,188	0%	1817	30121	5750.7	2212.37
baboab2	32	40	4,805	0%	145	1943	457.952	142.404
baboab3	80	107	24,386	0%	3305	27349	9926.87	3229.01
das9204	53	30	16,701	0%	53	153	79.4012	15.9311
das9208	103	145	8,060	0%	2614	20614	4833.58	1332.81
edf9r01	458	434	7,520,142	27.08%	551	289421	9437.65	22412.5
edfpa02	278	251	94.016	0%	7562	128847	53581.8	25209.2
edfpa09	196	142	497	0%	255	4727	1094.16	774.067
edfpr01	548	484	5,604,253	12.85%	8705	691766	185871	118286
edfrbd1	120	178	529,984	5.5%	145	34557	1325.65	2517.62
isp9602	116	122	5,197,617	0%	708	2993	1629.68	334.158
isp9603	91	95	3,431	0%	703	10950	4203.71	1783.94
isp9607	71	65	150,436	0%	149	805	409.923	64.4269

Table 1: Characteristics of fault-trees and some results

It is clear that there is no reason to write gates' fanins in any order rather than in another. This is why results found in some previous papers should be considered with care. In order to tackle this problem, we reorder gates' inputs at random, and we present the results obtained by the 6 heuristics on 500 random rewritings of each version (initial, O1, O2) of each tree.

Experimental Results The table 1 gives the characteristics of the trees as well as some results. The columns give the name, the number of variables (#var), the number of gates (#gat), the number of minimal cutsets (#min-cuts), the percentage of rewritings for which we did not succeed in computing the BDD within a limit of 10^6 nodes (fail.%), the minimum (min), the maximum (max), the mean (mean) and the standard deviation (std.dev) of the numbers of nodes of the computed BDDs. These statistics amalgamate, for each tree, the results obtained by the application of the 6 heuristics on 500 rewritings of each version of the tree.

More detailed results are given in Figure 1. Since the number of nodes differ very much from one tree to another, we "standardized" BDD sizes by defining a "relative size". The relative size of a BDD is the quotient of its actual number of nodes by the min given in Table 1. This measure makes it possible to amalgamate the results obtained for different trees. Since the minimum possible relative size is 1, it is interesting to note on how many trees this minimum is obtained (see Fig. 1).

The statistics (min, max, etc) are computed on the subset of successful trials. This makes direct comparisons uneasy, when the failure percentages differ. In contrast, the cumulative distribution functions of the relative sizes, shown as graphs in cells of Fig. 1,

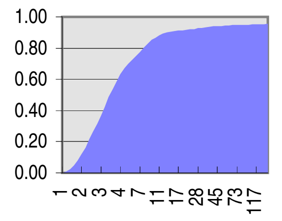
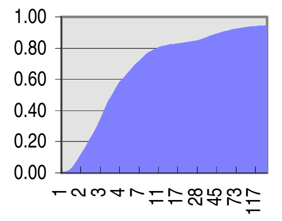
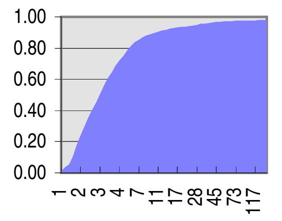
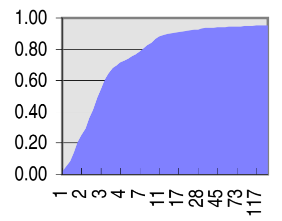
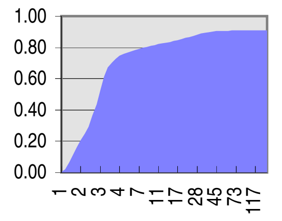
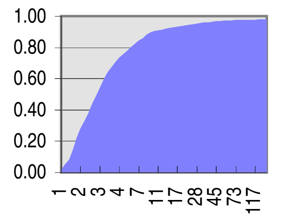
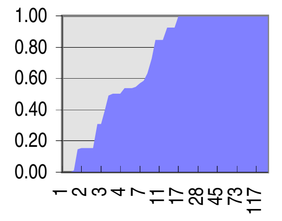
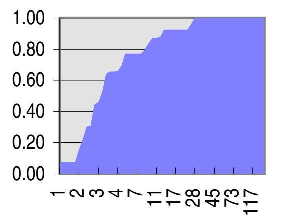
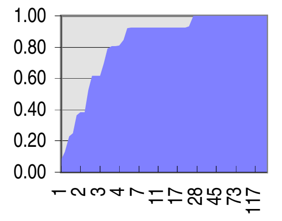
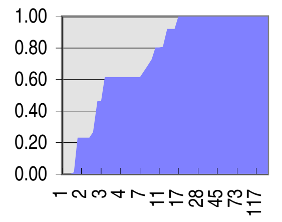
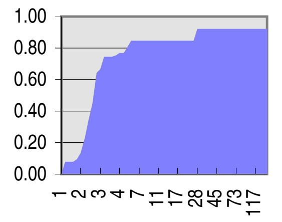
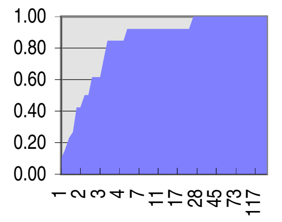
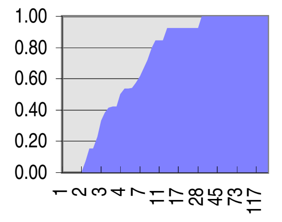
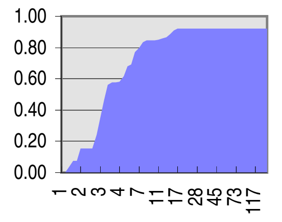
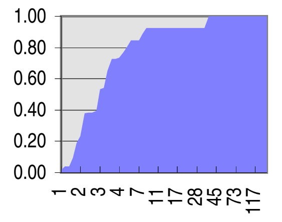
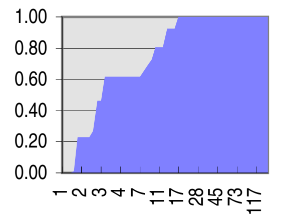
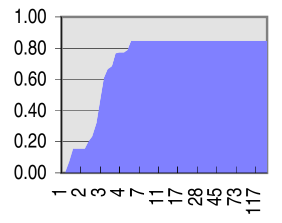
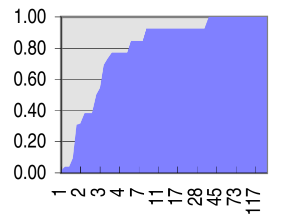
	initial tree		*optimized tree (O1)*		*optimized tree (O2)*	
h1	min 1 (1 tree) max 481,3 mean 8,19 st. dev. 25,4	 % failures : 3,8%	min 1 (1 tree) max 1559 mean 16,6 st. dev. 67,87	 % failures : 4%	min 1 (6 trees) max 425,69 mean 5,93 st. dev. 14,84	 % failures : 1,9%
h7	min 1 (3 trees) max 440,8 mean 7,39 st. dev. 23,2	 % failures : 4%	min 1,001 max 119,2 mean 5,39 st. dev. 9,19	 % failures : 9%	min 1 (6 trees) max 425,3 mean 5,78 st. dev. 14,73	 % failures : 1,9%
h4	min 1,34 max 19,5 mean 6,76 st. dev. 5,15	 % failures : 0%	min 1,02 max 28,9 mean 5,95 st. dev. 7,16	 % failures : 0%	min 1 (2 trees) max 26,69 mean 4,19 st. dev. 6,37	 % failures : 0%
h7h4	min 1,34 max 18,8 mean 6,15 st. dev. 5,49	 % failures : 0%	min 1,11 max 29,5 mean 4,78 st. dev. 7,54	 % failures : 7,7%	min 1 (2 trees) max 27,72 mean 4,22 st. dev. 6,89	 % failures : 0%
a3	min 1,83 max 33,7 mean 7,65 st. dev. 7,88	 % failures : 0%	min 1,30 max 20,31 mean 4,67 st. dev. 3,79	 % failures : 7,7%	min 1 (1 tree) max 39,42 mean 5,90 st. dev. 9,66	 % failures : 0%
h7a3	min 1,50 max 30,8 mean 7,63 st. dev. 7,99	 % failures : 7,6%	min 1,30 max 5,87 mean 2,91 st. dev. 1,17	 % failures : 15,4%	min 1 (1 tree) max 37,81 mean 5,72 st. dev. 9,47	 % failures : 0%

Figure 1: Each cell of this table contains statistics on the 13 (trees) x 500 (trials per tree) BDD relative sizes obtained for a given heuristic (row) and a given version of the trees

are directly comparable from cell to cell.

The results clearly show that there are 2 classes of heuristics. The first class, which includes H4, A3, and the combinations H7(H4), H4(A3), tends to give results with a very low standard deviation on a given tree (with a c.d.f. which looks like a step function), because the rule which determines the order of the sons of each gate leads to ties that, in most cases, are due to symmetrical sub-trees. Thus, the size of the BDD obtained by the application of the heuristic is not very sensitive to the initial random order.

The second class, including H1 and H7, clearly shows a very high dispersion of the obtained BDD sizes on a given tree, with an S-shaped, smooth c.d.f. from the smallest to the largest sizes.

In most cases H7 gives better results than H1, be it in terms of medians, means, maximums or minimums.

The heuristics of the first class always give results which are "somewhere in between" for initial trees (neither excellent, nor very bad), and good, or even close to optimal results for optimized trees. But in all cases, the exploration capabilities of H1 and H7 remain the best means to get the smallest BDD sizes.

The only problem with H1 and H7 is that they can also lead to extremely large BDDs (up to 1500 times larger than the smallest one !). But this pitfall can easily be avoided by the allocation of a maximum CPU time and of a maximum BDD size for each trial.

5 Conclusion

In this paper, we set up a method to compare and evaluate variable ordering heuristics for BDD based analyses of fault-trees, and gave the results obtained with this method on 6 different heuristics, tried on a large benchmark of real-life fault-trees. Some of these heuristics were already proposed in the literature, some are original. As a final synthesis of all these results, we now give the two strategies one can choose to process a new fault-tree, depending on the objective of this processing.

If the main objective is to obtain a first BDD as soon as possible, then the procedure (which, of course, can be abandoned as soon as a result is obtained) should be to try once the robust heuristics (H4, A3, H7(H4), H4(A3)) on the initial and then on optimized trees (O1, then O2).

If the main objective is to obtain a BDD as small as possible for further intensive use (once a first BDD has been obtained), one can afford to spend a lot of time in order to optimize the BDD. The best strategy should be to try H7 and H1 on random rewritings of the optimized O2 version of the tree (after each trial, the best already obtained result should be used to limit the maximum running time allowed for further computations). This process may, of course, be interrupted at any moment by the user.

References

[Groupe Aralia 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*, pages 190–202, Bournemouth – England, June 1995. European Safety and Reliability Association.

[Berman 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.

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

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

[Butler *et al* 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.

[Fujita *et al* 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*, pages 2–5, 1988.

[Fujita *et al* 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.

[Malik *et al* 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*, pages 6–9, November 1988. Santa Clara CA, USA.

[Minato *et al* 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*, pages 52–57, June 1990.

[Rauzy 95] A. Rauzy. Aralia version 1.0 : the Toolbox Manual. Technical report 1093-95, LaBRI – URA CNRS 1304 – Université Bordeaux-I, 1995.