Title	A dynamic flowgraph methodology approach based on binary decision diagrams
Author(s)	Björkman, Kim; Karanta, Ilkka
Citation	Proceedings of 2011 International Topical Meeting on Probabilistic Safety Assessment and Analysis (PSA 2011), Wilmington, NC, 13 – 17 March 2011, pp. paper 80
Date	2011
Rights	Reprinted from Proceedings of ANS PSA 2011 International Topical Meeting on Probabilistic Safety Assessment and Analysis. This article may be downloaded for personal use only

ANS PSA 2011 International Topical Meeting on Probabilistic Safety Assesment and Analysis
Wilmington, NC, March 13-17, 2011, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2011)

A DYNAMIC FLOWGRAPH METHODOLOGY APPROACH BASED ON BINARY DECISION DIAGRAMS

Kim Björkman and Ilkka Karanta
VTT Technical Research Centre of Finland
P.O. Box 1000, FI-02044 VTT, Finland
kim.bjorkman@vtt.fi; ilkka.karanta@vtt.fi

ABSTRACT

The dynamic flowgraph methodology (DFM) is an approach to model and analyze the behavior of dynamic systems for reliability assessment. The methodology can be utilized to identify how certain postulated top events may occur in a system. The result is a set of prime implicants which represent system faults resulting from diverse combinations of software logic errors, hardware failures, human errors, and adverse environmental conditions. A binary decision diagram (BDD) is a data structure used to represent Boolean functions applied, e.g., in fault tree analysis and model checking. This paper presents an alternative DFM approach based on BDD called YADRAT. The objective of a YADRAT model analysis is to find the root causes of the query (top event) of interest, similarly to traditional fault tree analysis. The main difference of YADRAT compared to the existing DFM approach is that YADRAT employs a BDD to represent a DFM model. Two different approaches to solving a BDD model have been implemented for exact computation of prime implicants. These approaches have previously been applied in static failure tree analysis. In this work the ideas for prime implicant calculation are adapted to a dynamic reliability approach combined with the multi-valued logic of DFM. In this paper the basic concepts and algorithms of YADRAT and the identified strengths and limitations of the employed approach are discussed. Also a case study illustrating the usage of YADRAT and a comparison of computational effort between two BDD implementations is presented.

Key Words: reliability analysis, binary decision diagram, dynamic flowgraph methodology

1 INTRODUCTION

The static event tree/ fault tree (ET/FT) approach has been widely used in reliability modeling especially in the nuclear power plant domain. However, its inability to capture time dependent dynamic behaviors has made the use of fault trees somewhat impractical in assessing the reliability of dynamic systems.

The dynamic flowgraph methodology (DFM) is an approach to model and analyze the behavior of dynamic systems for reliability assessment. The modeling approach of DFM is promising, since it is attractive for higher level abstraction of a dynamic system. The methodology can be utilized to identify how certain top events may occur in a system. The result is a set of prime implicants which represents system faults resulting from diverse combinations of software logic errors, hardware failures and adverse environmental conditions.

This paper presents an alternative DFM approach based on binary decision diagrams (BDD) called YADRAT. The main difference of YADRAT compared to the existing DFM approach is that YADRAT employs a BDD to represent a DFM model. Two different approaches to solving a BDD model have been implemented for the exact computation of prime implicants. Both approaches have previously been applied in traditional static failure tree analysis [17, 18]. In this work the ideas for prime implicant calculation are adapted to a dynamic reliability approach combined with the multi-valued logic of DFM. Besides presenting the basic concepts and algorithms of YADRAT also a case study illustrating the usage of YADRAT and a comparison of computational effort between two BDD implementations are presented.

2 BINARY DECISION DIAGRAMS

A Binary Decision Diagram [5, 6] is a data structure used to represent Boolean functions. The BDD is based on the repeated application of the classical Shannon expansion formula:

$$F(x_1, x_2, x_3, ...) = x_1 \cdot F(1, x_2, x_3, ...) + \bar{x}_1 \cdot F(0, x_2, x_3, ...) \qquad (1)$$

where x_1, x_2, x_3 are Boolean variables.

The Boolean function is represented as a rooted, directed acyclic graph that consists of decision nodes with two edges the 1-edge and 0-edge, and terminal nodes called 0-terminal and 1-terminal representing the Boolean functions 0 and 1. A variable assignment for which the represented Boolean function is true is represented by a path from the root node to 1-terminal node. A BDD with the constraint that the input variables are ordered and every decision node to terminal node path in the BDD visits the input variables in ascending order is called an ordered binary decision diagram (OBDD). The size of the OBDD representation can be considerably reduced by the following two reduction rules (Figure 1).

- remove redundant nodes whose two edges point to same node
- share equivalent sub-graphs.

This reduced OBDD is called a reduced ordered binary decision diagram (ROBDD). Bryant [5] demonstrated how a BDD could be modified to an ROBDD so that a canonical representation a Boolean function could be created. In the rest of the paper a BDD is understood to mean the reduced and ordered form of a BDD.

The size of a BDD for a given function depends on the variable order chosen for the function. Problem of finding the order that minimizes the size of the BDD, is NP-hard.

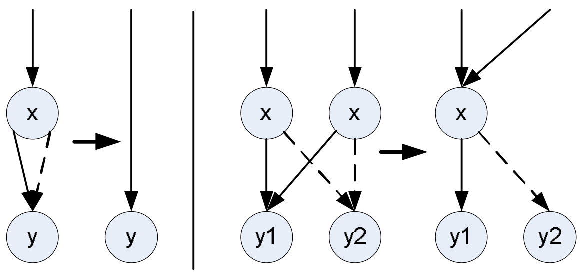

Figure 1. Reduction rules for BDD

Another type of BDD is the Zero-suppressed BDD (ZBDD) introduced by Minato [14]. A ZBDD is a BDD with different semantics and reduction rules. The following reduction rules are used:

- remove the nodes whose 1-edge points to 0-terminal node. Then connect the edge leading to the node to the other sub-graph directly (Figure 2).
- share equivalent sub-graphs (as for original BDDs).

Contrary to the original BDD the nodes whose two edges point to the same node are not removed. This reduction rule is asymmetric for the two edges, because the nodes whose 0-edge points to terminal nodes are not removed. As the original form the ZBDD representation is canonical.

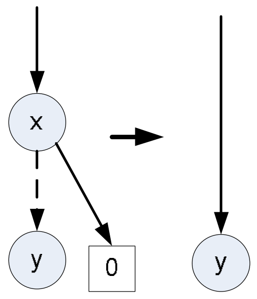

Figure 2. Reduction rule for ZBDD

3 DYNAMIC FLOWGRAPH METHODOLOGY

The dynamic flowgraph methodology is an approach to modeling and analyzing the behavior of dynamic systems for reliability/safety assessment and verification [9]. DFM models express the logic of the system in terms of causal relationships between physical variables and states of the system; the time aspects of the system (execution of control commands, dynamics of the process) are represented as a series of discrete state transitions. DFM can be used for identifying how certain postulated events may occur in a system; the result is a set of timed fault trees, whose prime implicants (multi-state analogue of minimal cut sets) can be used to identify system faults resulting from unanticipated combinations of software logic errors, hardware failures, human errors and adverse environmental conditions. DFM has been used to assess the reliability of nuclear power plant control systems [1], but also of space rockets [21] and chemical batch processes [10].

DFM models are directed graphs, analyzed at discrete time instances. They consist of variable and condition nodes; causality and condition edges; and transfer and transition boxes and their associated decision tables. A node represents a variable that can be in one of a finite number of predefined states. The state of a node can change at discrete time instances. The state of the node is determined by the states of its input nodes. Each node can have several inputs but only one output — its state. The state of the node can act as an input to possibly several other nodes. The state of a node at time t is determined by

- the states of its input nodes at a single instance of time (say, $t - n$)
- the lag n, an integer that tells how many time instances it takes for an input to cause the state of the present node.

The state of a node — as a function of the states of its input nodes — is determined by a decision table. A decision table is an extension of the truth table where each variable can be represented with any finite number of states. The decision table contains a row for each possible combination of input variable states. The maximum possible number of rows in the decision table is the product of the numbers of states of the input nodes.

After construction, the DFM model can be analyzed in two different modes, deductive and inductive [10]. In inductive analysis, event sequences are traced from causes to effects; this corresponds to simulation of the model. In deductive analysis, event sequences are traced backward from effects to causes.

A deductive analysis starts with the identification of a particular system condition of interest (a top event); usually this condition corresponds to a failure. To find the root causes of the top event, the model is backtracked for a predefined number of steps through the network of nodes, edges, and boxes. This means that the model is worked backward in the cause-and-effect flow to find what states of variables (and at what time instances) are needed to produce the top event. The result of a deductive analysis is a set of prime implicants.

A prime implicant consists of a set of triplets (V, S, T); each triplet tells that variable V is in a state S at time T. The circumstances described by the set of triplets cause the top event. Prime implicants are similar to minimal cut sets of fault tree analysis, except that prime implicants are timed and they deal with multi-valued variables (fault trees deal with Boolean variables). A useful analogy is that deductive analysis corresponds to minimal cut set search of a fault tree. Once primary implicants have been found, the top event probability can be quantified as in the MCS analysis of a fault tree.

4 YADRAT

The YADRAT approach [3] is from the modeling point of view similar to DFM [9]. A YADRAT model is a directed graph. Each node of the graph represents a system variable and the arcs between the nodes represent input-output relationships between variables. In YADRAT boxes have been left out from the graph, because the decision tables contained in them can be attached to the output node of the box. Additionally, variable and condition nodes have been combined into one node type; in DFM, the distinction between these is only for documentation purposes, and no semantic difference exists between them from the DFM point of view. Similarly, causality and condition edges have been merged into one edge type. Figure 3 illustrates a simple YADRAT directed graph.

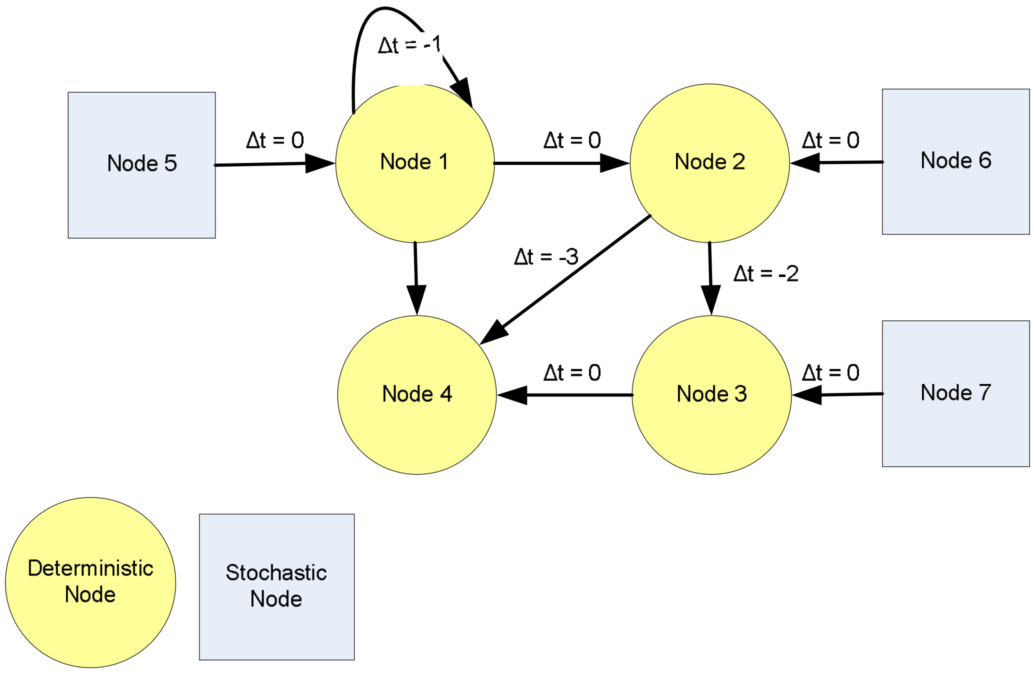

Figure 3. The graph of a simple YADRAT model.

In the YADRAT approach, for a single output node time lags can be constant or each input can have a different time lag. The latter case is relevant, e.g., in cases where the node depends on its previous value. This doesn't add expressiveness to DFM – in ordinary DFM models different time lags can be expressed by adding new variables – but makes the models simpler (less nodes).

Another modification of the YADRAT approach to the ordinary DFM models is that variable nodes are divided into two categories: deterministic and stochastic. The state of a deterministic node is determined by the values of the input nodes via the decision table. The state of a stochastic variable is determined stochastically (multinomial probability distribution), and a stochastic variable doesn't have any inputs. We denote deterministic nodes by circles and stochastic nodes by squares.

YADRAT only supports deductive analysis of DFM models. The deductive analysis of YADRAT is similar to the deductive analysis of DFM as described in section 3.

5 IMPLEMENTATION OF YADRAT

5.1 Logical operations on BDD

For the representation of Boolean functions as BDDs the ite (If-Then-Else) connective is usually used. ite is a Boolean function defined for three inputs F, G, and H, which computes: if F then G else H. The ite connective is defined as follows. Let F, G and H be Boolean functions then

$$ite(F,G,H) = (F \cdot G) + (\overline{F} \cdot H) \qquad (2)$$

The ite operation can be used to implement all two-variable Boolean functions. Because ite is the logical function performed at each node of the BDD, it can efficiently be used as a building block for many other operations on the BDD.

A hash table is used to store the ite nodes (the ite-table). The ite-table maps a triple (x, G, H) to a BDD node F = (x, G, H). The main idea of the ite-table is to guarantee that each node is unique. That is, there is no other node labeled by the same variable and with the same children. This uniqueness property makes decision diagrams canonical. Each node has an entry in the ite-table. Before a new node is added to the table, a lookup is performed in the ite-table to determine if a node for that function already exists. If a node already exists, it is used. Otherwise, the new node is added to the table. It is assumed that when creating a new node F, G and H are already in the strong canonical form. The sharing of equivalent sub-trees to one or several BDDs is automatically performed.

Bryant [5] noted that the performance of the ite function can be improved by introducing an additional table, called the computation-table that maintains the results of previous computations. This idea has been used in several implementations [4, 16]. When applying the algorithm to two nodes, the computation-table is first checked. If the table contains an entry of the computation, the result can immediately be returned.

5.2 BDD construction of Decision Tables

The binary decision diagram is constructed based on the decision tables. A memory function is used to improve the performance of the construction. The idea of the memory function is the same as in the computation of ite-nodes.

The size of the resulting BDD depends heavily on the variable order. At this stage the variable order is the order where nodes are created (depends on how user created model) and how the underlying BDD software might dynamically reorder the variables.

5.3 Representing states by BDDs

Let S be a finite set of states of a variable. Because BDDs encode Boolean functions, the elements of S need to be coded as Boolean values. Without loss of generality, we may assume here that S = {0, 1, ..., k–1}, meaning that |S| = k.

A way to do this is to assign to each element $s \in S$, a unique vector of Boolean values $(v_1, v_2, \ldots, v_n)$, each $v_i \in [0,1]$ [11]. There are 2^n Boolean vectors of length n. Therefore n should be chosen such that $2^{n-1} < |S| \leq 2^n$, where |S| is the number of elements of S. For example, let S be {0, 1, …, 7}. S contains 8 states and it can be represented with a Boolean vector of size 3 ($2^3 = 8$) (v_1, v_2, v_3)(e.g. vector (0, 0, 0) represents state 0, (0, 1, 1) represents state 3, and (1, 1, 1) represents state 7).

5.4 Calculation of Prime Implicants

YADRAT supports two approaches for exact computation of prime implicants. Both approaches utilize the same decomposition theorem [7]. Let $F(x_1, \ldots, x_n)$ be a Boolean function. Then the set of prime implicants can be attained as the union of three sets

$$
\begin{aligned}
&PI = PI\left[F(1, x_2, \ldots, x_n) \cdot F(0, x_2, \ldots, x_n)\right] \cup \\
&\{\bar{x}_1\} \cdot \left(PI\left[F(0, x_2, \ldots, x_n)\right] \setminus PI\left[F(1, x_2, \ldots, x_n) \cdot F(0, x_2, \ldots, x_n)\right]\right) \cup \\
&\{x_1\} \cdot \left(PI\left[F(1, x_2, \ldots, x_n)\right] \setminus PI\left[F(1, x_2, \ldots, x_n) \cdot F(0, x_2, \ldots, x_n)\right]\right)
\end{aligned} \tag{3}
$$

where \ stands for set difference.

The idea of the first approach is to represent set of products with Boolean formulae namely meta-products [7]. Meta-products are used to have a canonical representation of set of products and the idea is to use a BDD to efficiently represent the meta-products.

The second approach uses a ZBDD to represent the prime implicants. In this approach nodes are labeled with literals (not just by variables) and to decompose sets of products according to the presence of a given literal [17]. The idea is to traverse the BDD representing the top event in a depth-first way and to build the ZBDD in a bottom-up way, by collecting the results of the BDD traversal. The complete cover of prime implicants can be found from the prime implicant (Z)BDDs by computing all minterms.

5.5 Calculation of Top event probabilities

For each state of a stochastic node a probability is specified. Additionally, a probability is specified for each state of a deterministic node for the initial time. The probability of each prime implicant can be computed by multiplying the probabilities of each node of a prime implicant being at given state at given time. The total probability of a top event is approximated based on the prime implicant’s probabilities as follows:

$$
P_{tot} = 1 - \prod_{i=1}^{n} (1 - P_i) \tag{4}
$$

where P_{tot} is the top event probability, P_i is the probability of i:th prime implicant and n total number of prime implicants.

6 CASE STUDY

6.1 Emergency core cooling system

The example presented in this paper is an emergency core cooling system of a boiling water reactor. This system has been analyzed as a demonstration example for model checking [20]. The purpose of the emergency cooling system is to guarantee adequate water cooling of the reactor core if the ordinary cooling systems are out of order. The cooling system is controlled by electronic control system which regulates the water level in the reactor containment by controlling the pumps and valves. The operational principle is like in a thermostat: when the water level gets too low (due to boiling and evaporation) in the containment, more water is pumped in until the water level reaches the upper limit. This cycle is repeated as long as necessary. The functional principle of the system is illustrated in Figure 4.

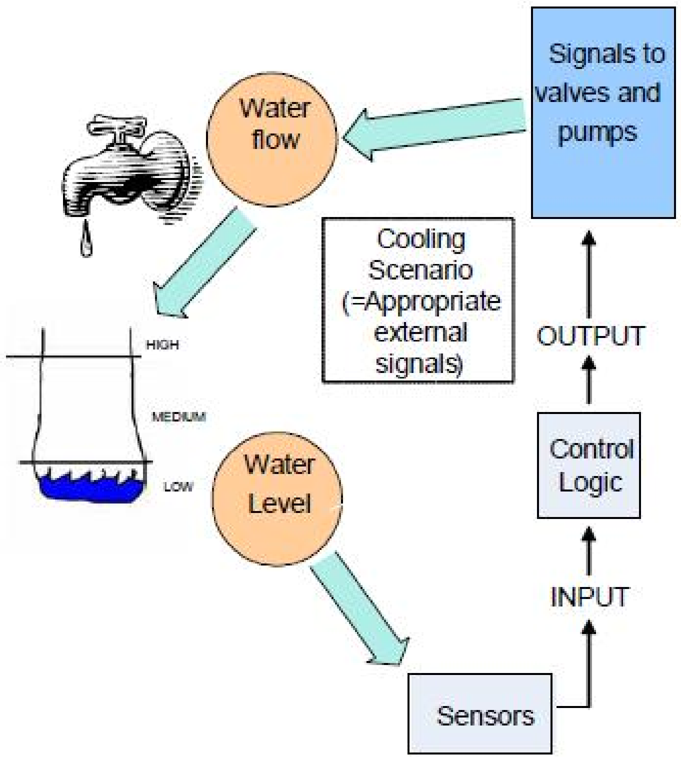
Signals to valves and pumps
Water flow
Cooling Scenario (=Appropriate external signals)
OUTPUT
HIGH
MEDIUM
LOW
Water Level
Control Logic
INPUT
Sensors

Figure 4. Functional principle of the emergency core cooling system [20]

6.2 Results

The following top event was considered: Water level is low on four consecutive time steps. The number of time steps the analysis was backtracked was 6. The analysis was carried on a PC with 1 GB of RAM and Intel Core 2 Duo T7500 processor running at 2.20 GHz. For BDD construction and for the calculation of prime implicants the CUDD package [19] was used. The CUDD package provides its own methods to solve prime implicants. However, the approach utilized by CUDD does not solve all prime implicants, but only the first set of prime implicants that cover the function. As the underlying underlying structure of the BDD is highly dependent of the variable order, the resulting prime implicants may vary with different variable orders.

The system model was analyzed with the Meta-product (MP) and the ZBDD approach. For comparison, also the algorithm provided by CUDD was used to analyze the model. The results of the analyses are summarized in Table 1.

The analysis times of the different approaches differed considerably. As expected, the analysis times with CUDD were much shorter than the analysis times of the two other approaches. The large difference can be mainly explained by that the ZBDD and the MP approach do not yet utilize a computation table (see section 5.1) to store intermediate results, but all recursive computations are performed anew even though the same computation might have already been computed. The ZBDD approach was faster than Meta-product approach. The ZBDD approach allows in most cases to represent the prime implicant BDD with fewer nodes than the MP approach causing partly the shorter computation times.

Since both the ZBDD and MP approach compute all prime implicants, their results were identical. Because the CUDD approach only returns the first set of prime implicants that cover the function the number of computed prime implicants is smaller.

Table 1. Analysis results

Algorithm	Meta-product	ZBDD	CUDD
Number of prime implicants	620	620	460
Computation time	32s	23s	<1s

6.3 Discussion

To test the applicability of YADRAT several test cases have been analyzed. [2] presents some of these test cases and a comparison of two tools (YADRAT and DYMONDA[8]) to solve DFM models. It should be noted that the algorithm used in [2] to solve the prime implicants with YADRAT utilized the approach provided by the CUDD package.

The approach employed by YADRAT seems to work quite well for small to moderate sized system [2]. When computing even more complicated models (larger systems or higher number of steps the analysis is backtracked) with YADRAT, the computation times grow very fast making the analysis of such cases somewhat impractical. YADRAT should scale up much better when analyzing pure fault tree models, because fault tree models contain only Boolean variables and, thus, the time consuming post-processing of prime implicants, caused by the multi-valued logic, can be ignored. However, analysis of pure fault tree models to test scalability has not yet been performed.

The largest system analyzed thus far with the ZBDD algorithm is the feedwater control system described in [13]. In the most complex analysis performed successfully for the system the ZBDD algorithm was able to compute 5802 prime implicants in 840s. When the number of steps the analysis was backtracked was increased by one, YADRAT was no longer able to perform the computation in reasonable time (< 1 day). The most time consuming phases of the analysis are the computation of prime implicants and the post processing of prime implicants. Of these two phases, post processing is generally more time and memory consuming.

The computation times of the ZBDD (and MP) approach can easily be improved by introducing a computation table for the prime implicant BDD calculation. For improving scalability of YADRAT truncation, i.e. neglecting low probability implicants, should also be considered. Algorithms for truncated computations of prime implicants of fault trees are presented, e.g., in [17, 18]. However, these approaches are not directly applicable to YADRAT, since each state is represented as Boolean vector in contrast to the Boolean variables of a fault tree.

To our knowledge, the computational complexity of DFM hasn't been analyzed theoretically. In experiments conducted in [12], some interesting observations were made. There are problem instances of DFM models where the computational effort grows exponentially as a function of problem size. It also turned out that the computational effort is in some cases extremely sensitive to small changes in the model: in one case, changing just one entry in a decision table resulted in the computational complexity changing from a third-degree polynomial to exponential. On the other hand, no meaningful way of expressing computational complexity as a function of the number of variables was found; in DFM, computational complexity depends much more on model topology and decision tables than on the number of variables.

At this stage the variable order is the order in which the nodes are created (depends on how the user created model). Some preliminary studies have been made to test how variable ordering affects the computation times. The preliminary test showed that computation time is very sensitive to the selected variable order. With a bad variable ordering the computation times could be over 20 times longer than with good one. For instance, for the example system the solved with the ZBDD algorithm the computation times varied between 6s and 120s depending on the variable order. Besides the size of the BDD representing the model also the structure of BDD affects the computation times. The effect of different variable orders to the size of the prime implicant BDD has not yet been studied.

The main purpose of YADRAT is not to solve pure fault tree models but to model dynamic systems that the fault tree approach is not able to model accurately. The benefits of DFM in general are that it can capture the time dependent behavior of dynamic systems quite well. YADRAT is able to compute moderate sized systems in reasonable time accurately. The main limitation of YADRAT at the moment is its inability to scale up very well. Additionally, model construction is not very intuitive. In current version of YADRAT a model is written in a Microsoft Excel file. Even though the Excel format has several benefits, the task of building a model is prone to human errors that can be cumbersome to identify and correct.

YADRAT has been implemented in such a way that the DFM model can be translated into a code which can be further examined by the formal model checking tool NuSMV [15]. This facilitates various means to check the properties of the DFM model and the correctness of results.

7 CONCLUSIONS

Dynamic reliability assessment methodologies can provide a more accurate representation of probabilistic system evolution in time than the traditional FT approach. Using the traditional fault tree/event tree methodology it is very difficult to model the time dependent dynamic behaviors of, e.g., digital instrumentation and control systems with required accuracy. The DFM modelling approach is promising for reliability assessment of dynamic system. The main benefits of the approach are the simplicity of its formalism, the possibility to model time dependencies and loop dependencies, the possibility to model multi-state logic and incoherent reliability structures, and separation of the system model from top events (a single system model can be used to analyse different top events). The main drawback and limitation of DFM is that a more realistic modelling easily causes a combinatorial explosion as the number of states in the decision tables grows.

This paper presented an alternative DFM approach based on BDD, implemented in a program called YADRAT. The objective of a YADRAT model analysis is to find the root causes of the query (top event) of interest. The main difference of YADRAT compared to the existing DFM approach is that YADRAT employs BDD to represent a DFM model. Two different approaches to solving a BDD model have been implemented for exact computation of prime implicants.

The algorithms for BDD computation are well studied but scalability is a problem. One way to improve scalability is to consider truncation in the calculations. Also new approaches are being studied, e.g. propositional satisfiability, to support BDD-based algorithms to increase the scalability of YADRAT.

The goal of YADRAT is not to replace the fault tree/event tree approach but to provide a complementary tool to solve dynamic reliability analysis problems. The approach employed by YADRAT is very general and it can be used to model the dynamical and logical behaviour of complex systems. The basic solution of YADRAT seems to work well and the preliminary results of YADRAT are promising but there exist several challenges, such as scalability. There are also central reliability theoretical issues relating to DFM and BDD modelling that should be studied, e.g. risk importance measures and the management of common cause failures.

8 REFERENCES

1. T. Aldemir, M.P. Stovsky, J. Kirschenbaum, D. Mandelli, P. Bucci, L.A. Mangan, D.W. Miller, X. Sun, E. Ekici, S. Guarro, M. Yau, B. Johnson, C. Elks, and S.A. Arnd,. “Dynamic reliability modeling of digital instrumentation and control systems for nuclear reactor probabilistic risk assessments”, *NUREG report NUREG/CR-6942*, October 2007.
2. K. Björkman, J.-E. Holmberg, “Comparison of two dynamic reliability analysis tools to solve dynamic flowgraph method models”, *VTT Research Report, VTT-R-00775-10*, Espoo, 2010.
3. K. Björkman, *YADRAT — Concepts and algorithms*, *VTT Research Report, VTT-R-07658-10*, Espoo, 2010.
4. K.L. Brace, R.L. Rudell, R.E. Bryant, “Efficient Implementation of a BDD Package”, *27th ACM/IEEE Design Automation Conference*, 1990, Paper 3.1, 40-45, 1990
5. R. E. Bryant, “Graph-Based Algorithms for Boolean Function Manipulation”, *IEEE Transactions on Computers* C-35, 6(Aug), pp. 677-691, 1986
6. R. E. Bryant, “Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams”, *ACM Computing Survey,* 24, 3 (Sep. 1992), 293-318, 1992
7. O. Coudert, J.C. Madre, “Implicit and incremental computation of primes and essential primes of Boolean functions”, *Proceedings of the 29th ACM/IEEE design automation conference*, DAC'92 1992.
8. Dymonda software for DFM modelling and analysis http://ascainc.com/dymonda/dymonda.html
9. C.J. Garrett, S.B. Guarro, G.E. Apostolakis, “The Dynamic Flowgraph Methodology for Assessing the Dependability of Embedded Software Systems”, *IEEE Trans. on Systems, Man and Cybernetics* Vol. 25. No. 5, 824-840, 1995.
10. M. Houtermans, G. Apostolakis, A. Brombacher and D. Karydas. “Programmable electronic system design & verification utilizing DFM”, *In SAFECOMP 2000* (F. Koornneef and M. van der Meulen, eds.), Lecture Notes in Computer Science 1943 (2000), 275-285.
11. M. Huth, M. Ryan, *Logic in Computer Science: Modelling and Reasoning about Systems*, 2nd Edition. Cambridge University Press, 2004.

12. I. Karanta and P. Kuusela, ”On the Computational Effort of the Dynamic Flowgraph Methodology,” *VTT Research Report VTT-R-00831-10*, Espoo 2010.

13. I. Karanta, M. Maskuniitty, “Reliability of digital control systems in nuclear power plants — Modelling the feedwater system”, *VTT Research Report VTT-R-01749-08*, VTT, Espoo, January 2009.

14. S. Minato, “Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems”, *In the proceedings of the 30th ACM/IEEE Design Automation Conference*, pp. 272-277, June 1993.

15. NuSMV Model Checker v.2.5.2. http://nusmv.fbk.eu/ 2010.

16. A. Rauzy, “New Algorithms for Fault Trees Analysis”, *Reliability Engineering and System Safety* 40 203-211, 1993

17. A. Rauzy, “Mathematical foundation of minimal cutsets”, *IEEE Transactions on Reliability* 50(4), 389–396, 2001;

18. A. Rauzy, Y. Dutuit, “Exact and truncated computations of prime implicants of coherent and non-coherent fault trees within Aralia”, *Reliability Engineering & System Safety*, Volume 58, Issue 2, Pages 127-144, 1997.

19. F. Somenzi, CUDD: CU Decision Diagram package, Public software, Colorado University, Boulder, 1997.

20. J. Valkonen, V. Pettersson, K. Björkman, J.-E. Holmberg, M. Koskimies, K. Heljanko, I. Niemelä, “Model-Based Analysis of an Arc Protection and an Emergency Cooling System”, *VTT Working Papers 93*, VTT, Espoo. 2008.

21. M. Yau, S. Guarro and G. Apostolakis, “Demonstration of the dynamic flowgraph methodology using the Titan II space launch vehicle digital flight control system”, *Reliability Engineering and System Safety*, Vol. 49, 335-353, 1995.