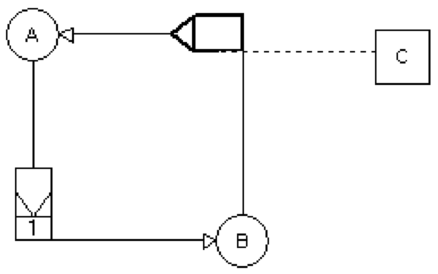
RESEARCH REPORT

VTT-R-00831-10



On the Computational Effort of the Dynamic Flowgraph Methodology

Authors: Ilkka Karanta, Pirkko Kuusela

Confidentiality: Public

Report's title **On the Computational Effort of the Dynamic Flowgraph Methodology**	
Customer, contact person, address VYR	Order reference 27/2007SAF
Project name CHAllenges in Risk-Informed Safety MAnagement (CHARISMA)	Project number/Short name 32503-1.5/CHARISMA-2009
Author(s) Ilkka Karanta, Pirkko Kuusela	Pages 44/
Keywords computational effort, dynamic flowgraph methodology (DFM), digital system reliability	Report identification code VTT-R-00831-10

Summary

Dynamic flowgraph methodology (DFM) is a formalism for modelling and analyzing dynamic systems. It can be used to produce prime implicants and probability estimates for top events in probabilistic safety analysis. This report analyzes the computational effort of analyzing DFM models.

A series of computational experiments have been made on a very simple model subjected to deductive analysis. The experiments consist of altering the model and recording the changes that these alterations cause to the computational burden. It turns out that even slight modifications, such as changing a single entry from false to true in a decision table, can have dramatic effects on the computational burden. On the other hand, some quite regular behaviour was also observed; from this, it is possible to e.g. determine a stopping criterion for increasing the number of time steps in deductive analysis.

A common theme that emerges from the experiments is that it is the branching occurring in decision tables that determines computational effort of a DFM model. The role of the top event is more subsidiary, mainly relating to whether the variables in them have many rows in the decision tables for the given values. Also the computational effort of fault tree analysis is touched upon.

Confidentiality	Public

Espoo 05.05.2010

Written by	Reviewed by	Accepted by
[illegible]	[illegible]	[illegible]
Ilkka Karanta, senior research scientist	Jan-Erik Holmberg chief research scientist	Jari Hämäläinen, Technology manager

VTT's contact address

Ilkka Karanta, VTT, P.O.Box 1000, FI-02044 VTT, Finland

Distribution (customer and VTT)

SAFIR TR8, VTT archive

Contents

1 Introduction

The dynamic flowgraph methodology (DFM) [5] is a method for the risk analysis of discrete-time, discrete-state systems. Its aim is to produce the prime implicants of a system for a given top event, and ultimately, the probability of the event.

A major issue in the use of the model is that with larger models, analysis times grow prohibitively long. Furthermore, in practice, seemingly minor changes in the model may result in dramatic increases in the computation time required.

The aim of this report is to lay some groundwork for the analysis of computational effort of solving DFM models. Some research questions that we try to address are as follows. Is there a way in which the computational effort of a DFM computation task (consisting of a DFM model and a top event applied to it) be analyzed without referring to any specific computational method used for solving the task? Can the complexity of a DFM model be separated from the complexity of a top event applied to that model? How does the top event affect computational effort? How to analyze the sensitivity of computational effort to specific modelling decisions, e.g. increasing the number of possible values of a variable by one? Is there a way to analyze the computational effort inherent in model structure (the way variables are connected together)?

Analysis of the complexity of DFM models, and the effects of modelling decisions on it, has not been conducted thus far to our knowledge.

A word on the terms “computational complexity” and computational effort” is in order. Since the term computational complexity is loaded with connotations to rigorous complexity analysis based on theoretical computer science, and since the approach taken in this report is experimental, we will mainly use the term computational effort henceforth.

This report is organized as follows. First, DFM is introduced in section 2; this section also contains a short description of the computation by which the prime implicants are obtained. Analysis of computational effort from the DFM point of view is considered in section 3. Section 4 describes the computational experiments carried out. Section 5 discusses some ideas from the field of machine learning that could be used in the complexity analysis of DFM models. Section 6 discusses the computational effort of fault tree models.

2 Dynamic flowgraph methodology

The dynamic flowgraph methodology (DFM) is an approach to modeling and analyzing the behaviour of dynamic systems for reliability assessment and verification [5]. DFM models express the logic of the system in terms of causal relationships between physical variables and states of the control systems; 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 and adverse environmental conditions.

DFM has been used to assess the reliability of nuclear power plant control systems [**1**], but also of space rockets [**13**] and chemical batch processes [**7**].

The basic modelling constructs of DFM have been covered elsewhere [5, 9] and will not be repeated here.

2.1 An example model

The model described in this section serves two purposes: first, it illustrates the concepts presented above. Second, it acts as a baseline model for the computational effort considerations in section 4.

The model consists of three variables A, B and C, of which A and B are in a feedback loop and C can be considered to represent a failure event. The DFM graph of the system is as follows:

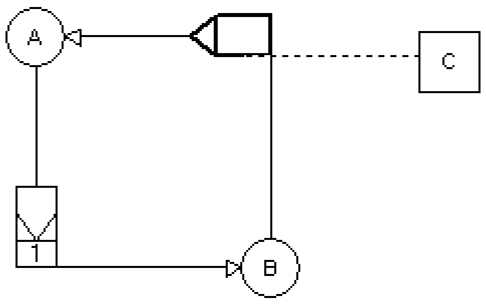


Figure 1. A simple DFM model used as a baseline for comparisons

The variables A and B in the baseline model may be in either of the two states 1 and 2. Variable C may be in either of the two states T and F.

The decision table of the transfer and the transition boxes are

Table 1. The decision table for variable A in the model of Figure 1

B	C	A
1	F	1
1	T	2
2	F	2
2	T	1

Table 2 The decision table for variable B in the model of Figure 1. There is a 1 step delay between A and B.

A	B
1	1
2	2

If C is false (the system works), the value of variable B equals that of variable A, but with a 1 step time lag. If C is true, then A is the inverse of B with a 1 step time lag.

2.2 Analysis of DFM models

After construction, the DFM model can be analyzed in two different modes, deductive and inductive [8]. 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 corresponding to a failure. A top event consists here of variable values at specific time instances. For example, the top event X=low@T=0 & X=low@T=-1 & Y=small@T=0 might describe a situation where the water level in the reactor (X) is too low for two consecutive time instances, and the inflow of water (Y) is still small at the later time.

To find the root causes of the top event, the model is backtracked through the network of nodes, edges, transfer and transition 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 is a conjunction of triplets (V, S, T); each triplet tells that node V is in state S at time T. The circumstances described by the set of triplets causes the top event. Prime implicants are similar to minimal cutsets of fault tree analysis, except that prime implicants are timed and prime implicants deal with multivalued 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 is quantified as in the MCS analysis of a fault tree.

The application scope of DFM is large. The most important current areas of application include

- determination of prime implicants or minimal cutsets for PRA purposes. These can be used to construct timed fault trees.
- inspection of a given design for requirements compliance
- setting up a testing plan by examining what events might lead to failure
- computation of probabilities of top events for PRA/PSA.

The computation of prime implicants in deductive analysis proceeds by starting at the top event. The variable(s) in the top event are traced back in causality and time to find out what triplets might be the immediate reasons for them; then, these triplets are traced back etc. The information discovered at each step of the backtracking process is represented in the form of a series of intermediate transition tables, logically equivalent to gates in a timed fault tree.

The transition table is structured as follows. Each column in the table represents a variable at a given time instance (e.g. A@-1). Each row represents a combination of values for the variables that makes a triplet the top event happen; when deductive analysis has been carried out, each row is a prime implicant. Thus, each entry in the table represents the state of a variable at a given time instance. An entry might also be "don't-care", if the state of the variable@time-instance in the column is not relevant for the prime implicant in the row.

An intermediate transition table is constructed as follows. First, the transition table contains only the top event. On each step of the deductive process, a copy of the decision table is made, and a variable in the decision table is expanded in the way that it is removed from the table, and its input variables (at the proper time instances) are inserted in its place. Also the rows in the decision table of the variable are inserted into the transition table. Then the transition table is simplified to eliminate impossible conditions from the table; for example, a single variable cannot be in two different states at the same time instance.

The expansion of the table goes on until all variable@timeinstance pairs in the table are either for condition nodes (that don't depend on any input variable), or for process variable nodes where timeinstance ≤ start-of-the-deductive-analysis.

A short example clarifies the procedure.

Consider the simple DFM model described in section 2.1. Let the top event be A=2@0, that is, the state of variable A is 2 at time 0. The transition table of the top event is

Table 3. Transition table for the top event A=2@0

t=0	
A	TOP
2	T

The decision table for variable A is represented in Table 1 (page 6). Two rows in the decision table give state 2 for A. Variable A is replace by variables B and C. The resulting transition table is

Table 4. The transition table after the first backtrack

t=0	t=0	
B	C	TOP
1	T	T
2	F	T

When we further backtrack in the model, we see from Table 2 that B exactly equals A after a delay of 1 unit. Thus we replace B=1@0 with A=1@-1 and B=2@0 with A=2@-1, yielding the following transition table:

Table 5. The transition table after the second backtrack

t=-1	t=0	
A	C	TOP
1	T	T
2	F	T

3 Analysis of computational effort

3.1 Traditional methods of computational complexity analysis

In theoretical computer science, quite much literature exists on computational complexity analysis in general (see, e.g., [3]). The standard vehicle of analysis is the Turing machine. The time complexity of a computation performed by a Turing machine is the number of transitions in the computation.

It seems natural to apply this view of computational complexity also to the analysis of DFM. However, doing such a rigorous analysis is a tedious task. On the other hand, typical information obtained from a theoretical complexity analysis – worst-case complexity and average-case complexity – don't usually shed much light on how various modelling decisions affect computational complexity. Therefore, a more light-weight approach, based on experimentation, is adopted in this report.

Although a different approach is taken, it is still useful to consider what elements of complexity analysis could be adapted to the present context. In DFM, a natural parallel to Turing machine transitions is the evaluation of a transition table. Therefore, a natural proxy for time complexity is the number of elements in transition tables generated by the computation. Also the number of resulting prime implicants and the number of triplets in them may shed some light on the computational complexity issue; here, however, it must be noted that the relationship between computational complexity and the number of prime implicants is a loose one at best, if it exists at all.

3.2 Special features of the DFM complexity analysis problem

When analyzing the computational effort of a DFM computation task (a system model and a related top event), the interest is in the counting of possible alternatives that must be inspected.

3.3 Experimental complexity analysis

3.3.1 Analyzing a very simple model empirically

This method is empirical, based on computer experiments. The idea is to formulate a very simple model – baseline model – and a simple top event. These are then enhanced in a number of ways. For each enhancement as well as the baseline model, the following key figures are recorded:

- computation time (if different from 0)
- number of time steps
- the number of intermediate steps in the computation, and the sizes of transition tables in each phase

- the number of resulting prime implicants, and the minimum and maximum number of variables involved in them.

The baseline model is described in section 2.1. The enhancements to the basic model are as follows:

- Changing the number of states in variable nodes from 2 to 3 (section 4.2)
- Increasing the number of variables by adding a new condition node (section 4.3)
- Increasing the number of variables by adding a new variable node (section 4.4)
- Replacing a condition node with a dependent condition node (section 4.5)
- Adding a dependent condition node without deleting the existing condition node
- Increasing the complexity of the top event by adding a new triplet
- Increasing the time lag in the transition box 1→2→3.

Each model is analyzed for four different number of time steps (1, 2, 3, 4). The top event in each case (except the last one) is A=2@T=0 & A=2@T=-1.

3.3.2 Analyzing more realistic models with computational experiments

This is also an empirical method of analysis. The idea is to select a reasonably realistic model, augment it with various extensions - e.g. corresponding to those listed in section 3.3.1 - and see how computation time, number of prime implicants etc. are affected.

Experiments can be conducted by the same principles as for the simple baseline model – that is, add new process nodes and condition nodes, increase the number of states of a node, tweak the decision table of a transition or transfer box etc. Thus, it could be verified whether the findings with the simple model carry on to more realistic settings. Another benefit is that with larger models, the differences in computation time are more substantial and can be measured.

3.3.2.1 Potential models

The report [4] lists four models that have been used in the comparison of two DFM programs, DYMONDA and YADRAT. These models handle tank water level control, emergency water cooling system, filling of a reactor tank, and a BWR feedwater control system. In addition, this section lists some models described in DFM literature. Using them has the advantage that results obtained can in some cases be compared with results in the literature. A DFM model has been proposed for each of them, although the level of detail in model description varies.

3.3.2.1.1 Pressure tank model

This model is used in [12], section 3.5. It is a slight modification of the pressure tank example used in Chapter VIII of [11]. It is illustrated in Figure 2.

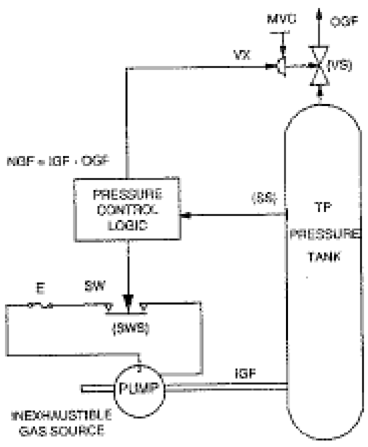


Figure 2. A simple pressure tank model

The model consists of a pressure tank and its control equipment and logic. If the pressure in the tank falls below a certain limit, more is pumped into it; if the pressure in the tank rises above a certain limit, gas is let out through a valve.

3.3.2.1.2 Digital feedwater control system of a generic pressurized water reactor

This is the benchmark system used in [2] for the comparison of Markov/CCMT and DFM in probabilistic risk assessment. The purpose of the digital feedwater control system is to maintain steam generator water level within designated limits from an assigned setpoint. The model is illustrated in Figure 3.

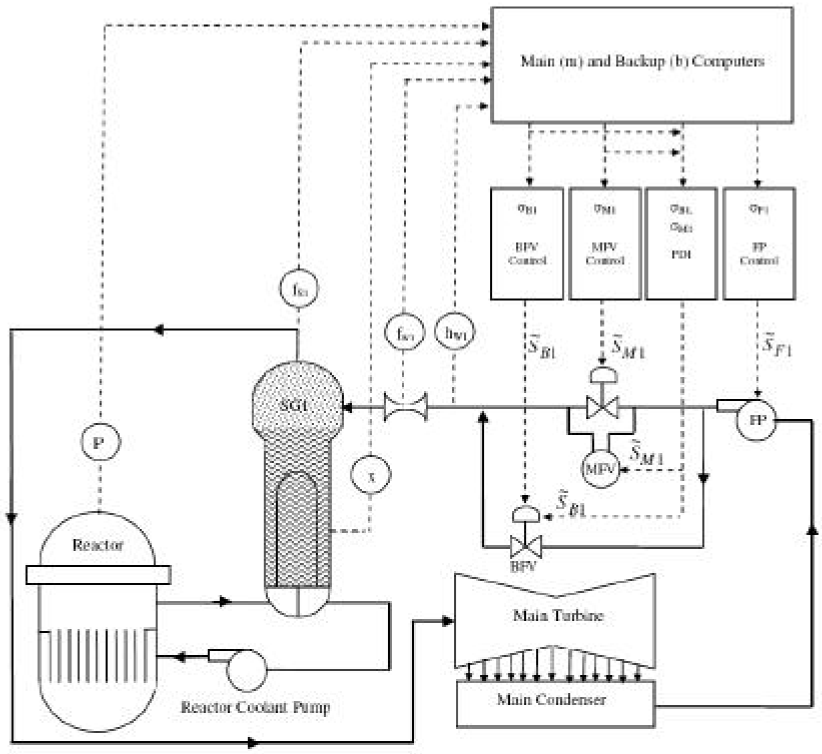


Figure 3. Digital feedwater control system of a generic pressurized water reactor

4 Computational experiments

This section describes a number of experiments made on a simple DFM model that is modified to reveal the effects of different modelling decisions. For each experiment, its setup and results are described, and some conclusions are drawn.

4.1 The baseline model

This is the model described in section 2.1. The model was analyzed with respect to the number of time steps in the analysis. The purpose of the experiment is to gain insight on how computational effort behaves when the number of time steps is increased.
Additionally, the following questions concerning prime implicants are analyzed. How does the number of prime implicants behave when the number of time steps is increased? Is there any pattern (e.g. repetition) observable in the prime implicants as the number of time steps is increased?

4.1.1 Experiment setup

The baseline model was used as such. The number of time steps was increased from 1 to 4.

4.1.2 Results

The model was analyzed for the size of the associated tables and the complexity of the resulting prime implicants as a function of the number of time steps. The results are given in

Table 6. Complexity of the baseline model as a function of time steps

N = # time steps	Sizes of transition tables at each intermediary stage (# rows, # columns)	Total # elements in trans. tables	# prime implicants	Minimum and maximum number of variables in the prime implicants
1	1x2, 2x3, 1x2(1x3), 2x3	17	2	3, 3
2	1x2, 2x3, 1x2(1x3), 2x3, 4x4	33	4	4, 4
3	1x2, 2x3, 1x2(1x3), 2x3, 4x4, 8x5	73	8	5, 5
4	1x2, 2x3, 1x2(1x3), 2x3, 4x4, 8x5, 16x6	169	16	6, 6

As seen from Table 6, the number of prime implicants grows exponentially as a function of time steps. Another observation is that the number of triplets in each prime implicant – denote it by $V_1(n)$ – grows as n+2.

The prime implicants are listed in

Table 7. The prime implicants of the baseline model at different time steps

# time steps	Prime implicants

1	B=1@-1 & C=T@-1 & C=F@0
	B=2@-1 & C=F@-1 & C=F@0

2	B=1@-2 & C=F@-2 & C=T@-1 & C=F@0
	B=1@-2 & C=T@-2 & C=F@-1 & C=F@0
	B=2@-2 & C=F@-2 & C=F@-1 & C=F@0
	B=2@-2 & C=T@-2 & C=T@-1 & C=F@0

3	B=1@-3 & C=F@-3 & C=F@-2 & C=T@-1 & C=F@0
	B=1@-3 & C=F@-3 & C=T@-2 & C=F@-1 & C=F@0
	B=1@-3 & C=T@-3 & C=F@-2 & C=F@-1 & C=F@0
	B=1@-3 & C=T@-3 & C=T@-2 & C=T@-1 & C=F@0
	B=2@-3 & C=F@-3 & C=F@-2 & C=F@-1 & C=F@0
	B=2@-3 & C=F@-3 & C=T@-2 & C=T@-1 & C=F@0
	B=2@-3 & C=T@-3 & C=F@-2 & C=T@-1 & C=F@0
	B=2@-3 & C=T@-3 & C=T@-2 & C=F@-1 & C=F@0

4	B=1@-4 & C=F@-4 & C=F@-3 & C=F@-2 & C=T@-1 & C=F@0
	B=1@-4 & C=F@-4 & C=F@-3 & C=T@-2 & C=F@-1 & C=F@0
	B=1@-4 & C=F@-4 & C=T@-3 & C=F@-2 & C=F@-1 & C=F@0
	B=1@-4 & C=F@-4 & C=T@-3 & C=T@-2 & C=T@-1 & C=F@0
	B=1@-4 & C=T@-4 & C=F@-3 & C=F@-2 & C=F@-1 & C=F@0
	B=1@-4 & C=T@-4 & C=F@-3 & C=T@-2 & C=T@-1 & C=F@0
	B=1@-4 & C=T@-4 & C=T@-3 & C=F@-2 & C=T@-1 & C=F@0
	B=1@-4 & C=T@-4 & C=T@-3 & C=T@-2 & C=F@-1 & C=F@0
	B=2@-4 & C=F@-4 & C=F@-3 & C=F@-2 & C=F@-1 & C=F@0
	B=2@-4 & C=F@-4 & C=F@-3 & C=T@-2 & C=T@-1 & C=F@0
	B=2@-4 & C=F@-4 & C=T@-3 & C=F@-2 & C=T@-1 & C=F@0
	B=2@-4 & C=F@-4 & C=T@-3 & C=T@-2 & C=F@-1 & C=F@0
	B=2@-4 & C=T@-4 & C=F@-3 & C=F@-2 & C=T@-1 & C=F@0
	B=2@-4 & C=T@-4 & C=F@-3 & C=T@-2 & C=F@-1 & C=F@0
	B=2@-4 & C=T@-4 & C=T@-3 & C=F@-2 & C=F@-1 & C=F@0
	B=2@-4 & C=T@-4 & C=T@-3 & C=T@-2 & C=T@-1 & C=F@0

It is easy to see that, although the number of prime implicants increases exponentially, they follow a simple pattern: either B=1 initially, and C=T for an odd number of times since the start of the analysis, or B=2 initially and C=T an even number of times since, and C=F@0.

However, it is more important what growth pattern the number of elements in transition tables — a better proxy for computational effort than the number of prime implicants — takes. It is easy to see from Table 6 that in this model, there is a regularity in the transition table size sequences. Denote the number total number of elements in transition tables for solving the model with N time steps by X_N. For the number of time steps N+1, the sequence is the same as for N time steps except that a table with size $2^N(N+2)$ is appended to the end. Assuming that the pattern holds, we get the formula

$$X_{N+1} = X_N + 2^N(N+2) \tag{1}$$

with X_0=11. For example, it can be predicted that the total number of elements in transition tables for N = 5 equals 169 + $2^{5+1}(5 + 3)$ = 169 + 256 = 425 elements. As seen from (1), the increment of the number of elements grows exponentially. Thus we may tentatively conclude that X_N grows exponentially as a function of N.

4.1.3 Conclusions from the experiment

The observation that the number of prime implicants increases exponentially as a function of the number of time steps is easy to explain. Each value of variable A can be arrived in two ways: either its value was the same in the previous time step, or its value was off by one and C=T at the previous time step.

The more important observation that the total number of elements in decision tables seems to increase also exponentially is more difficult to explain. However, if the observation holds more generally, it has important consequences to the complexity analysis of DFM. Validation (or otherwise) of this hypothesis must wait the rigorous analysis of the DFM deductive analysis algorithm.

The experiment brings about the following hypothesis: all distinct prime implicants occur in a finite number of time steps. Here, two prime implicants are distinct if they don’t follow a common pattern recognizable by e.g. a regular expression. This number depends on the topology of the DFM graph, and the decision tables and time steps involved. If the number of time steps is larger than this, only prime implicants equivalent to those obtained by a smaller number of time steps emerge.

If the hypothesis is true (as it seems), it has profound effect on determining the number of time steps needed to find all relevant prime implicants. Let the number of prime implicants obtained by running the analysis n steps be Pn. We can tentatively formulate a candidate for a stopping criterion in increasing the number of time steps in DFM:

Increasing the number of time steps can be stopped if either of the following two conditions are met:

1. the difference in the number of prime implicants between two consecutive analysis runs (with # of time steps increased by 1) reaches a constant:

$$P_n - P_{n-1} = P_{n+1} - P_n$$

2. the ratio of the number of prime implicants between two consecutive analysis runs reaches a constant:

$$P_{n+1}/P_n = P_n/P_{n-1}$$

An alternative way to choose the maximum number of analysis steps needed is to analyze the model and the top event. A useful proxy for the number of time steps needed could be the maximum of the delays of the feedback loops of the model (here the delay of a feedback loop is the sum of delays in it).

4.2 Varying the number of states in variables, experiment 1

The baseline model is compared with a model where the number of states in the continuous variables has been increased from 2 to 3. The purpose of this experiment is to gain insight into how computational effort increases when the number of states in a variable is increased.

It is easy to see that for this model, the prime implicants are produced by the following three patterns:

- B=1 initially, and C=T for 3n+1 times in the analysis interval
- B=2 initially, and C=T for 3n times in the analysis interval
- B=3 initially, and C=T for 3n+2 times in the analysis interval

4.2.1 Experiment setup

The idea here is to increase the number of the continuous variables A and B in the baseline model from 2 to 3. In increasing the number of variables, the design decision remaining is how to set up the decision tables. Here, to preserve symmetry, the decision table is altered so that when C=F, A equals B with a 1 step time lag; if C=T, A is rotated so that B=1⇒A=2, B=2⇒A=3, and B=3⇒A=1.

4.2.2 Results

The complexity results are exactly the same as with the baseline model (see Table 6). Also the number of prime implicants is the same as in the baseline model.

This is notable in the light that the prime implicants are different from those of the base model (see above).

4.2.3 Conclusions from the experiment

The experiment shows that it is possible to increase the number of states in the variables without affecting computational effort in any way. On a closer inspection, this isn't surprising: because branching in decision tables hasn't been affected in any way: in the decision table of A, there are still exactly two ways to arrive in a given value for A.

4.3 Varying the number of states in variables, experiment 2

The purpose of this experiment, compared with the previous experiment, is to see how branching in decision tables affects computational effort. In the previous experiment, there was no branching. Here, several rows in the decision table of A lead to state 2.

4.3.1 Experiment setup

The basic idea here is the same as in the previous experiment 4.2: to increase the number of states in process variables. However, here a different decision table is constructed for node A: if C=F, then A=B; if C=T, then A=2.

4.3.2 Results

Table 8. Computational effort of the model of section 2.1 with modifications explained in section 4.3.1 as a function of time steps

N = # time steps	Sizes of transition tables at each intermediary stage (# rows, # columns)	# prime implicants	Minimum and maximum number of variables in the prime implicants
1	1x2, 2x3, 1x1(2x3), 2x2	2	1, 1
2	1x2, 2x3, 1x1(2x3), 2x2, 2x2, 3x3	3	1, 1
3	1x2, 2x3, 1x1(2x3), 2x2, 2x2, 3x3, 3x3, 4x4	4	1, 1
4	1x2, 2x3, 1x1(2x3), 2x2, 2x2, 3x3, 3x3, 4x4, 4x4, 5x5	5	1, 1

The number of prime implicants grows linearly, whereas the number of triplets in each prime implicants stays the same.

Here, again, it is more interesting to consider the number of elements in transition tables. It is easily seen from Table 8 that the total number of elements in transition tables, X_N, is

$$X_{N+1} = X_N + (N+1)^2 + (N+2)^2 \qquad (2)$$

Summing, we get the general formula

$$X_N = X_0 + \sum_{k=1}^{N} \left(k^2 + (k+1)^2\right) = X_0 + \frac{\left(N(N+1)(N+2) + (N+1)(N+2)(2N+3) - 6\right)}{6} \qquad (3)$$

The latter formula is obtained by a straightforward application of the sum of squares formula (see, e.g., [14], p. 33). Thus we see that X_N is a third degree polynomial with respect to the number of time steps.

4.3.3 Conclusions from the experiment

The most remarkable feature of these results is that here (with branching in the decision table of A), effort wrt. the number of time steps is polynomial, whereas in the previous experiment, exponential growth was obtained. Thus, the present model is computationally less demanding than the baseline model of section 2.1, even though the number of states is larger.

Another salient feature of the experiment is that the number of triplets needed for each prime implicant stays constant. This might provide a key to understanding why effort wrt. the number of time steps is linear.

A caveat of these conclusions is that they hold only if the number of elements in the transition tables follows the pattern present in Table 8.

4.4 Increasing the number of variables by adding a new condition node

The baseline model is compared with a model to which a new condition node has been added. This experiment aims at producing information about how new fault modes affect DFM model complexity: a condition node can be equated with a fault mode, either a single failure (if the condition node is connected to one variable node) or a dependent failure (if the condition node is connected to two or more variable nodes).

4.4.1 Experiment setup

A new fault variable D, identical to C, was added to the input of B; the decision table for B was made symmetrically like that of A (see Table 1). The model looks as follows:

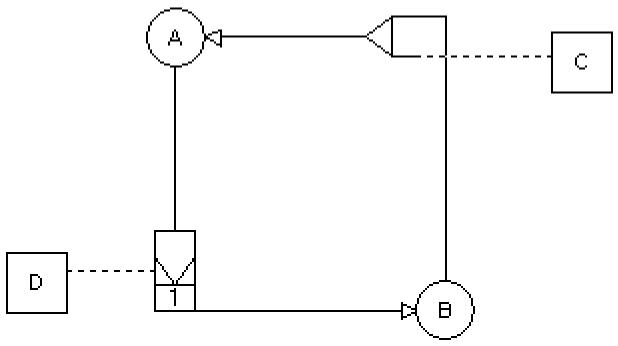


Figure 4. The simple feedback model of section 2.1 with a new condition node D

The decision table of B was made symmetrical to that of A in the baseline model (see section 2.1).

4.4.2 Results

Table 9. Computational effort of the model of section 4.4.1 as a function of time steps

n = # time steps	Sizes of transition tables at each intermediary stage (# rows, # columns)	# prime implicants	Minimum and maximum number of variables in the prime implicants
1	1x2, 2x3, 2x4(2x3), 4 x 4	4	4, 4
2	1x2, 2x3, 2x4(2x3), 4x4, 8x5, 16x6	16	6, 6
3	1x2, 2x3, 2x4(2x3), 4x4, 8x5, 16x6 , 32x7, 64 x 8	64	8, 8
4	1x2, 2x3, 2x4(2x3), 4x4, 8x5, 16x6 , 32x7, 64x8, 128x9, 256x10	256	10, 10

It is easy to see that the number of prime implicants, $V_3(n)$, equals 4^n.

When the numbers of prime implicants are compared with those of the baseline model, a clear pattern emerges:

Table 10. Numbers of prime implicants produced by model of this section and model 2.1 compared

Number of time steps	1	2	3	4
P_3=prime implicants of this model	4	16	64	256
P_1=prime implicants of model 2.1	2	4	8	16
P_3/ P_1	2	4	8	16

Thus, P_3 is the square of P_1.

Another comparison can be made between the numbers of triplets in the prime implicants produced by the two models. This can be compared with the ratio

$$\frac{V_3(n)}{V_1(n)} = \frac{2n+2}{n+2} \xrightarrow{n\to\infty} 2 \qquad (4)$$

The sizes of transition tables show also a pattern. Namely. the total number of elements in the transition tables with N number of steps, X_N, is now

$$X_N = X_{N-1} + 2^{2N-1}(2N+1) + 2^{2N}(2N+2) \qquad (5)$$

and, solving for X_N,

$$X_N = X_0 + \sum_{j=1}^{2N} 2^j (j+2) \qquad (7)$$

so that X_N grows faster than exponentially[1] with regard to the number of time steps. It is seen from Table 9 that X_N = 2 + 6 = 8.

4.4.3 Conclusions from the experiment

Adding a new condition node to the model has a dramatic effect on computational effort in this case. It seems that the number of elements in transition tables grows faster than exponentially, and that the number of prime implicants produced by two condition nodes equals the product of the numbers of prime implicants produced by each condition node.

Another conclusion is that, in this case, the number of variables in the prime implicants produced by the condition nodes equals the sum of the numbers of the variables produced by each condition node.

[1] Finding a closed-form expression for the sum in (7) is an unsolved problem (see http://www.research.att.com/~njas/sequences/index.html, entry A036799), but it is easy to see that the dominating term in the sum grows faster than exponentially.

So far, generalizations of these two conclusions are just hypotheses. More experimentation and theoretical analysis would be needed to verify (or refute) the generalizations.

Adding a new condition node to a DFM model has an analogy in traditional reliability analysis. Each condition node adds a new possibility for the system to fail, that is, a new set of prime implicants to minimal cutsets. If two condition nodes are independent, the number of prime implicants produced by them should be the product of the numbers produced by each.

4.5 Increasing the number of variables by adding a new variable node and a condition node

The purpose of this experiment is to highlight how adding variables to a feedback loop affects computational effort.

4.5.1 Experiment setup

A new variable node, D, was introduced to the model 2.1. Since it isn't the number of variables per se that increases computational complexity but rather the branching occurring in the decision tables corresponding to those variables, also a new condition node was added to the model. The resulting model looks as follows:

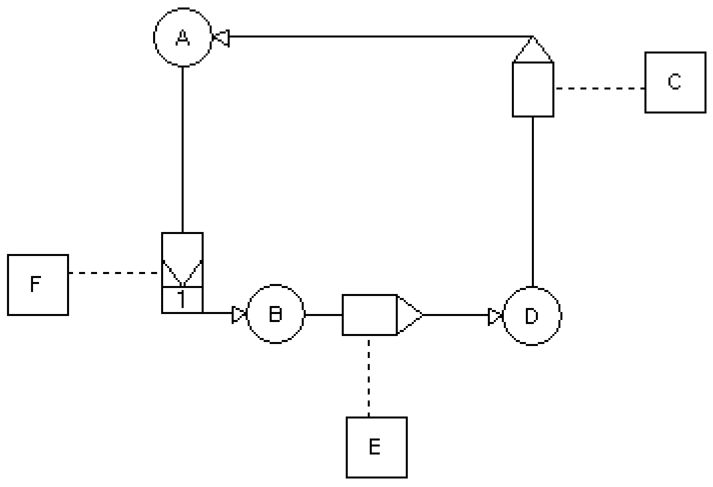


Figure 5. The simple feedback model with a new variable node D and condition nodes E and F

4.5.2 Results

Table 11. Complexity of the model of section 4.5.1 as a function of time steps

n = # time steps	Sizes of transition tables at each intermediary stage (# rows, # columns)	# prime implicants	Minimum and maximum number of variables in the prime implicants
1	1x2, 2x3, 4x4, 4x4(4x5), 8x5, 16x6	16	6, 6
2	1x2, 2x3, 4x4, 4x4(4x5), 8x5, 16x6, 32x7, 64x8, 128x9	128	9, 9
	Computation was interrupted		

Here it seems that the number of prime implicants follows a pattern 2^{3n+1}.

It is premature to predict the number of elements in transition tables, X_N, based on just two time steps. However, one can hypothesize from Table 11 that

$$X_N = X_{N-1} + 2^{3N-1}(3N+1) + 2^{3N}(3N+2) + 2^{3N+1}(3N+3) \tag{8}$$

From this X_N can be solved to be

$$X_N = X_0 + \sum_{j=1}^{3N} 2^{j+1}(j+3) \tag{9}$$

Here $X_0 = 2 + 6 + 16 = 24$.

4.5.3 Conclusions from the experiment

This experiment supports the hypothesis put forth in section 4.4.2 that the number of prime implicants is the product of the complexities induced by individual condition nodes.

If the hypothesis put forth on the number of transition table elements is true, computational effort (as measured by the number of transition table elements) grows faster than exponentially.

4.6 Increasing the number of variables by adding a new variable node in parallel

The experiment aims at producing information on how adding a new variable parallel to an existing variable affects computational effort. In addition, the effect of slightly varying the decision table is analyzed.

Since the new node is added in parallel, it is natural to compare the results to those of section 4.5: there the new node was added in series.

4.6.1 Experiment setup

A new variable was inserted to the model in parallel of the variable B. The topology of the model is now

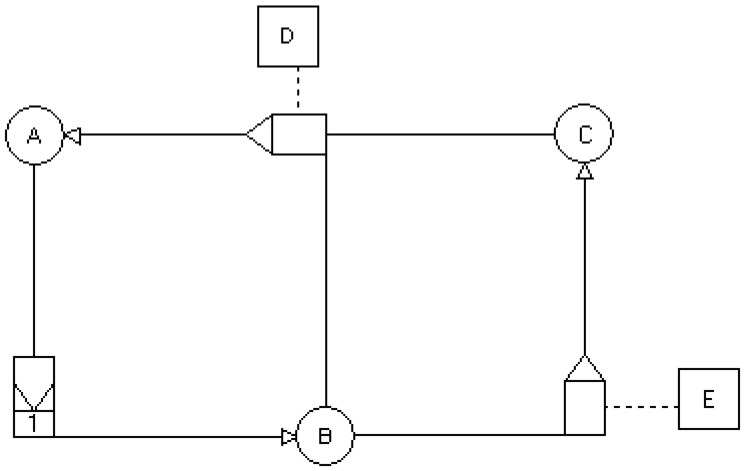


Figure 6. The simple feedback model with a new variable node C in parallel with B, and condition node E

The decision table of variable C equals that of variable A in the baseline case (Table 1), with inputs B and E instead of B and C. The decision table of variable A is subject to change because of a new input variable. Its decision table is now

Table 12. The decision table for variable A in the model of Figure 6

B	C	D	A
1	F	1	1
1	F	2	2
1	T	1	2
1	T	2	1
2	F	1	2
2	F	2	1
2	T	1	1
2	T	2	2

The idea in the decision table is that whenever C=F, A = (B xor D) (that is, the exclusive or). When C=T, A = $\neg$(B xor D). This means rather perfect symmetry: the effect of C is to reverse the truth value that A would otherwise get.

A variant of the present model is one in which only the decision table for A is changed to the following form, breaking the symmetry between variables B and D:

Table 13. An alternative decision table for variable A in the model of Figure 6

B	C	D	A
1	F	1	1
1	F	2	2
1	T	1	2
1	T	2	2
2	F	1	2
2	F	2	1
2	T	1	1
2	T	2	2

The only difference here is that in the fourth line, 1 has been changed to 2. This will have dramatic effects on the number of prime implicants and computational effort, as we shall see.

4.6.2 Results

Table 14. Computational effort of the model of section 4.6.1 as a function of time steps, with the symmetric decision table for A

n = # time steps	Sizes of transition tables at each intermediary stage (# rows, # columns)	# prime implicants	Minimum and maximum number of variables in the prime implicants
1	1x2, 4x4, 4x4(4x5), 2x3(2x4), 8x5, 8x5(8x6), 4x4	4	4, 4
2	1x2, 4x4, 4x4(4x5), 2x3(2x4), 8x5, 8x5(8x6), 4x4(8x5), 4x4	4	4, 4
3	1x2, 4x4, 4x4(4x5), 2x3(2x4), 8x5, 8x5(8x6), 4x4(8x5), 4x4	4	4, 4
4	1x2, 4x4, 4x4(4x5), 2x3(2x4), 8x5, 8x5(8x6), 4x4(8x5), 4x4	4	4, 4

Computational effort (as measured by the number of elements in transition tables) stays constant.

Table 15. Computational effort of the model of section 4.6.1 as a function of time steps, with the nonsymmetric decision table for A

n = # time steps	Sizes of transition tables at each intermediary stage (# rows, # columns)	# prime implicants	Minimum and maximum number of variables in the prime implicants
1	1x2, 4x4, 4x4(5x5), 1x3(2x4), 8x5, 8x5(10x5)	8	4, 4
2	1x2, 4x4, 4x4(5x5), 1x3(2x4), 8x5, 8x5(10x6), 8x5, 16x7, 16x7(16x8), 12x7	12	4, 6
3	1x2, 4x4, 4x4(5x5), 1x3(2x4), 8x5, 8x5(10x6), 8x5, 16x7, 16x7(16x8), 12x7, 24x9, 24x9(24x10)	24	4, 7
4	1x2, 4x4, 4x4(5x5), 1x3(2x4), 8x5, 8x5(10x6), 8x5, 16x7, 16x7(16x8), 12x7, 24x9, 24x9(24x10), 24x9, 40x11, 40x11(40x12), 32x11	32	4, 9

It turns out that a slight breaking up of the symmetry in the decision table has dramatic effects on computational effort and the number of prime implicants. Also the number of prime implicants, and the maximum number of variables in them, doesn't grow steadily.

Finding an analytic expression for the total number of elements in transition tables in Table 15 is not an easy task. However, it is easy to see that the number of elements grows very rapidly as a function of the number of time steps: the X_N for N=1,…,4 are 117, 465, 897, and 2345, respectively.

4.6.3 Conclusions from the experiment

It is surprising the perfectly symmetric decision table helps produce a situation in which the number of prime implicants, or the effort of computing them, are constant with respect to the number of time steps. It is as of yet unclear why this is so. It might be that the deductive analysis algorithm of DYMONDA makes use of the symmetry in the decision table.

Even more surprising is the dramatic effect that changing one 2 to 1 in the decision Table 12 has on the computational effort. Further analysis is needed to find out why such an effect takes place.

An initial guess for the reasons lies in the structure of the decision table. In the perfectly symmetric decision table, no branching can occur; when the symmetry is broken, branching takes place.

When compared with the results in section 4.5, it would seem that increasing the length of a feedback loop will contribute more to computational effort than just adding a new (parallel) variable to it.

4.7 Adding a dependent failure variable to the model

In this experiment, the baseline model is appended by a condition node that connects to both process variable nodes.

4.7.1 Experiment setup

A new fault variable D was added to the model and connected to both A and B. The effect of D is exactly as that of C, except that it affects both continuous nodes at the same time. The model looks as follows:

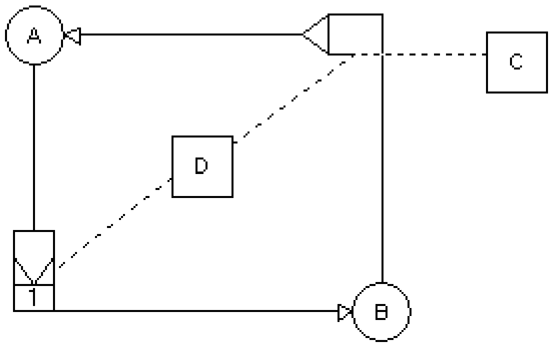


Figure 7. The simple feedback model of section 2.1 with a new dependent fault variable D

The logic in the decision table for A is that if either C or D is true, then A is inverted. In the decision table for B, D inverts A if true.

Table 16. Decision table of A for the model in Figure 7.

B	C	D	A
1	F	F	1
1	F	T	2
1	T	F	2
1	T	T	2
2	F	F	2
2	F	T	1
2	T	F	1
2	T	T	1

Table 17. Decision table of B for the model inFigure 7. There is a 1 step delay between the inputs and B.

A	D	B
1	F	1
2	F	2
1	T	2
2	T	1

4.7.2 Results

Table 18. Computational effort of the model of section 4.7.1 as a function of time steps

n = # time steps	Sizes of transition tables at each intermediary stage (# rows, # columns)	# prime implicants	Minimum and maximum number of variables in the prime implicants
1	1x2, 3x4, 4x4(4x5), 4x5(6x5)	4	3, 5
2	1x2, 3x4, 4x4(4x5), 3x4(3x5), 4x5(6x6)	12	4, 7
3	1x2, 3x4, 3x4(3x5), 4x5(6x6), 8x6, 12x7(16x8), 16x8, 24x8, 36x9(48x10)	36	5, 9
4	largest 108x11	108	6, 11

It turns out that the number of prime implicants with n time steps is three times the number of prime implicants with n-1 time steps.

The number of elements in decision tables follows a complex pattern. It is remarkable here that, whereas previously the sequence of sizes of transition tables, X_N, has followed the pattern that each sequence (with N time steps) is a prefix to the sequence with N+1 time steps, here the pattern breaks down. Therefore it is more difficult to analyze X_N here.

However, it is easy to see that the number of elements grows very rapidly as a function of the number of time steps: the X_N for N = 1, 2, 3 are 50, 62 and 822, respectively.

4.7.3 Conclusions from the experiment

The number of prime implicants increases exponentially as the function of time steps. However, the basis of the exponent is 3. This shows remarkable symmetry: when there was only one condition node (section 4.1), the basis of the exponent was 2, and when there were two independent condition nodes (section 4.4), the basis was 4.

The number of elements in transition tables seems to grow very rapidly.

4.8 Replacing the condition node with a dependent node

4.8.1 Experiment setup

In this model, the discrete variable C is the input of both A and B. It also has the same kind of effect on both variables as it had on A in the baseline model.

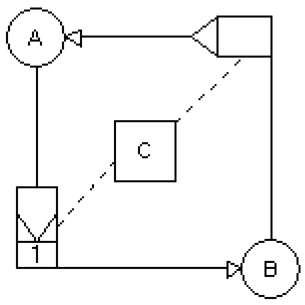


Figure 8. The simple feedback model of section 2.1 with the discrete variable C being a dependent fault variable

4.8.2 Results

Table 19. Computational effort of the model of section 4.8.1 as a function of time steps

n = # time steps	Sizes of transition tables at each intermediary stage (# rows, # columns)	# elements in transition tables	# prime implicants	Minimum and maximum number of variables in the prime implicants
1	2x1, 2x3, 2x3(2x4), 2x3(2x4)	24	2	3, 3
2	2x1, 2x3, 2x3(2x4), 2x3(2x4), 4x4, 4x4(4x5), 2x3	66	2	3, 3
3	2x1, 2x3, 2x3(2x4), 2x3(2x4), 4x4, 4x4(4x5), 4x4(8x5), 4x4(4x5), 2x3	126	2	3, 3
4	2x1, 2x3, 2x3(2x4), 2x3(2x4), 4x4, 4x4(4x5), 4x4(8x5), 4x4(4x5), 4x4(8x5), 4x4(4x5), 2x3	186	2	3, 3

The same two prime implicants occur no matter what the number of time steps.

There are too few steps taken here to analyze the growth in the number of elements in transition tables X_N. However, it seems that after an initial transient $X_{N+1} = X_N + 60$. Such a linear growth would be the simplest growth pattern observed so far (except in Table 14 where the number of elements remained constant after an initial transient).

4.8.3 Conclusions from the experiment

The number of prime implicants stays constant as a function of time steps. However, the number of operations to yield these prime implicants, as measured by the number and size of transition tables, grows. After an initial transient, this growth seems to be linear.

4.9 Adding a new triplet to the top event

4.9.1 Experiment setup

The new triplet was chosen to be A=2@-2. Thus the top event is now A=2@T=0 & A=2@T=-1 & A=2@T=-2.

4.9.2 Results

Table 20. Computational effort of the model as a function of time steps

n = # time steps	Sizes of transition tables at each intermediary stage (# rows, # columns)	# prime implicants	Minimum and maximum number of variables in the prime implicants
1	The number of time steps is too small for the top event		
2	1x3, 2x4, 1x3(1x4), 2x4, 1x3(1x4), 2x4	2	4, 4
3	1x3, 2x4, 1x3(1x4), 2x4, 1x3(1x4), 2x4, 2x4, 4x5	4	5, 5
4	1x3, 2x4, 1x3(1x4), 2x4, 1x3(1x4), 2x4, 2x4, 4x5, 4x5, 8x6	8	6, 6

4.9.3 Conclusions from the experiment

The complexity behaviour in this experiment was essentially similar to the baseline model with the original top event. This seems to indicate that the top event doesn’t have as much impact on computational effort as the model. However, more experimentation is needed to affirm this hypothesis.

4.10 Increasing the time lag in the transition box

4.10.1 Experiment setup

The time lag in the transition box of the baseline model was increased 1→2→3.

4.10.2 Results

Table 21. Computational effort of the model as a function of time steps and the lag in the transition box

n = # time steps	Length of lag in transition box	Sizes of transition tables at each intermediary stage (# rows, # columns)	# prime implicants	Minimum and maximum number of variables in the prime implicants
1	2	1x2, 2x3, 4x4	4	4, 4
1	3	1x2, 2x3, 4x4	4	4, 4
1	4	1x2, 2x3, 4x4	4	4, 4
2	2	1x2, 2x3, 4x4, 8x5	8	5, 5

Increasing the lag beyond 2 didn’t have any effect on the prime implicants in the first three cases. Indeed, the same prime implicants were obtained in each case.

4.10.3 Conclusions from the experiment

When compared to the baseline (section 4.1), the results for the first three cases seem intuitive. With the lag being 2 or more and the number of time steps only 1, variable A doesn’t have any influence on itself. Thus it is the initial values of B and C for the two time instances that determine the values of A for those time instances.

When the number of time steps is increased (and the lag is kept at 2), the familiar exponential growth in the number of prime implicants ensues.

5 Other computational effort considerations

5.1 Influence of choosing the top event on deductive effort

We illustrate how branching in the deductive analysis may depend on the chosen top event. By branching we mean here the number of preimages, or solutions, for a given variable value. For example, if the value of variable X depends on variables Y and Z, we are interested in the number of value combinations of Y and Z that result to some given value of X. We call this branching, and each value combination is a preimage. Decision tables describe how values of a given variable depend on the values of other variables and they determine the amount of branching as deductive analysis proceeds.

Case 1: Normal events occur in many ways

Here we analyze a setting in which normal events occur is many ways, alarming events in a few ways and failures are rare. Assume that all variables are discretized into three values: normal, alarm, and failure. The top variable depends on k new variables, where each variable in turn again depend on k other new variables and so on. See illustration in Figure 9.

We assume that each variable appears only once in this chain. As we are interested in the ratios in which the three possible values occur, we define for variable X, R(X=normal)=rn, R(X=alarm) = ra, R(X=failure)=rf, where rn+ra+rf=1. Thus the above ratios describe how the decision table for X looks like, for the purposes of deductive analysis. Note that there is no randomness in this definition, but these ratios behave like probabilities.

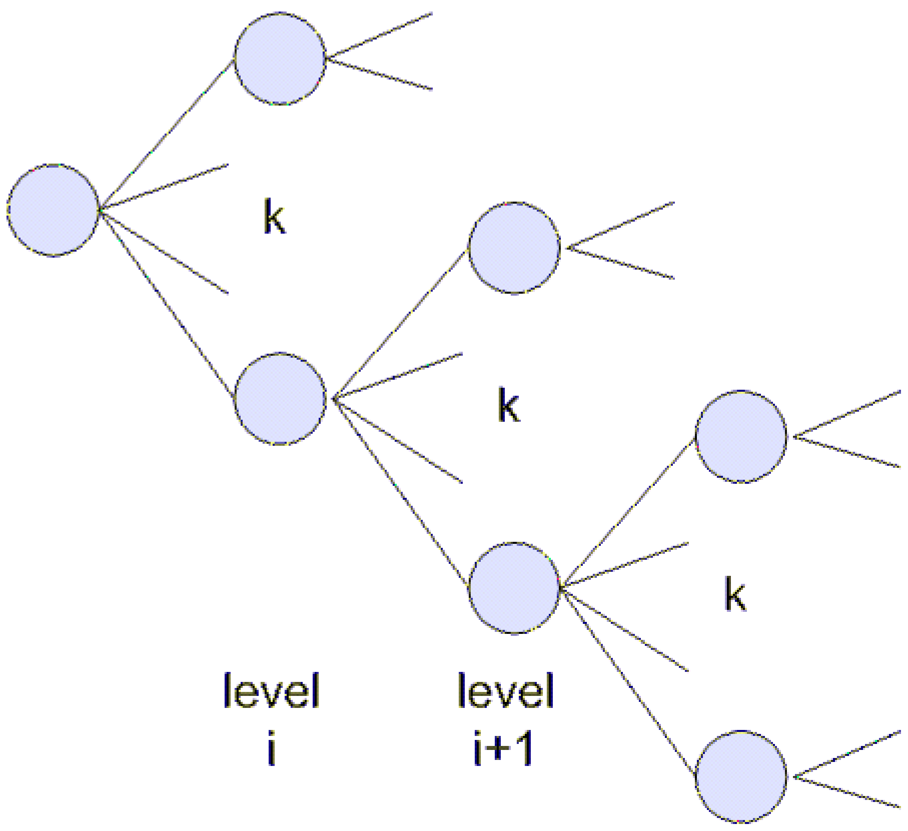


Figure 9: Illustration of chain in defining variables.

We assume that the ratios will be backpropagated from top variable to variables that define it and further on. From level i to level $i+1$, we assume that $R(X^{i+1}$=normal | X^i = normal$)$ = R_{nn}, which is the ratio of normal values at level $i+1$, given that the variable at level i has value normal. We denote the top level by 0. Also $R(X^{i+1}$=alarm | X^i = normal$)$ = R_{na} and similarly for other combinations of normal, alarm and failure values. This ratio definition can be written in the matrix form

$$R = \begin{bmatrix} R_{nn} & R_{na} & R_{nf} \\ R_{an} & R_{aa} & R_{af} \\ R_{fn} & R_{af} & R_{ff} \end{bmatrix}$$

where row sums equal to 1. Although the top event X at level 0, depend on k other variables at level 1, we consider that only the value of previous level variable defines values at the next level, i.e., there is no cross dependence between values of variables at a given level. The above ratio propagation matrix is similar to a state transition matrix of Markov chains, see e.g. [6].

The branching of the deductive analysis starting from the top event proceeds as follows: Branching after the first step is $r3^k$, where $r= r_n$, r_a or r_f depending on weather the top event is "*X*=normal", "*X*=alarm" or "*X*=failure", respectively. This is because there are 3^k rows in the decision table of *X* (because of *k* variables defining *X* and each variable having 3 possible values). As the ratio of normal, alarm and failure states are r_n, r_a and r_f, there are $3^k\ r_n$ normal rows, etc. The first deductive step will produce event "α_1 OR α_2 OR … OR α_d", where $d= r3^k$. Each statement α_i is of the form "$X^1_1 = v_1$ AND … AND $X^1_k = v_k$", where v_i= normal, alarm or failure. The deductive analysis proceeds on each level 1 variable in turn in a similar fashion.

When analysis is continued to *m*th level the average branching will be

$$3^k (r_n\ r_a\ r_f)\pi^T{}_m, \quad \pi_m = \pi_0 R^m$$

where T denotes transpose and π_0 = (1 0 0) for top event "*X*=normal" or (0 1 0) for top event "*X*= alarm" or (0 0 1) for top event "*X*=failure". Note that π_m is exactly the state probability vector after m state transitions in a Markov chain with initial state probabilities given by π_0.

The average branching can be plotted as a function of the level of deductive analysis. Here it is easy to see how branching depends on the top event chosen. Initial ratios r_n, r_a and r_f together with the ratio transition matrix *R* needs to be defined for each scenario.

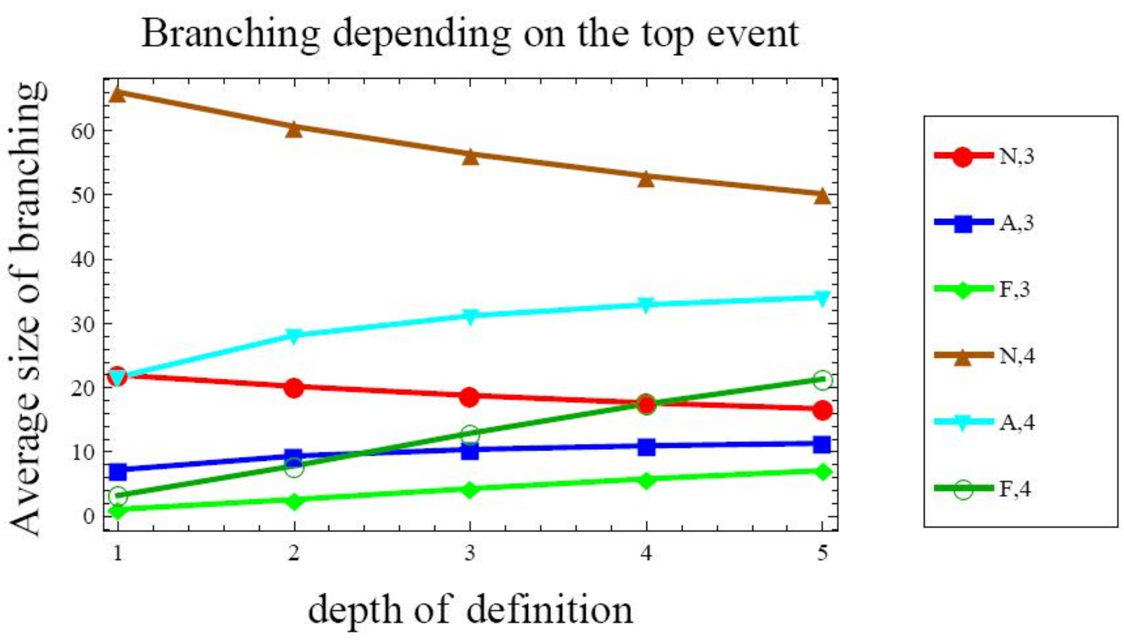


Figure 10: Average branching depending on the top event, N=normal, A=alarm and F=failure. N,k refers to top event "X=normal", where each variable depends on k other variables, where k= 3 or 4. Parameters are r_n=0.9, r_a=0.07, R_{nn}=0.9, R_{na}=0.05,R_{an}=0.25, R_{aa}=0.5, R

Figure 10 illustrates how asking about normal events will produce a lot of variation on how they can occur. Curves corresponding to asking about normal events, (N,4 and N,3) have the highest number of branching. These are also the only decreasing curves. Asking about failure events (curves F,4 and F,3) will produce less branching (and diversity), but variable values gradually start to obtain values "alarm" or "normal" the more frequently when the depth of definition chain increases.

In the second example illustrated in Figure 11 the failure state is absorbing, i.e., once the variable at the higher level fails, then the variables at a lower level will obtain value "failure" too. Normal and alarm states gradually develop into the failure state.

Cases illustrated in Figure 10 and Figure 11 are compared in Figure 12, when each variable depends on 3 other new variables. We can see that ability to recover from a failure state, i.e., whether failure state is absorbing or not, affects on branching most, if the top event concerns about failure and there is minor impact on branching for top events concerning alarm states. The influence on normal states is not considered relevant.

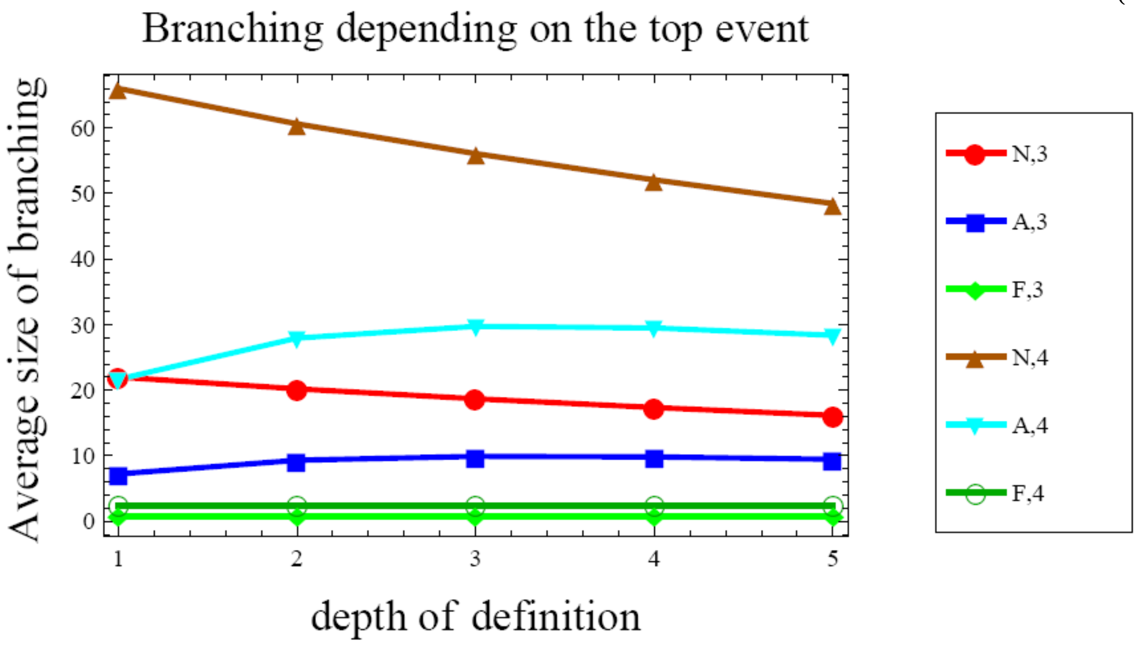


Figure 11: Average branching depending on the top event, N=normal, A=alarm and F=failure. Failure state is absorbing. N,k refers to top event "X=normal", where each variable depends on k other variables, where k= 3 or 4. Parameters are r_n=0.9, r_a=0.07, R_{nn}=0.9, R_{na}=0.05,R_{an}=0.25, R_{aa}=0.5, R_{fn} =0, R_{fa}=0.

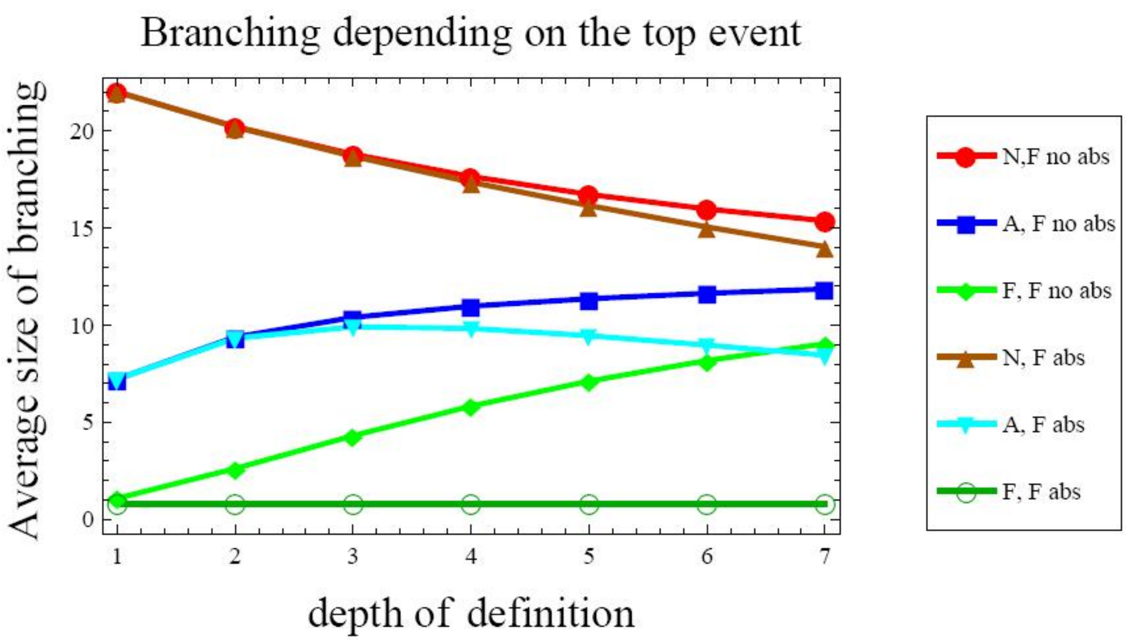


Figure 12: Comparison of branching when k= 3. "N, F no abs" refers to normal top event and failure state is not absorbing. Other curves are denoted accordingly.

Case 2: A water tank with a valve

An example of a water tank with a valve has been discussed in [4]. The system has three components: a water tank, a water level measurement device and a water level control valve. A constant flow of water into the tank is assumed. When the high level limit is reached, the water level measurement device sends an open command to the control valve, causing discharge of the tank. When the low level limit is reached, the water level measurement device sends a close command to the control valve. Unwanted situations are too high level of the tank (overfilling or overpressurisation) or too low level of the tank. These may happen due to failure of the components.

The DFM decision tables for the system are given in [4]. The decision tables indicate symmetries in the system and all states have equally many preimages. Therefore the selection of the variable value in the top event does not have any influence on the branching of the deductive analysis, when probabilities of states are not considered.

In the original model the water level was discretized into three levels: high, middle and low, coded by values 1,0,-1, respectively. Assume next that water levels high and middle are equally interesting and there is no need to make a distinction between those. Then we can make a coarser discretization and cope with smaller decision tables. Let us code water level, WL, and water level measurement, WLM, as binary valued: 1 indicates OK value (previously middle or high) and -1 indicates low value. Presumably the ratios of water level values would change, making value 1 more frequent than -1. First, the new coarser scale leads to conflicting rules in the decision tables for water level, WL and valve, V. Original rules that lead to confliction in coarser scale are shown in Table 22.

#Output			
ID	Time		
WL	0	-1	0
#Inputs			
V	-1	1	1
WL	-1	0	1
#End			
#Output			
ID	Time		
V	0	0	1
#Inputs			
V	-1	0	1
VF	0	0	0
WLM	0	1	1
#End			

Table 22: Rules that lead to confliction in coarse scaling.

Depending on the selection of the winning rule in rule conflict, the number of columns (i.e., preimages of certain water level values or valve values) resulting to different states may change. Here the selection can be done so that each state has equally many preimages. Then the deductive analysis branches equally much regardless of the value of the variable in the top event. If selection is done so that states have unequal number of preimages, then deductive analysis branches differently depending on the values of the variables in the top event. In this case only a minor change in ratios occurs.

5.2 Computational effort depends on how the model is built

The aim of this section is to illustrate some model building aspects. In general, the computational effort of solving a fault tree (or top event) depends on the breadth and width of the tree. However, when the shape of the tree can be detected in advance, the computational effort may be improved. We illustrate this by writing a top event in various logically equivalent forms. However, different forms may behave computationally differently and in some cases the computational performance may depend on how some computer software treats the model internally.

Let us consider a top event, called TOP, concerning six Boolean variables A, B, C, D, E and F. We denote logical operators AND, OR and NOT by ·, + and ¬, respectively. We can write

$$\begin{aligned} TOP &= A\cdot(B\cdot C+\neg C\cdot D+E\cdot(\neg F+\neg B)) \\ &= A\cdot B\cdot C+A\cdot\neg C\cdot D+A\cdot E\cdot\neg F+A\cdot E\cdot\neg B \\ &= A\cdot((\neg C+B)\cdot(C+D))+A\cdot E\cdot(\neg F+\neg B) \end{aligned}$$

The last from is obtained by using ¬V · G + V · H = (V + G) · (¬V · H). The second expression is in the prime implicant form. The first expression clearly expresses the fact that the top event concerns A AND some more complicated event. While the prime implicant form can be seen as a completely balanced from, the first form is quite the opposite. Note that the prime implicant form is actually the problem solved. In reality, the top event is unlikely to be expressed in the prime implicant form. The last form consists of two somewhat complicated events connected by OR operator. This could represent a more typical case of the top event.

Fault trees in all cases are built by reading the top event from left to right. The fault tree corresponding to the first form of the top event is illustrated in Figure 13. In the first form the unbalance is easy to see. Computational effort of the evaluation can be very low – if evaluation is started from A and this happens to be false – or high, if computation is started from the right-hand branch of the fault tree. Now the computational time may depend on how software starts to solve the problem and whether the software can detect asymmetries in the problem and utilize it in the solution.

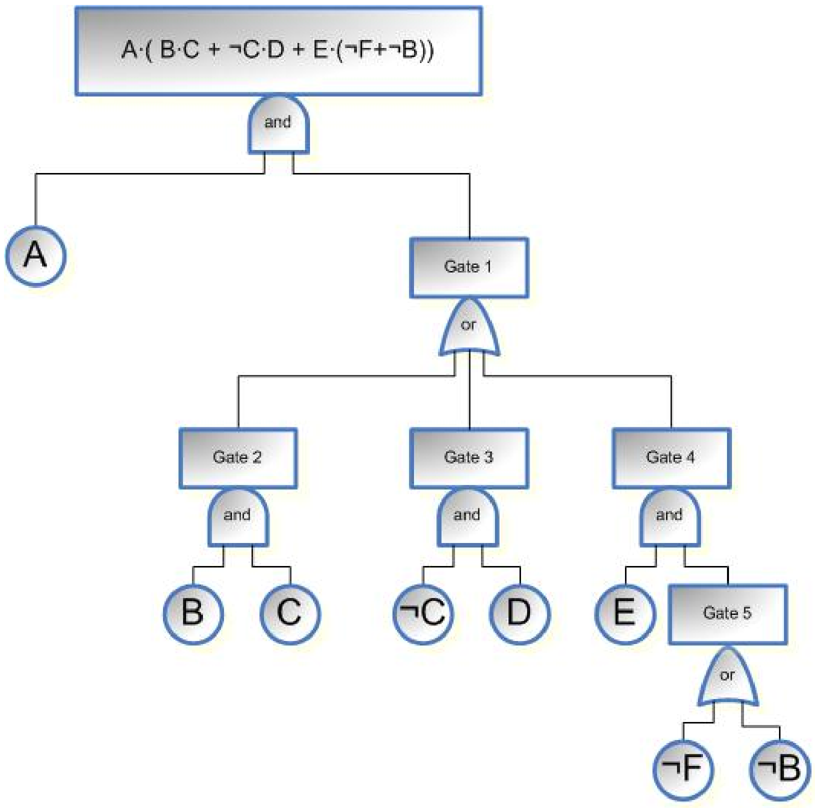


Figure 13: Fault tree of top event, and gate at highest level.

The fault tree of the top event in prime implicant from is illustrated in Figure 14. This form is likely to be computed with similar computational effort is various software implementations. All branches in OR gate need to be computed and all sub-problems are of equal size and complexity.

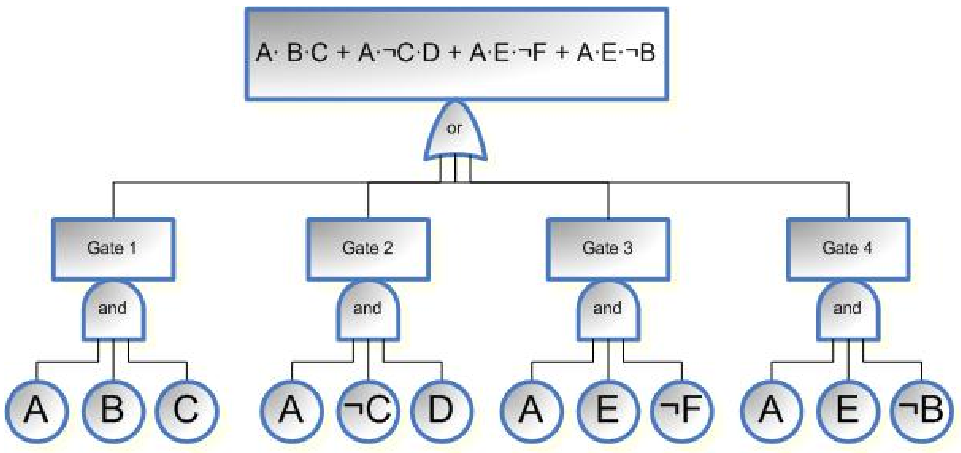


Figure 14: Fault tree of the top event in prime implicant form.

A compromise between the two above formulations and fault trees is illustrated in Figure 15. There the tree is balanced by braches of almost equal complexity, but the topmost gate is OR. Some fault tree solver softwares examine the shape of the fault tree in advance to optimize the computational effort. However, it is not clear if Dymonda is taking advantage of examining the top event for reducing the computational effort. This last fault tree would be beneficial to convert to the first presented form in order to utilize the asymmetry and speed the solving.

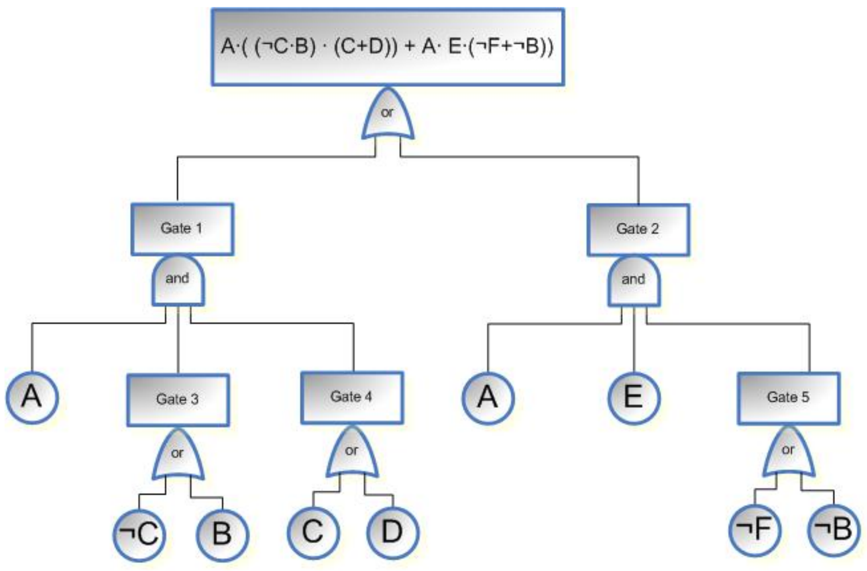


Figure 15: Fault tree of the top event, balanced branches.

6 Conclusions

A common theme that emerges from the experiments in section 4 and considerations in section 5 is that it is the branching occurring in decision tables that determines computational complexity of a DFM model. Here branching is to be understood as the number of rows that match a given state of the result variable. The role of the top event is more subsidiary, mainly relating to whether the variables in it can be linked with a chain of variables to a variable whose decision table contains a lot of branching.

It yet remains a research issue to define this branching in more precise terms, and find the principles that govern its relations to computational complexity in model structures that are typically used. Another research topic, only tangentially touched in this report, is the dependence of the sizes of the transition tables on model structure.

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. Arndt. Dynamic reliability modeling of digital instrumentation and control systems for nuclear reactor probabilistic risk assessments. NUREG report NUREG/CR-6942. United States Nuclear Regulatory Commission, October 2007.

[2] T. Aldemir, S. Guarro, J. Kirschenbaum, D. Mandelli, L.A. Mangan, P. Bucci, M. Yau, B. Johnson, C. Elks, E. Ekici, M.P. Stovsky, D.W. Miller, X. Sun, S.A. Arndt , Q. Nguyen, J, Dion. A Benchmark Implementation of Two Dynamic Methodologies for the Reliability Modeling of Digital Instrumentation and Control Systems. NUREG/CR-6985, United States Nuclear Regulatory Commission, February 2009.

[3] Sanjeev Arora and Boaz Barak. Computational complexity – a modern approach. Cambridge University Press 2009.

[4] K. Björkman and J-E Holmberg, Comparison of two dynamic reliability analysis tools to solve dynamic flowgraph method models, VTT Research Report VTT-R-00775-10, 2010.

[5] Chris J. Garrett, Sergio B. Guarro and George E. Apostolakis. The dynamic flowgraph methodology for assessing the dependability of embedded software systems. *IEEE Transactions on Systems, Man , and Cybernetics* **25** (May 2005), No. 5, 824-840.

[6] D.Gross and C.M. Harris, *Fundamentals* of *Queueing Theory* (3rd ed.), New York: Wiley, 1998, ISBN 0-471-17083-6, xi + 459 pp.

[7] Houtermans, M., 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.

[8] Houtermans, M., G. Apostolakis, A. Brombacher, and D. Karydsas. The dynamic flowgraph methodology as a safety analysis tool: programmable electronic system design and verification. *Safety Science* **40** (2002), 813-833.

[9] Karanta, I., Maskuniitty, M. Reliability of digital control systems in nuclear power plants - Modelling the feedwater system. VTT Research Report VTT-R-01749-08, 2009.

[10] D.E. O'Leary, The relationship between errors and size in knowledge-based systems. *International Journal of Human–Computer Studies* **44** (1996), pp. 171–185.

[11] W.E. Vesely, F.F. Goldberg, N.H. Roberts, D.F. Haasl, Fault tree handbook. NUREG-0492, Nuclear Regulatory Commission, January 1981.

[12] Kar-Fai Michael Yau, Dynamic flowgraph methodology for the analysis of software based controlled system. PhD Dissertation, University of California, Los Angeles, June 1997 (unpublished).

[13] Yau, M., 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 (1995), 335-353.

[14] Zeidler, E. (ed.). *Oxford user's guide to mathematics*. Oxford: Oxford University Press, 1996, ISBN 0 19 850763 1, xxii+1285 pp.