Reliability Engineering and System Safety 186 (2019) 78–87

Contents lists available at ScienceDirect

Reliability Engineering and System Safety

journal homepage: www.elsevier.com/locate/ress

Model synthesis using boolean expression diagrams

Liu Yang*, Antoine Rauzy

Department of Mechanical and Industrial Engineering, Norwegian University of Science and Technology, Trondheim 7491, Norway

ARTICLE INFO

Keywords:
Fault tree synthesis
Boolean expression diagrams
Boolean formulas
System architecture

ABSTRACT

In this article, we propose a new method for fault tree analysis, called model synthesis, which comes in addition to traditional assessment techniques. It consists in rewriting the fault tree under study, or the set of minimal cutsets extracted from this fault tree, so to make some relevant information emerge.

Our implementation of model synthesis relies on encoding Boolean formulas by means of zero-suppressed Boolean expression diagrams. Rewriting heuristics are efficiently implemented by means of local operations on these diagrams. A key feature of zero-suppressed Boolean expression diagrams is that they make it possible to perform partial normalization of Boolean formulas, avoiding in this way the exponential blow-up of calculation resources most of the other methods suffer from.

We show how to take advantage of the architecture of the systems under study to guide rewriting heuristics. We illustrate the principles of model synthesis and of its implementation by means of examples.

1. Introduction

Fault tree analysis is one of the prominent techniques for safety and reliability analyses [1–3]. It is applied in a wide range of industries [4]. Fault trees are classically assessed in two ways: by means of qualitative analyses, which consists in identifying failure scenarios via the extraction of minimal cutsets, and by means of quantitative analyses, which consists in calculating probabilistic indicators such as the top event probability, importance factors or safety integrity levels.

In this article, we propose a new assessment method, called model synthesis, which comes in addition to classical assessment techniques. It aims at making some relevant information emerge out of the fault tree under study. The idea behind model synthesis is to rewrite the original fault tree or the minimal cutsets extracted from this fault tree, into an equivalent formula that hopefully sheds a new light on the system under study. We use thus here the term model synthesis in a quite different way as several other authors who focus mainly on the automatic construction of fault trees either from high level models [5] or from reliability graphs [6,7].

Model synthesis in our sense can be useful in different contexts. First, it can be used to deal with large fault trees, like the ones involved in probabilistic safety analyses of nuclear power plants. In this context, model synthesis can help to better understand which parts of the model are influencing the results the most, i.e. eventually to determine who are the main contributors to the risk. Second, model synthesis can be applied onto fault trees that are automatically generated from higher level descriptions, as nowadays routinely done in avionic industry [8]. Automatically generated fault trees tend to be very different from those an expert would design by hand. Model synthesis can be used in this case to create more amenable models in view of certifying the system under study. Model synthesis can finally be used in any context to create synthetic view of failure scenarios. One can for instance factorize minimal cutsets according to basic events related to similar components in order to study the impact of these components on the risk as a whole [9].

Technically, synthesizing a model means rewriting a Boolean formula so to obtain a more hierarchical, compact and hopefully informative representation. The method we propose here relies on zero-suppressed Boolean expression diagrams, a variant of Boolean expression diagrams [10] to implement this rewriting process. Compared to classical binary decision diagrams [11] or zero-suppressed binary decision diagrams [12], zero-suppressed Boolean expression diagrams make it possible to implement partial rewritings. Partial normalizations are of a great interest in our context as the information we are seeking for is in general concentrated onto a small fraction of the basic events. The rewriting process can therefore be applied on these basic events only, leaving the remaining of the model unchanged. We provide mathematical justification for such partial normalizations as well as the principles of an efficient implementation.

Rewriting procedures are essentially heuristics: they may or may not give interesting results depending on where and when they are applied. We show here that it is possible to use the architecture of the system

* Corresponding author.
E-mail address: liu.yang@ntnu.no (L. Yang).

https://doi.org/10.1016/j.ress.2019.02.019
Received 16 August 2018; Received in revised form 5 January 2019; Accepted 12 February 2019
Available online 14 February 2019
0951-8320/

under study as guideline for these heuristics. The idea is to keep related basic events close into the model. It ensures also that the fault tree stays consistent with the architecture so to make it understandable by the analyst [13].

The contribution of this article is eventually twofold. First, it proposes to complement traditional fault tree analysis with a new tool, model-synthesis. Second, it proposes an effective implementation of this new tool based on zero-suppressed Boolean expression diagrams.

The remainder of this article is organized as follows. Section 2 presents an illustrative example of what can be achieved by means of model synthesis. Section 3 introduces the zero-suppressed Boolean expression diagrams technology. Section 4 shows how implement model synthesis by means of partial operations on zero-suppressed Boolean expression diagrams. Section 5 reports the experimental results of the scalability test of the proposed method. Finally, Section 6 concludes this article.

2. Illustrative example

This section presents the case study we shall use throughout the article to illustrate ideas and techniques, namely the cooling system of a pressurized water nuclear reactor.

2.1. System description

The power of model synthesis stands primarily in its ability to make information emerges out of large models, typically such as those used to analyze the safety of cooling systems of nuclear power plants. Nevertheless, we keep here the description of the system as simple as possible, for pedagogical purposes.

Fig. 1 shows the principle of cooling systems of a pressurized water nuclear reactor. Nuclear reactions in the reactor vessel produce heat, heating the pressurized fluid in the primary coolant loop (in black on the figure). The hot primary coolant is pumped into the steam generator. Heat is transferred to a lower pressure secondary coolant loop (in dark grey on the figure) where the coolant evaporates into pressurized steam. The transfer of heat is accomplished without mixing the two fluids in order to prevent the secondary coolant from becoming radioactive. The pressurized steam is fed through a steam turbine which drives an electrical generator connected to the electric grid for transmission. After passing through the turbine the secondary coolant is cooled down and condensed in a condenser. The condenser converts the steam to a liquid so that it can be pumped back into the steam generator. The heat of the second loop is transferred to a third loop (in light grey on the figure) via the condenser. This third loop is usually made of a refrigerating tower and water which is typically pumped in a river.

We assume here that the cooling system is quadruplicate for the sake of production and safety, i.e. they are four independent circuits, each circuit consisting of the above three coolant loops. For technological reasons, these four circuits are however not fully independent as they share the same pressurizer.

To cool the nuclear reaction (and to produce electricity) the coolants must circulate into each of the three loops. This circulation is ensured by means of pumps which need to be powered. Normally, the pumps are powered by the power generated by the plant itself. In case the power of the plant does not suffice, they can be powered by the electric grid (off-site power). In case there is no off-site power, on-site diesel generators can be temporarily used to supply the required power.

2.2. Original model of the system

The fault tree for the cooling system is built classically top-down (see e.g. [1]), starting from the top-event LOCA (Loss of Coolant Accident). The cooling system is failed when all the four circuits are failed. A circuit is failed when at least one of its loops (primary, secondary and ternary) is failed. This process continues until the suitable level of decomposition. Table 1 gives the Boolean equations describing the original fault tree model we shall consider. Table 2 gives definitions of acronyms that are used in this fault tree.

Table 1
Fault tree describing the failures of the cooling system pictured Fig. 1.

$LOCA$	=	$LOCC_1 \cdot LOCC_2 \cdot LOCC_3 \cdot LOCC_4$
$LOCC_i$	=	$LOPC_i + LOSC_i + LOTC_i$
$LOPC_i$	=	$LOPP_i + FPR$
$LOSC_i$	=	$LOSP_i$
$LOTC_i$	=	$LOTP_i$
$LOPP_i$	=	$FPP_i + LOPS$
$LOSP_i$	=	$FSP_i + LOPS$
$LOTP_i$	=	$FTP_i + LOPS$
$LOPS$	=	$LOIP \cdot LOOP \cdot FDG$
$LOIP$	=	$LOPG_1 \cdot LOPG_2 \cdot LOPG_3 \cdot LOPG_4$
$LOPG_i$	=	$FCG_i + FTB_i + LOPC'_i + LOSC'_i$
$LOPC'_i$	=	$FPP_i + FPR$
$LOSC'_i$	=	FSP_i

2.3. Model synthesis

Model synthesis consists in rewriting the original model into equivalent formulas so to make information emerge. In other words, it makes it possible for the analyst to create different views on the model.

View 1: Role of the pressurizer and the power supply. Both the pressurized and the power supply have obviously a strong impact on the cooling system safety. An idea can thus be to rewrite the model (the set of equations given Table 1) so to visualize their role. Fig. 2 shows graphically the result of the factorization of the top event *LOCA* with respect to the two events *FPR* (failure of the pressurizer) and *LOPS* (loss of the power supply). This diagram can be interpreted as follows.

- If the pressurizer is failed (*FPR*), then the top event (*LOCA*) is realized.
- If the pressurizer is not failed but the power supply is lost (*LOPS*), then the top event is also realized.
- If neither the pressurizer is failed nor the power supply is lost, then the remaining scenarios to realize the top event are described by the formula $f_1 \cdot f_2 \cdot f_3 \cdot f_4$, where $f_i = FPP_i + FSP_i + FTP_i$, $1 \leq i \leq 4$.

Note that in this decomposition *FPR* is a basic event, but *LOPS* is an intermediate event. Note also that the choice and the order of these events depend on the needs of the analyst.

The model pictured Fig. 2 and the original model are equivalent. The former is indeed much more compact than the latter, but is probably not the one that the analyst would build upfront.

View 2: Role of primary pumps. The analyst may also want to study the role of primary pumps. The idea is thus to factorize the original model with respect to the *FPPi*'s events (failure of primary pump *i*, $1 \leq i \leq 4$). Fig. 3 shows a partial view of the result. It is possible to calculate an upper bound $\lceil p(\sigma) \rceil$ of the probability of a branch σ. For instance, any scenario under the upper branch of the diagram of Fig. 3, $\lceil p(FPP_1 \cdot FPP_2 \cdot FPP_3) \rceil = p(FPP_1) \times p(FPP_2) \times p(FPP_3)$. This upper bound may exceed a predefined threshold, meaning that the branch can be discarded. In our example, the probability of the simultaneous failure of three of the four primary pumps may be considered as too improbable to be reasonably considered.

The branch $FPP_1 \cdot FPP_2 \cdot \overline{FPP_3} \cdot \overline{FPP_4}$ involves only two failures of primary pumps. The analyst may want to check the formula corresponding to this branch. This formula may be however too large to be really informative. It can be nevertheless exploited, typically by extracting the basic events it involves, as shown on the figure. This is of interest in

Table 2
Definitions of acronyms.

LOCA	Loss of Coolant Accident	$LOCC_i$	Loss of Coolant Circuit i
$LOPC_i$	Loss of Primary Circuit i	$LOSC_i$	Loss of Secondary Circuit i
$LOTC_i$	Loss of Ternary Circuit i	$LOPP_i$	Loss of Primary Pump i
$LOSP_i$	Loss of Secondary Pump i	$LOTP_i$	Loss of Ternary Pump i
FPR	Failure of Pressurizer	FPP_i	Failure of Primary Pump i
FSP_i	Failure of Secondary Pump i	FTP_i	Failure of Ternary Pump i
LOPS	Loss of Power Supply	*LOIP*	Loss of Inside (on-site) Power
LOOP	Loss of Off-site Power	*FDG*	Failure of Diesel Generator
$LOPG_i$	Loss of Power Generation from circuit i	FCG_i	Failure of Current Generator i
FTB_i	Failure of Turbine i		

order to validate the model. In our case, the analyst can verify that no basic event issued from the first and second circuits shows up in this branch.

Note that the branches of the decision tree pictured Fig. 3 can be expanded on demand. Some branches may be unfolded further, while some other may be kept folded, typically because their exploration is made less relevant, thanks to symmetry arguments.

Table 3 summarizes the size of the original fault tree (using the number of basic and intermediate events), the size of ZBEDs (using the number of nodes) and the running time of getting the normalized ZBED. As comparison, *View 0* gives the result of a full normalization with an order of variables corresponding to a depth-first left-most traversal of the model.

2.4. Discussion

In the previous section, we showed two examples of extraction of useful information from the model. The extraction process requires to rewrite the original model, typically by factorizing some of its basic and intermediate events. Several factorizations are possible, providing different views on the model.

Performing such rewritings by hand would be tedious and error prone. We shall show in the next sections that Zero-Suppressed Expression Diagrams provide a suitable algorithmic framework to implement them in a simple and efficient way.

3. Zero-suppressed boolean expression diagrams

3.1. Definition

Zero-suppressed Boolean expression diagrams result of ideas stemmed from Minato's zero-suppressed binary decision diagrams [12] and Andersen's Boolean expression diagrams [10]. A zero-suppressed Boolean expression diagram (ZBED) is a directed acyclic graph with three types of nodes:

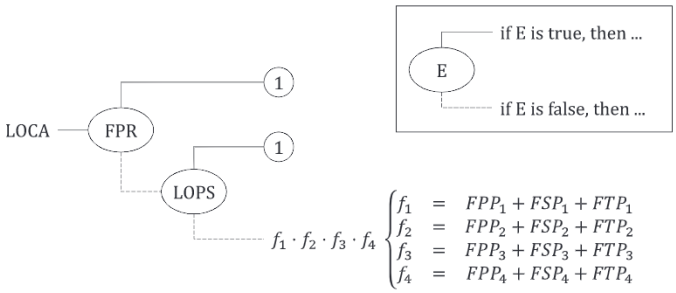

Fig. 2. View of the model obtained by factorizing events *FPR* and *LOPS*.

- Constant nodes: leaves labeled with Boolean constants (1 and 0).
- Variable nodes: leaves labeled with basic events.
- Operator nodes: internal nodes with three out-edges. Such a node is denoted by $t = \langle u, v, w \rangle$, where u is the node pointed by the first out-edge, called the "if-edge", v is the node pointed by the second out-edge, called the "then-edge", and w is the node pointed by the third out-edge, called the "else-edge".

Each node of a ZBED is interpreted as a Boolean formula as follows.

- A constant node is interpreted by the constant it is labeled with.
- A variable node is interpreted by the basic event it is labeled with.
- An internal node $t = \langle u, v, w \rangle$ is interpreted by the formula $f \cdot g + h$, where f, g and h are the respective interpretations of nodes u, v and w.

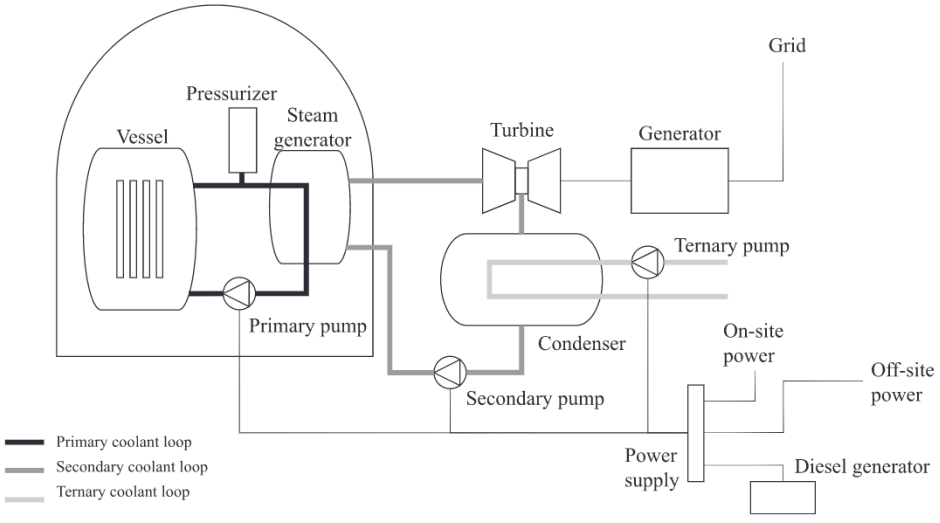

Fig. 1. A cooling system of the pressurized water nuclear reactor.

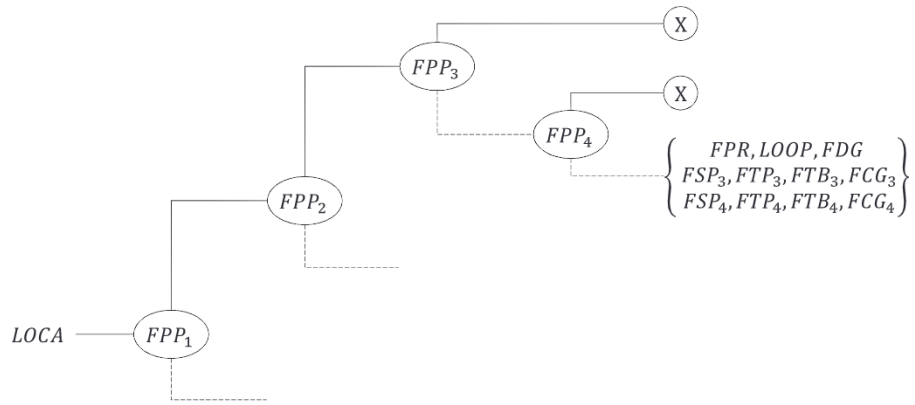

Fig. 3. View of the model obtained by factorizing events *FPPi*'s.

Table 3
Partial normalization comparing with a full normalization with an order of variables obtained from the depth-first-left-most traversal of the architecture in Fig. 16.

	# of fault tree events		# of ZBED nodes			Running time (s)	
	Basic	Intermediate	Original	Partial	Full	Partial	Full
View 0	23	39	81	–	215	–	0.657893
View 1	23	39	51	29	76	0.0024375	0.0207086
View 2	23	39	81	253	229	0.1364435	0.8062672

Table 4
Interpretation of the ZBED of Fig. 4 in terms of Boolean formulas.

Node	Definition	Interpretation	Formula
t_1	$\langle t_2, 1, t_7\rangle$	$t_2 \cdot 1 + t_7$	$t_2 + t_7$
t_2	$\langle t_3, 1, t_4\rangle$	$t_3 \cdot 1 + t_4$	$t_3 + t_4$
t_3	$\langle A, t_5, 0\rangle$	$A \cdot t_5 + 0$	$A \cdot t_5$
t_5	$\langle B, C, 0\rangle$	$B \cdot C + 0$	$B \cdot C$
t_4	$\langle A, t_6, 0\rangle$	$A \cdot t_6 + 0$	$A \cdot t_6$
t_6	$\langle B, D, 0\rangle$	$B \cdot D + 0$	$B \cdot D$
t_7	$\langle C, D, 0\rangle$	$C \cdot D + 0$	$C \cdot D$

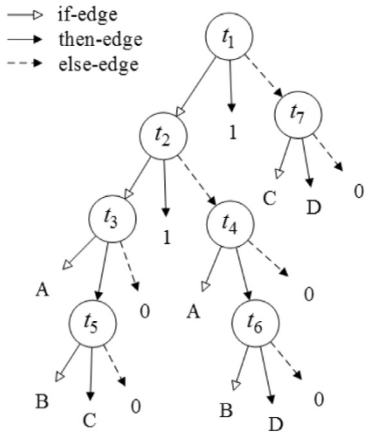

Fig. 4. An example of ZBED.

In the sequel, for the sake of the simplicity, we shall not make the distinction between nodes and their interpretation as formulas, i.e. we shall write simply $t = u \cdot v + w$. The set of basic events showing up in the ZBED rooted by the node t is denoted by $var(t)$.

Any coherent Boolean formula can be easily encoded by means of ZBED. The binary operators "$\cdot$" (and) and "+" (or) can be written as $u \cdot v = \langle u, v, 0\rangle$ and $u + w = \langle u, 1, w\rangle$. Fig. 4 shows an example of the ZBED whose interpretation is given in Table 4. The formula at top level is thus written as $((A \cdot (B \cdot C)) + (A \cdot (B \cdot D))) + (C \cdot D)$.

$$\begin{array}{lcl} \langle u, 1, 0\rangle & \rightarrow & u \\ \langle 1, v, w\rangle & \rightarrow & v \\ \langle 0, v, w\rangle & \rightarrow & w \\ \langle u, 0, w\rangle & \rightarrow & w \\ \langle u, w, w\rangle & \rightarrow & w \end{array}$$

Fig. 5. Rewriting rules used for node elimination.

3.2. Unicity of nodes

As for binary decision diagrams [11], ZBED nodes are managed in a unique table. Each time a node needs to be created, the table is looked-up to see whether a node with the same characteristics already exists. This technique is at the core of the efficiency of the decision diagrams technology.

Moreover, some simplifications are performed at node creation. Rewriting rules for eliminating equivalent nodes are given Fig. 5. The adjective "zero-suppressed" comes from the rule $\langle u, 0, w\rangle \rightarrow w$, which has been originally used by Minato for zero-suppressed binary decision diagrams [12]. In the sequel, we shall assume that these simplifications are systematically applied at node creation.

3.3. Indices

Variable nodes do not contain directly references to the variable they are labeled with. Rather, each variable is assigned an index and this index is used to label the node encoding this variable.

Both basic events and intermediate events are assigned an index. Nodes encoding intermediate events are not variable nodes, as they encode formulas. Nevertheless, as explained in the previous section, we may want to consider them as variable nodes, typically to factorize the ZBED. In this case, they are labeled with the index of the intermediate

$$
\begin{array}{rcll}
\texttt{cofactor}\ d\ E\ c & \rightarrow & d & \text{if } d \text{ is a Boolean constant} \\
\texttt{cofactor}\ E\ E\ c & \rightarrow & c & \\
\texttt{cofactor}\ F\ E\ c & \rightarrow & E & \text{if } F \text{ is a variable or a pseudo-variable node and } F \neq E \\
\texttt{cofactor}\ t\ E\ c & \rightarrow & t & \text{if } t.leastIndex > E.index \\
\texttt{cofactor}\ \langle u, v, w\rangle\ E\ c & \rightarrow & \langle u', v', w'\rangle & \text{where} \\
u' & = & \texttt{cofactor}\ u\ E\ c & \\
v' & = & \texttt{cofactor}\ v\ E\ c & \\
w' & = & \texttt{cofactor}\ w\ E\ c &
\end{array}
$$

Fig. 6. Recursive rewriting rules defining the cofactor algorithm.

$$
\begin{array}{rcll}
\texttt{factorize}\ c & \rightarrow & c & \text{if } c \text{ is a constant node} \\
\texttt{factorize}\ E & \rightarrow & E & \text{if } E \text{ is a variable or a pseudo-variable node} \\
\texttt{factorize}\ t & \rightarrow & \langle E, v, w\rangle & \text{if } t \text{ is an internal node and} \\
\multicolumn{4}{l}{\quad E \text{ is the variable or pseudo-variable node of } t \text{ with the least index} \\
\quad g & = & \texttt{cofactor}\ t\ E\ 1 & \\
\quad h & = & \texttt{cofactor}\ g\ E\ 0 & \\
\quad v & = & \texttt{factorize}\ g & \\
\quad w & = & \texttt{factorize}\ h &
\end{array}
$$

Fig. 7. Recursive rewriting rules defining the factorize algorithm.

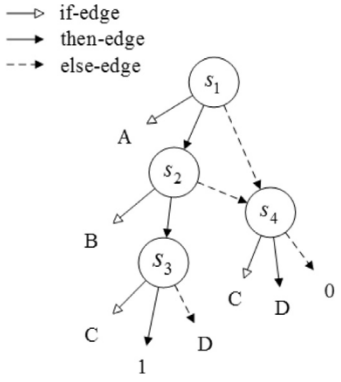

Fig. 8. Factorized version of the ZBED in Fig. 4 with *A*<*B*<*C*<*D*.

event they encode. In the sequel, we shall call such nodes pseudo-variable nodes.

Each node *t* embeds also the index *t.leastIndex* of the least variable occurring in the ZBED rooted by *t* (for variable nodes and pseudo-variables nodes, *t. leastIndex* = *t. index*).

We shall explain the reason of this labeling principle in the next section.

3.4. Cofactors

Model synthesis relies heavily on factorization. Let *f* be formula and *g* be subformula of *f*. Factorizing *f* with respect to *g* consists in rewriting *f* as $g{\cdot}f|_{g=1}+f|_{g=0}$, where $f|_{g=c}$ denotes the formula *f* in which the constant *c* has been substituted for the subformula *g*. $f|_{g=0}$ and $f|_{g=1}$ are called respectively the negative cofactor and the positive cofactor of *f* with respect to *g*.

In model synthesis, we factorize formulas only with respect to variable or pseudo-variable nodes. Recursive rules defining this operation on ZBED are given Fig. 6. The fourth recursive equation makes clear the interest of the *leastIndex* field of ZBED nodes: when *t.leastIndex* > *E.index*, we know for sure that the node *E* does not occur in the ZBED rooted by the node *t*. We can therefore stop here the exploration of this ZBED.

3.5. Caching

Most of operations on ZBED are defined recursively, similarly as the cofactor operation of the previous section. They take typically one or more ZBED as input and return a ZBED as output. The efficiency of

$$
\begin{array}{rcll}
\texttt{SoC}(0) & \stackrel{def}{=} & \emptyset & \\
\texttt{SoC}(1) & \stackrel{def}{=} & 1 & \\
\texttt{SoC}(E) & \stackrel{def}{=} & E & \text{If } E \text{ is a variable or a pseudo-variable node} \\
\texttt{SoC}(\langle E, v, w\rangle) & \stackrel{def}{=} & E \odot \texttt{SoC}(v) \cup \texttt{SoC}(w) &
\end{array}
$$

Fig. 9. Recursive rules to obtain SoC(t).

$$
\begin{array}{rcll}
\texttt{without}\ 0\ t & \rightarrow & 0 & \\
\texttt{without}\ 1\ t & \rightarrow & 1 & \text{if } t \neq 1 \\
\texttt{without}\ s\ 1 & \rightarrow & 0 & \\
\texttt{without}\ s\ 0 & \rightarrow & s & \\
\texttt{without}\ t\ t & \rightarrow & 0 & \\
\texttt{without}\ s\ t & \rightarrow & s & \text{if } s.leastIndex > t.index \\
\texttt{without}\ \langle E,s,t\rangle\ \langle E,v,w\rangle & \rightarrow & \langle E,s',t'\rangle & \text{where} \\
s'' & = & \texttt{without}\ s\ w & \\
s' & = & \texttt{without}\ s''\ v & \\
t' & = & \texttt{without}\ t\ w & \\
\texttt{without}\ \langle E,s,t\rangle\ \langle F,v,w\rangle & \rightarrow & \langle E,s',t'\rangle & \text{if } E.index < F.index \text{ and} \\
s' & = & \texttt{without}\ s\ \langle F,v,w\rangle & \\
t' & = & \texttt{without}\ t\ \langle F,v,w\rangle & \\
\texttt{without}\ \langle E,s,t\rangle\ \langle F,v,w\rangle & \rightarrow & \texttt{without}\ \langle E,s,t\rangle\ w & \text{if } F.index < E.index
\end{array}
$$

Fig. 10. Recursive rewriting rules defining the without algorithm.

$$
\begin{array}{rcll}
\texttt{minimize}\ c & \rightarrow & c & \text{if } c \text{ is a constant node} \\
\texttt{minimize}\ E & \rightarrow & E & \text{if } E \text{ is a variable or a pseudo-variable node} \\
\texttt{minimize}\ \langle E,s,t\rangle & \rightarrow & \langle E,v,w\rangle & \text{where} \\
u & = & \texttt{minimize}\ s & \\
w & = & \texttt{minimize}\ t & \\
v & = & \texttt{without}\ u\ w &
\end{array}
$$

Fig. 11. Recursive rewriting rules defining the minimize algorithm.

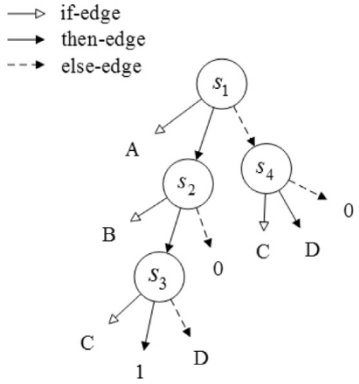

Fig. 12. The minimized ZBED of Fig. 8.

these operations can be significantly improved by using caching: each time the operation *op* must be performed on parameters p_1,...p_k, the cache is looked up. If it contains an entry for *op* and p_1,...p_k, the result is immediately returned. Otherwise, the operation is actually performed, then cached.

It is again Bryant &. al [11] who introduced this idea in the binary decision diagram technology.

In the sequel, we shall assume that all algorithms use such caching technique.

3.6. Factorization and ordering

It is possible to calculate a normal form for ZBED by factorizing them recursively with respect to basic events. A ZBED is factorized if either it is reduced a constant or a variable node, or it is rooted by a node $\langle E, v, w\rangle$ where E is a variable node and v and w are ZBED in which E does not show up.

Let $\prec$ be a total order over variables. A factorized ZBED is ordered with respect to $\prec$ if any of its node $\langle E, v, w\rangle$, any variable F showing up in ZBED in either v or w verifies $E \prec F$.

The recursive rules defining the algorithm that rewrite a ZBED into an equivalent factorized and ordered one are given Fig. 7. To be more efficient, a node is not replaced by a new one, but rather transformed into a factorized one. In this way, all the nodes pointing to the node get the factorized version at no additional cost. This principle has been introduced for dynamic reordering of binary decision diagrams [14].

Fig. 8 shows the factorized version of the ZBED pictured Fig. 4 according to the ordering $A \prec B \prec C \prec D$. The fully factorized ZBED ordered according to the order $\prec$ over the variables (basic events) of a formula f is isomorphic to the reduced ordered (according to $\prec$) binary decision diagram encoding f. The two diagrams differ however completely in the way they are obtained. The binary decision diagram associated with a formula f is built by composing the binary decision diagrams associated with the sub-formulas of f. The spaces (data structures) of formulas and binary decision diagrams are thus separated. With ZBED, both formulas

$$
\begin{array}{rcll}
\texttt{normalize}\ c & \rightarrow & c & \text{if } c \text{ is a constant node} \\
\texttt{normalize}\ E & \rightarrow & E & \text{if } E \text{ is a variable or a pseudo-variable node} \\
\texttt{normalize}\ t & \rightarrow & \langle E, v, w\rangle & \text{if } t \text{ is an internal node and} \\
\multicolumn{4}{l}{\quad E \text{ is the variable or pseudo-variable node of } t \text{ with the least index}} \\
\multicolumn{4}{l}{\quad g = \texttt{cofactor}\ t\ E\ 1} \\
\multicolumn{4}{l}{\quad h = \texttt{cofactor}\ t\ E\ 0} \\
\multicolumn{4}{l}{\quad u = \texttt{normalize}\ g} \\
\multicolumn{4}{l}{\quad w = \texttt{normalize}\ h} \\
\multicolumn{4}{l}{\quad v = \texttt{without}\ u\ w}
\end{array}
$$

Fig. 13. Recursive rewriting rules defining the normalize algorithm.

$$
\begin{array}{rcll}
\texttt{p-normalize}\ c\ q & \rightarrow & c & \text{if } c \text{ is a constant node} \\
\texttt{p-normalize}\ E\ q & \rightarrow & 0 & \text{if } E \text{ is a variable or a pseudo-variable node and } q \times p(E) < \tau \\
\texttt{p-normalize}\ E\ q & \rightarrow & E & \text{if } E \text{ is a variable or a pseudo-variable node and } q \times p(E) \geq \tau \\
\texttt{p-normalize}\ t\ q & \rightarrow & t & \text{if } t.leastIndex \geq i_{max} \\
\texttt{p-normalize}\ t\ q & \rightarrow & w & \text{if } t \text{ is an internal node and} \\
\multicolumn{4}{l}{\quad E \in \mathcal{S} \text{ is the variable or pseudo-variable node of } t \text{ with the least index and } q \times p(E) < \tau} \\
\multicolumn{4}{l}{\quad h = \texttt{cofactor}\ t\ E\ 0} \\
\multicolumn{4}{l}{\quad w = \texttt{p-normalize}\ h\ q} \\
\texttt{p-normalize}\ t\ q & \rightarrow & \langle E, v, w\rangle & \text{if } t \text{ is an internal node and} \\
\multicolumn{4}{l}{\quad E \in \mathcal{S} \text{ is the variable or pseudo-variable node of } t \text{ with the least index and } q \times p(E) \geq \tau} \\
\multicolumn{4}{l}{\quad g = \texttt{cofactor}\ t\ E\ 1} \\
\multicolumn{4}{l}{\quad h = \texttt{cofactor}\ t\ E\ 0} \\
\multicolumn{4}{l}{\quad u = \texttt{p-normalize}\ g\ q \times p(E)} \\
\multicolumn{4}{l}{\quad w = \texttt{p-normalize}\ h\ q} \\
\multicolumn{4}{l}{\quad v = \texttt{without}\ u\ w}
\end{array}
$$

Fig. 14. Recursive rewriting rules defining the p − normalize algorithm.

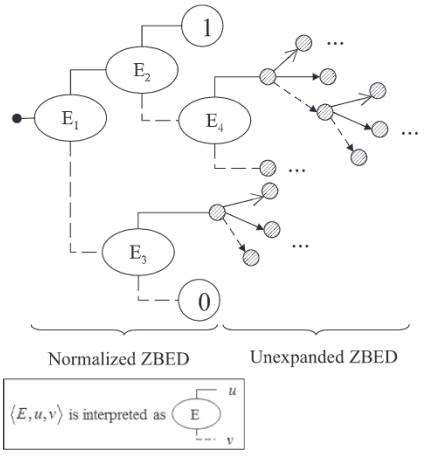

Fig. 15. Example of partially normalized ZBED.

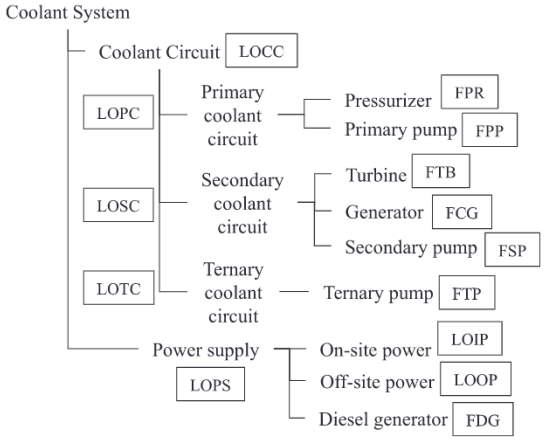

Fig. 16. An architecture of the coolant system in Fig. 1.

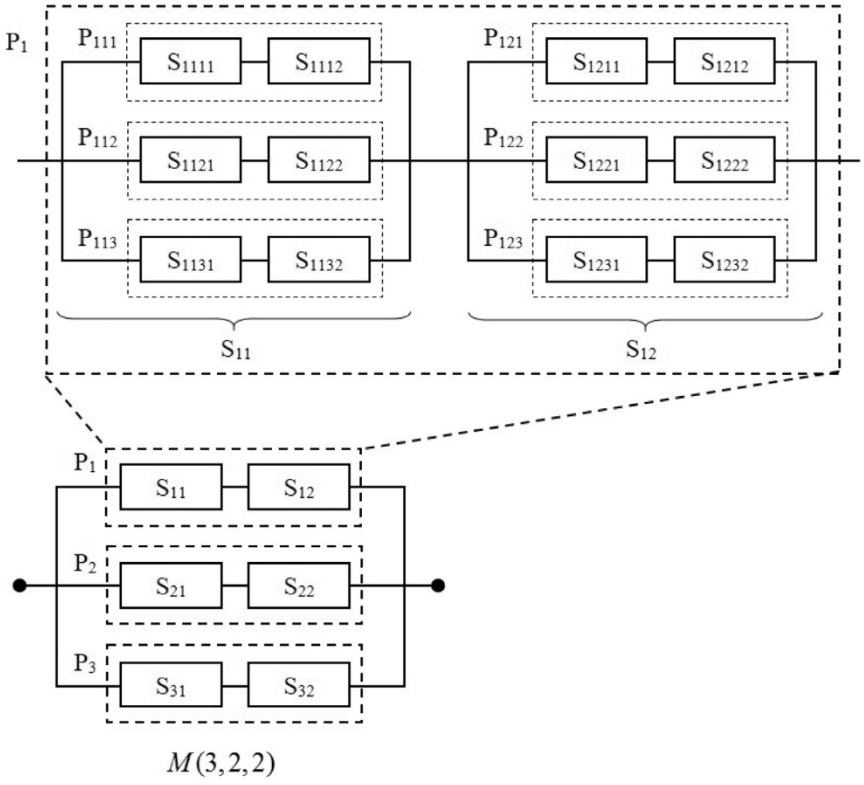

Fig. 17. Parametric model $M(3, 2, 2)$.

Table 5
Fault tree describing the failures of $M(3, 2, 2)$ in Fig. 17.

$TOP = P_1 + P_2 + P_3$	
$P_i = S_{i1} \cdot S_{i2}$	(i = 1,2,3)
$S_{ij} = P_{ij1} + P_{ij2} + P_{ij3}$	(j = 1,2)
$P_{ijk} = S_{ijk1} \cdot S_{ijk2}$	(k = 1,2,3)
$S_{ijkl} = C_{ijkl} + SS_l$	(l = 1,2)

and their normal forms are encoded within the same space.

3.7. Minimization

In general, once factorized, a ZBED becomes more compact. It is however often possible to make it even more compact by removing non-minimal branches. To explain this process, we shall first show how to interpret a ZBED as a set of cutsets, or equivalently as a sum-of-products.

Let E be a basic event and s be a set of cutsets. We define the product $E \odot s$ as follows:

$$E \odot s \stackrel{def}{=} \{E.\,\pi; \pi \in s\}$$

Let t be a factorized, ordered ZBED. $\mathrm{SoC}(t)$ denotes the set of cutsets interpretation of a t. Recursive rules to build $\mathrm{SoC}(t)$ are given Fig. 9. Consider again the ZBED pictured in Fig. 8. We have $\mathrm{SoP}(s1) = \{A.\,B.\,C, A.\,B.\,D, A.\,C.\,D, A.\,D\}$.

Note that $\mathrm{SoP}(s1)$ contains the non-minimal cutset *A.C.D*.

The method to minimize, i.e to remove non-minimal cutsets from a factorized ordered ZBED $\langle E, v, w\rangle$ works into two steps. First, it minimizes v into $v_{\min}$ and w into $w_{\min}$. Second, it removes from $v_{\min}$ all the cutsets π such that there exists a cutset ρ in $\mathrm{SoP}(w)$ such that $\rho \subseteq \pi$. This second operation is performed by the without algorithm introduced for zero-suppressed binary decision diagrams by one of the author in [15] and further improved in [16].

Figs. 10 and 11 give respectively recursive rewriting rules defining without and minimize algorithms adapted to ZBED. The minimized version of the ZBED pictured Fig. 8 is pictured Fig. 12. Note that the above minimization principle is similar to the one propose by Jung [17] to calculate minimal cutsets using zero-suppressed binary decision diagrams.

Table 6
Full normalization of $M(p, s, d)$ with order of variables corresponding to a depth-first left-most traversal of the model.

M			# of fault tree events		# of ZBED nodes		Running time (s)
p	s	d	Basic	Intermediate	Original	Fully normalized	
2	2	1	6	7	15	15	0.0005875
2	3	1	8	9	21	20	0.001016
2	4	1	10	11	27	25	0.0016374
3	2	1	9	10	22	29	0.0027617
3	3	1	12	13	31	39	0.0048888
3	4	1	15	16	40	49	0.005376
4	2	1	12	13	29	55	0.0059631
4	3	1	16	17	41	74	0.0257535
4	4	1	20	21	53	93	0.0238569
5	5	1	30	31	81	213	0.0929677
6	6	1	42	43	115	479	0.7946444
7	7	1	56	57	155	1067	8.5812105
8	8	1	72	73	201	2361	140.4506352
2	2	2	18	31	51	43	0.0105937
2	3	2	38	57	111	91	0.0275659
2	4	2	66	91	195	159	0.0564192
2	5	2	102	133	303	247	0.1878167
3	2	2	39	64	112	126	0.0375149
3	3	2	84	121	247	273	0.3376026
3	4	2	147	196	436	480	2.1421809
4	2	2	68	109	197	317	1.0885322
2	2	3	66	127	195	159	0.0922675
2	3	3	218	345	651	529	0.8691808

Table 7
Partial normalization (factorization of SS_i) comparing with a full normalization compatible with the factorization.

M			# of fault tree events		# of ZBED nodes			Running time (s)	
p	s	d	Basic	Intermediate	Original	Partial	Full	Partial	Full
2	2	1	6	7	15	14	15	0.0003807	0.0005609
2	3	1	8	9	21	18	20	0.0014846	0.0010145
3	3	1	12	13	31	31	39	0.001792	0.0035515
3	4	1	15	16	40	37	49	0.0027814	0.0315186
3	5	1	18	19	49	43	59	0.0034137	0.0202382
4	5	1	24	25	65	68	112	0.009577	0.0249844
5	5	1	30	31	81	109	213	0.0198436	0.0790558
6	6	1	42	43	115	194	479	0.0564447	0.778016
7	7	1	56	57	155	347	1067	0.1532794	8.763043
8	8	1	72	73	201	632	2361	0.463898	145.5475206
2	3	2	38	57	111	78	91	0.0050992	0.0220251
2	4	2	66	91	195	134	159	0.0099643	0.0960328
2	5	2	102	133	303	206	247	0.0183755	0.130448
3	2	2	39	64	112	85	126	0.0107254	0.0507543
3	3	2	84	121	247	175	273	0.0760447	0.3209596
3	4	2	147	196	436	301	480	0.0449491	1.7469204
4	2	2	68	109	197	156	317	0.033493	1.0738509
2	2	3	66	127	195	134	159	0.0148028	0.0839745
2	3	3	218	345	651	438	529	0.043567	0.8689405
2	4	3	514	731	1539	1030	–	0.1329544	–
3	2	3	219	388	652	445	–	0.0730657	–
3	3	3	732	1093	2191	1471	–	0.3639312	–
3	4	3	1731	2356	5188	3469	–	1.193553	–
3	5	3	3378	4339	10,129	6763	–	3.7578617	–
4	2	3	516	877	1541	1052	–	0.3506644	–
4	3	3	1732	2513	5189	3484	–	2.1515793	–
4	4	3	4100	5461	12,293	8220	–	9.837303	–
4	5	3	8004	10,105	24,005	16,028	–	31.5285081	–

3.8. Normalization

The normalization of a ZBED performs simultaneously both factorization and minimization, which is more efficient than applying one operation after the other. The recursive rewriting rules defining the normalization algorithm are given Fig. 13. The fully normalized ZBED ordered according to the order $\prec$ over the variables (basic events) of a formula f is isomorphic to the reduced ordered (according to $\prec$) zero-suppressed binary decision diagram encoding the minimal cutsets of f. Here again, the two diagrams differ in the way they are obtained.

4. Partial normalizations

The advantage of the ZBED technology over the (zero-suppressed) binary decision diagram technology stands in the ability to perform partial operations, including partial factorizations, minimizations and normalizations, thanks to the encoding of both formulas and their normal forms within the same data structures.

4.1. Algorithm

A ZBED can be seen as (a compact encoding of) a decision tree. As discussed Section 2, it is often sufficient to develop only partially such decision tree to get relevant information. More exactly, the partial development of a decision tree involves: first, a subset $\mathcal{S}$ of basic and intermediate events of interest, together with an order over these events; and second, a probability threshold τ under which branches can be discarded. $\mathcal{W}$ is called the care set. It contains the events on which is decomposition of the ZBED will be performed.

We shall thus modify algorithms presented in the previous section so to perform partial normalizations. This works as follows.

The first step consists in giving indices $1, 2, \ldots, i_{\max}$ to events of $\mathcal{S}$, including intermediate events, according to the order in which we want them to show up in the decision tree. The remaining basic events are given indices $i_{\max} + 1$, $i_{\max} + 2$, ... The ZBED is re-labeled according to these indices. This operation is linear in the size of the ZBED.

The second step consists in modifying the normalize so to take into account the set $\mathcal{S}$ and the threshold τ. Recursive rewriting rules defining the $\mathrm{p} - \mathrm{normalize}$ algorithm are given Fig. 14. The algorithm is initially called with its second parameter set to 1.0. In the last two rules, if the pivot variable E is pseudo-variable, $p(E)$ is set to 1.0. It is possible to take the actual probability of E (or an approximation of this probability) only if E is a module, i.e. it shares no variable with the rest of the model.

A general representation of the partially normalized ZBED is given Fig. 15. In the normalized part of this ZBED is essentially similar to decision trees (or event trees) of Section 2.

4.2. Choice of the care set

In model synthesis, we suggest to use the system architecture as a guideline to achieve informative rewritings. As exemplified in Fig. 16, an architecture can be seen as a functional or physical decomposition of the system, in which events (marked in rectangle) can be assigned to their relevant functions or components. Based on such assignment, traversal algorithms (like the depth-first-left-most algorithm) can be implemented to order and group events automatically in the architecture.

Guiding by the system architecture, the ZBED can be rewritten in a way that maps the architecture, which means to create similarities in their way of decomposition between the ZBED and the architecture. With the help of this mapping, we provide a possibility to check the consistency between the fault tree model and the system design.

5. Experiments

In this section, we provide the scalability test of the proposed model synthesis method. The model used to perform the test is a multilevel parallel-series system, denoted by $M(p, s, d)$, which is characterized by three parameters:

- p: the number of parts in parallel at each layer
- s: the number of parts in series at each layer
- d: the depth of alternation

Fig. 17 gives an example of $M(3, 2, 2)$. The last level of the hierarchy is made of a series of two components: a local independent component C and a support system SS dedicated to the first parallel line. For instance, the unit S_{ijkl} ($i = 1, 2, 3$; $j = 1, 2$; $k = 1, 2, 3$; $l = 1, 2$) in the last level of the model $M(3, 2, 2)$ is comprised by the series of C_{ijkl} and SS_i, where SS_i is dedicated to P_i.

The fault tree describing the failures of $M(3, 2, 2)$ is exemplified in Table 5.

Two kinds of experiments are implemented:

- A full normalization of the model with an order of variables corresponding to a depth-first left-most traversal of the model, of which the results are given in Table 6.
- A partial normalization (factorization of SS_i) comparing with a full normalization compatible with the factorization, of which the results are given in Table 7. For the last nine cases in Table 7, the full normalization is not proceeded since the running time is more than 2000 seconds. It also shows that partial normalizations are far more efficient when dealing with large models.

6. Conclusion

In this article, we propose a new method, called model synthesis, which consists in rewriting the fault tree under study so to make some relevant information emerge. The rewriting relies on an encoding of Boolean formulas by means of zero-suppressed Boolean expression diagrams. A key feature of zero-suppressed Boolean expression diagrams is that they make it possible to perform partial normalization of Boolean formulas. To get relevant information, it is often sufficient to develop only partially the diagram, which is also more efficient than a full factorization. For partial normalization, analyst can choose a care set guided by the system architecture and a probability threshold τ to decide under which branches can be discarded. As future work, we plan to extend this method to non-coherent systems and to support other quantitative analyses like the calculation of importance measures [18].

References

[1] Kumamoto H, Henley EJ. Probabilistic risk assessment and management for engineers and scientists. Piscataway, N.J., USA: IEEE Press; 1996. 978–0780360174.
[2] Andrews JD, Moss RT. Reliability and risk assessment. (2nd ed.) Materials Park, Ohio 44073-0002, USA: ASM International; 2002. 978–0791801833.
[3] Rausand M, Arnljot H, et al. System reliability theory: models, statistical methods, and applications. 396. John Wiley & Sons; 2004. https://doi.org/10.1002/9780470316900.
[4] Ruijters E, Stoelinga M. Fault tree analysis: a survey of the state-of-the-art in modeling, analysis and tools. ComputSciRev 2015;15:29–62. https://doi.org/10.1016/j.cosrev.2015.03.001.
[5] Xiang J, Yanoo K, Maeno Y, Tadano K. Automatic synthesis of static fault trees from system models. IEEE Publishing978-1-4577-0780-3; 2011. p. 127–36. https://doi.org/10.1109/SSIRI.2011.32.
[6] Camarda P, Corsi F, Trentadue A. An efficient simple algorithm for fault tree automatic synthesis from the reliability graph. Reliab IEEE Trans 1978;R-27(3):215–21. https://doi.org/10.1109/TR.1978.5220330.
[7] Elliott M. Computer-assisted fault-tree construction using a knowledge-based approach. Reliab IEEE Trans 1994;43(1):112–20. https://doi.org/10.1109/24.285124.
[8] Prosvirnova T, Rauzy A. Automated generation of minimal cutsets from altarica 3.0 models. Int J Crit Comput-Based Syst 2015;6(1):50–79. https://doi.org/10.1504/IJCCBS.2015.068852.
[9] Leblond A. Synthèse de coupes minimales fonctionnelles en coupes minimales composant. Actes du 19ième congrés Lambda-Mu. Dijon, France: Institut pour la Maîtrise des Risques; 2014.
[10] Andersen HR, Hulgaard H. Boolean expression diagrams. Information and computation 2002;179(2):194–212. https://doi.org/10.1006/inco.2001.2948.
[11] Brace KS, Rudell RL, Bryant RS. Efficient implementation of a BDD package. Proceedings of the 27th ACM/IEEE design automation conference. Orlando, Florida, USA: IEEE0-89791-363-9; 1990. p. 40–5. https://doi.org/10.1145/123186.123222.
[12] Minato S-I. Zero-suppressed BDDs for set manipulation in combinatorial problems. Proceedings of the 30th ACM/IEEE design automation conference, DAC'93. Dallas, Texas, USA: IEEE0-89791-577-1; 1993. p. 272–7. https://doi.org/10.1145/157485.164890.
[13] Getir S, Van Hoorn A, Grunske L, Tichy M. Co-evolution of software architecture and fault tree models: an explorative case study on a pick and place factory automation system. 1074. CEUR-WS; 2013. p. 32–9.
[14] Rudell RL. Dynamic variable ordering for ordered binary decision diagrams. In: Lightner M, Jess JAG, editors. Proceedings of IEEE International Conference on Computer Aided Design, ICCAD'93. Santa Clara, CA, USA: IEEE0-8186-4490-7; 1993. p. 42–7.
[15] Rauzy A. New algorithms for fault trees analysis. Reliab Eng Syst Saf 1993;05(59):203–11. https://doi.org/10.1016/0951-8320(93)90060-C.
[16] Rauzy A. Mathematical foundations of minimal cutsets. IEEE Trans Reliab 2001;50(4):389–96. https://doi.org/10.1109/24.983400.
[17] Jung WS, Han SH, Ha J. A fast bdd algorithm for large coherent fault trees analysis. Reliab Eng Syst Saf 2004;83(3):369–74. https://doi.org/10.1016/j.ress.2003.10.009.
[18] Aliee H, Borgonovo E, Glaß M, Teich J. On the boolean extension of the birnbaum importance to non-coherent systems. Reliab Eng Syst Saf 2017;160. http://search.proquest.com/docview/1944562495/?pq-origsite=primo