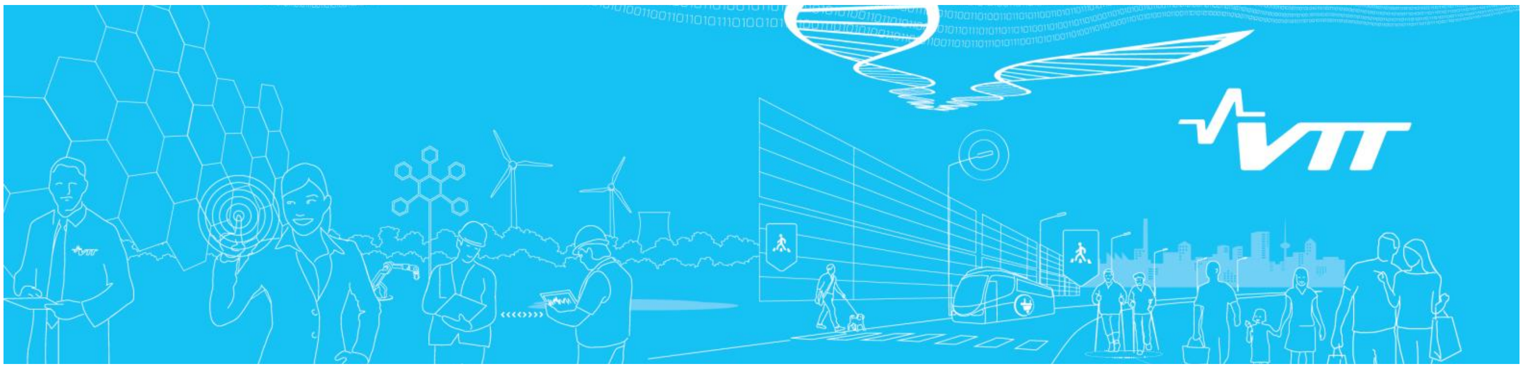

RESEARCH REPORT

VTT-R-04628-17

# Solving dynamic flowgraph models with FinPSA

Authors: Tero Tyrväinen

Confidentiality: Public

| | |
|---|---|
| **Report's title**<br>Solving dynamic flowgraph models with FinPSA | |
| **Customer, contact person, address**<br>VYR | **Order reference**<br>SAFIR 4/2017 |
| **Project name**<br>Probabilistic risk assessment method development and applications | **Project number/Short name**<br>114491/PRAMEA |
| **Author(s)**<br>Tero Tyrväinen | **Pages**<br>13/2 |
| **Keywords**<br>Dynamic flowgraph methodology, fault tree, FinPSA | **Report identification code**<br>VTT-R-04628-17 |

**Summary**

Dynamic flowgraph methodology (DFM) is a method for the reliability analysis of dynamic systems with time-dependencies and feedback loops. A DFM model is a graph representation of the analysed system, and its components are analysed at discrete time points and they can have multiple states. The aim of DFM is to identify which conditions can cause a top event, e.g. the system's failure. The reason for the development of DFM is that traditional methods, such as fault tree analysis, can describe the system's dynamic behaviour only in a limited manner. DFM can more accurately represent the system's evolution in time.

VTT has previously developed a DFM tool called Yadrat, which solves a DFM model by transforming it into a binary decision diagram. It has however not been possible to apply Yadrat to large systems. To improve Yadrat's computational capacity and to reduce computation times, this report studies an alternative way of solving a DFM model. A DFM model is transformed from Yadrat into the fault tree format of the probabilistic risk assessment (PRA) software FinPSA, then solved by the fault tree algorithm of FinPSA. To make this possible, FinPSA's fault tree algorithm has been developed to take into account the non-coherent logic of DFM.

A simple DFM model has been solved correctly using FinPSA. However, with the preliminary implementation it has not yet been possible to solve more complex models correctly. Therefore, there has not been a possibility to evaluate the computational efficiency of the implementation yet. The next step is to resolve the issues conserning the computation of larger models. Then, FinPSA implementation can be benchmarked with Yadrat to find out whether FinPSA is faster and able to solve large models.

| | |
|---|---|
| **Confidentiality** | Public |

Espoo 16.10.2017

| **Written by** | **Reviewed by** | **Accepted by** |
|---|---|---|
| [illegible] | [illegible] | [illegible] |
| Tero Tyrväinen<br>Research Scientist | Kim Björkman<br>Research Scientist | Eila Lehmus<br>Research Team Leader |

**VTT's contact address**
VTT, PL 1000, 02044 VTT

**Distribution (customer and VTT)**
SAFIR reference group 2

## Contents

# 1. Introduction

Dynamic flowgraph methodology (DFM) is a method for the reliability analysis of dynamic systems with time-dependencies and feedback loops [1-5]. As in fault tree analysis, the aim of DFM is to identify which conditions can cause a top event, which can be, for example, a system's failure. A DFM model is a directed graph representation of the analysed system. The nodes of the graph represent system’s variables, e.g. physical and software variables, and the edges between them represent causal and other relationships between the variables. Components of DFM models are analysed at discrete time points and they can have multiple states. The reason for the development of DFM is that traditional methods, such as fault tree analysis, can describe a system's dynamic behaviour only in a limited manner. DFM can more accurately represent system's evolution in time.

DFM has been most often applied to different digital control systems that include both hardware and software components. One reason for this is that a DFM model can represent the interactions between a control system and the controlled process. DFM supports the modelling of multi-state components, which is an advantage in modelling digitally controlled systems because their variables generally do not behave in binary manners. Another advantage of DFM is that only one model is needed to represent the complete behaviour of a system and therefore different states of the system can be analysed using the same model [2].

The result of DFM analysis is a set of prime implicants [6, 7]. A prime implicant is a minimal combination of basic events and other conditions that is sufficient to cause the top event. DFM analysis considers at which time points events have to occur to cause the top event. Compared to static fault tree analysis, DFM provides more accurate information about the development of accident scenarios and enables more accurate probability calculations.

VTT has previously developed a DFM tool called Yadrat [8], which solves a DFM model by transforming it into a binary decision diagram (BDD). It has however not been possible to apply Yadrat to large systems, mainly because DFM analysis is generally quite demanding, but also because of the limitations of BDDs. To improve Yadrat’s computational capacity and to reduce computation times, this report studies an alternative way of solving a DFM model. A DFM model is transformed from Yadrat into the fault tree format of the probabilistic risk assessment (PRA) software FinPSA [9], then solved by the fault tree algorithm of FinPSA.

It is not a new idea to solve a DFM model using fault trees. The original DFM implementation [1] already used that approach. In Dymonda software [10], prime implicants are solved by applying the method of generalized consensus [11, 12] to initial prime implicants solved from a fault tree. FinPSA implementation developed in this work is also based on post-processing of initial prime implicants but is slightly different from the method of generalized consensus.

# 2. Dynamic flowgraph methodology

A DFM model is a directed graph that consists of nodes representing the system's components and variables, and edges representing causal and other dependencies between nodes [1-5]. A node can have a finite number of states and the state of a node is determined either by a probability model or by states of its input nodes at specified time steps relative to the time step considered. Input dependencies of a node are represented in a decision table which is an extension of a truth table. Decision tables can be constructed based on empirical knowledge on the system, physical equations, simulations, expert judgement, software design or software code.

Figure 1 shows an example of a DFM model based on a tank system with a digitally controlled valve, and Table 1 gives an example of a decision table. A tank gets water from an infinite water source. The outflow from the tank is regulated by a valve, which is controlled by a digital controller based on measurement of the water level. In the model, node V represents the functional state of a valve (V=0 means that the valve is closed, V=1 means that it is open), WLM represents water level measurement value (WLM=0 means that the water level is below the reference value, WLM=2 that it is above the reference value, and WLM=1 that it is within tolerance limits of the reference value) and WL represents water level. Nodes VF and MF determine whether the valve and the water level measurement are failed and they change states by a probability model. Each row of the decision table represents a state combination of input nodes (VF, WLM and V) and the state of the output node V to which the state combination of the input nodes leads. The state of a node can also be defined irrelevant ("irr" in the table for WLM), which means that the state of the output node is the same regardless of in which state the input node (WLM) is. The time lag row determines the delays in the dependencies between the input nodes and the output node. The time lags are also seen in Figure 1. In Table 1, node V depends on its own state at the previous time step because the time lag is 1.

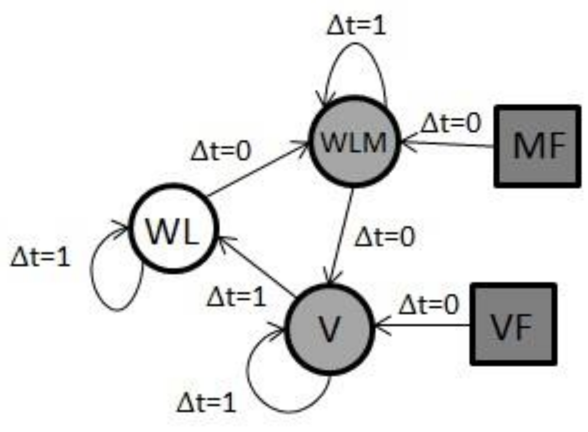

*Figure 1: A DFM model with five nodes [7].*

*Table 1: The decision table of component V.*

| | Output | Inputs | | |
|---|---|---|---|---|
| Node | V | VF | WLM | V |
| Time lag | | 0 | 0 | 1 |
| | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 1 |
| | 0 | 0 | 1 | 0 |
| | 1 | 0 | 1 | 1 |
| | 1 | 0 | 2 | 0 |
| | 1 | 0 | 2 | 1 |
| | 0 | 1 | irr | 0 |
| | 1 | 1 | irr | 1 |

The primary target of DFM is to identify prime implicants of the top event. A prime implicant is a minimal combination of conditions that is sufficient to cause the top event. Respectively, an implicant is any combination of conditions that causes the top event, minimal or non-minimal. In DFM, these conditions are represented by literals. In this context, a literal is triplet consisting of a node $V$, state $s$ and time point $-t$, and denoted as $V_s(-t)$. Hence, prime implicants of DFM can be understood as multi-state and timed minimal cut sets. The mathematical definition of prime implicants is presented and discussed in [7].

The top event is also defined as a set of literals that all must hold for the top event to happen. The analyst can freely choose any top event.  Therefore, it is possible to analyse several top events in parallel with the same DFM model, and both success and failure scenarios can be analysed. The analyst has to also choose the initial time $-t_I$, which defines how many time steps the analysis will cover. Typically, the top event is fixed to time step 0, and then the analysis covers $t_I$ time steps.

In DFM, there are two types of nodes: deterministic nodes and stochastic nodes. The state of a deterministic node is determined by its input nodes through a decision table. The state of a stochastic node is determined by a probability model. At the initial time step, a deterministic node behaves like a stochastic node. Implicants of a top event can contain initial states of deterministic nodes and states of stochastic nodes at any time step.

A DFM model is typically analysed by tracing event sequences backwards from effects to causes. Deductive analysis starts from the top event. The model is traced backwards in the cause-and-effect flow to identify what initial states of deterministic nodes and states of stochastic nodes produce the top event. The process ends when the initial time step is reached.

Failures of non-repairable components can be modelled using failure nodes. A failure node is a stochastic node with two states, a “functioning state” (0) and a “failure state” (1). It cannot turn from state 1 to state 0, and it is fixed to state 0 at the initial time step. In the Yadrat tool, a failure node is modelled with an assistive random node and a decision table. The random node does not appear in the graph model created by the user, but is added there automatically by Yadrat. First, the failure is affected by the random node so that the failure node turns to the state 1 if the random node gets value 1. After the failure node has turned to the state 1, it is not affected by the value of the random node anymore and the failure node cannot turn back to the state 0. The decision table of a failure node is presented in Table 2.

*Table 1: The decision table of a failure node.*

| | Output | Inputs | |
|---|---|---|---|
| Node | F | F | H |
| Time lag | | 1 | 0 |
| | 0 | 0 | 0 |
| | 1 | 0 | 1 |
| | 1 | 1 | 0 |
| | 1 | 1 | 1 |

## 3. FinPSA

FinPSA is a software tool for full-scope PRA [9]. FinPSA supports PRA levels 1 and 2. Level 1 PRA concerns accident sequences leading to core damage and calculation of the core damage frequency. Level 2 PRA concerns the progression of severe accidents after core damage and calculation of frequencies and amounts of radioactive releases.

In level 1 PRA, FinPSA uses event trees and fault trees [13, 14]. An event tree represents how an accident can evolve from an initiating event via failures of safety systems to a consequence, e.g. core damage. A fault tree represents which events can cause the analysed system to fail. Fault trees are linked to branching points in event trees. From fault trees, minimal cut sets are solved. Minimal cut sets are minimal combinations of events that can cause the top event, e.g. core damage. Probabilistic analysis is performed based on minimal cut sets and reliability data of components.

FinPSA has been designed for large models and its computational capacity is extremely high. FinPSA uses efficient data structures, fault tree manipulation and truncation to solve minimal cut sets. It divides a fault tree into a set of subproblems and solves the subproblems in an order that is optimal with regard to computation times.

A relevant property of FinPSA with regard to this work is a not-group. It defines events that are mutually exclusive and cannot appear in same minimal cut sets. Typically, some initiating events are placed in the same not-group. In DFM, different states of a node are mutually exclusive. Therefore, not-groups are utilised in the generation of prime implicants. However, they are used a bit differently than in normal PRA.

# 4. Solving DFM model with FinPSA

To solve a DFM model with FinPSA, the following steps need to be performed:

1. A DFM model from Yadrat is transformed into FinPSA's fault tree import format.
2. The fault tree is imported in FinPSA.
3. Prime implicants of the DFM model are solved from the fault tree in FinPSA.

## 4.1 Transforming DFM model into fault tree

A DFM model can be transformed into a fault tree using the following basic rules:

1. Literals representing the top event are set as inputs for the top gate, which is an AND gate.
2. The rows of a decision table leading to the considered state are combined under an OR gate.
3. A row of a decision table is represented by an AND gate, and literals representing the states of the input nodes are set under this AND gate.
4. Literals representing stochastic nodes and initial states of deterministic nodes are set as basic events. Initial states of a deterministic node are the node's value(s) at time instances $-t \leq -t_I$.

The fault tree production is implemented in Yadrat using the following functions:

**StartBuildingFT**

Function writes the header rows of FinPSA fault tree import file. For example,

```
FinPSA transfer data
Fault tree created by Yadrat
 ---------- Fault tree <transfer data> ----------
FT3
```

```
        Tree
```

It also writes a row for the top gate, where literals representing the top event are combined under an AND gate. For example,

```
        Top             AND         WL_0_2      WL_-1_2
```

Literals are written in text form: 'Node_Time_State'. Then, the function starts developing the fault tree for each literal of the top event. If the time step of a literal is the initial time, function CreateFTatInitial is called for it. If the time step of a literal is not the initial time, function CreateGate is called for it.

The fault tree is finalised by writing "EndTree" to the FinPSA fault tree import file. After that, the function calls the WriteDataBase function.

**CreateFTatInitial (Literal)**

Function calls the IsLegalBasicEvent function to find out if the literal should be represented by a basic event or gate in the fault tree. If the literal is represented by a basic event, the CreateNewBE function is called. If the literal is represented by a gate, the CreateGate function is called.

**IsLegalBasicEvent (Literal)**

The function checks if the literal represents a stochastic node or a deterministic node. If it represents a stochastic node, the function returns 'true'. If the literal represents a deterministic node, the function checks if the time step of the literal is the initial time or smaller. If the time step is larger than the initial time, the function returns 'false'. If the time step is the initial time or smaller, the function checks the time lags of the input dependencies of the deterministic node. If one of the time lags is larger than 0, the function returns 'true'. If all time lags are 0, the function returns 'false'.

**CreateNewBE (Literal)**

The function writes a row where the literal is defined as a basic event. For example,

```
        V_-3_0          BE
```

It also adds the literal to the list of handled literals and returns the literal in text form. However, if the probability of the literal is 1, the function returns '1', and if the probability of the literal is 0, the function returns '0'. For example, a failure node must be in state 0 at the initial time. Initial state probabilities of some other nodes can also be 0 or 1.

**CreateGate (Literal)**

If the literal has already been handled, the function only returns the literal in text form. Otherwise, the function writes a row where the literal is defined as an OR gate, and the inputs are the rows of the decision table (of the node represented by the literal) that have the same state as the literal. For example,

```
        WL_-2_0         OR          WL_-2_0_Row1        WL_-2_0_Row3
```

For each row of the decision table having the same state as the literal, the function writes a row where literals representing the input nodes are combined under an AND gate. For example,

```
        WL_-2_0_Row1    AND         V_-3_1      WL_-3_0
```

For each input node, the state is taken from the decision table, and the time step is the time step of the literal representing the output node minus the time lag specified in the decision table. If the state of an input node is defined irrelevant, no literal representing the input node is created and written in the fault tree file. For each input literal, the IsLegalBasicEvent function is called. If it returns 'false', the CreateGate function is called. If the IsLegalBasicEvent function returns 'true', the CreateNewBE function is called. If the CreateNewBE or CreateGate function returns '0', no row is written for the corresponding decision table row. If the CreateNewBE or CreateGate function returns '1', no literal representing the input node is created and written in the fault tree file. If the CreateNewBE and CreateGate functions return '1' for all inputs, the function also returns '1' and does not write OR and AND rows. If the CreateNewBE or CreateGate function returns '0' in each relevant decision table row, the function returns '0' and does not write OR and AND rows.

State 1 of a failure node is handled slightly differently in order to optimize the fault tree structure. The literal is defined as an OR gate that has two inputs: the assistive random node of the failure node in state 1 at the same time step, and the failure node in state 1 at the previous time step. For example,

```
MF_-1_1          OR          MF_-2_1    MF=help_-1_1
```

For the assistive random node, the CreateNewBE function is called, and for the failure node at the previous time step, the CreateGate function is called. If the previous time step is the initial time, the assistive random node of the failure node in state 1 at the same time step is the only input. For example,

```
VF_-2_1          OR          VF=help_-2_1
```

The function finally adds the literal (which is represented by the OR gate) to the list of handled literals, and returns the literal in text form.

**WriteDataBase**

The function writes the literals of the DFM model into FinPSA's data base format. First, the function writes header rows:

```
 ---------- Data records <transfer data> ----------
Name             Not-group    RelP1
```

Then, the function goes through all literals that are represented by basic events in the fault tree. For each literal, the function writes the name, not-group and probability. For example,

| | | |
|---|---|---|
| V_-3_0 | V | 0.4 |
| VF=help_-2_0 | VF-2 | 0.999999 |
| V_-3_1 | V | 0.6 |
| WL_-3_0 | WL | 0.33 |
| WL_-3_1 | WL | 0.34 |
| WLM_-3_0 | WLM | 0.33 |
| MF=help_-2_0 | MF-2 | 0.999999 |
| WLM_-3_1 | WLM | 0.34 |
| WLM_-3_2 | WLM | 0.33 |
| MF=help_-2_1 | MF-2 | 1.0E-6 |

Literals that represent the same node at the same time step are placed in the same not-group. For deterministic nodes, the name of the not-group is simply the name of the node: there is no need to attach a time label because only literals representing deterministic nodes at the initial time are treated as basic events. For stochastic nodes, the name of the not-group is a combination of the name of the node and the time step.

## 4.2 Solving prime implicants from a fault tree

The main challenge in the solving of prime implicants is the non-coherent logic of DFM. The algorithms of FinPSA assume that the model is coherent. Hence, the correct prime implicants cannot be generated in FinPSA without some modifications to the source codes. The normal algorithm of FinPSA can generate a set of implicants, but they are not minimized correctly. It has to be taken into account that each node is always in one of its states at each time step.

The problem can be illustrated with a simple example. Let node A be a failure node, and nodes B and C be deterministic nodes with two states 0 and 1. The initial time of the analysis is -3. Literal sets $\{A_0(-2), B_0(-3), C_0(-3)\}$, $\{A_1(-2), B_0(-3), C_0(-3)\}$ and $\{A_1(-2), B_0(-3), C_1(-3)\}$ are identified as implicants, but since node A is always either in state 0 or 1, set $\{B_0(-3), C_0(-3)\}$ must also be an implicant. Similarly, set $\{A_1(-2), B_0(-3)\}$ must be an implicant because node C is always in state 0 or 1. Hence, the set of those three implicants can be minimized into a set of two implicants, $\{B_0(-3), C_0(-3)\}$ and $\{A_1(-2), B_0(-3)\}$.

In the preliminary DFM implementation in FinPSA, the literals that represent the same node at same time step are placed into a not-group. When a new implicant is identified, for each literal in the implicant, it is checked if the literal can be switched to the other literals in the same not-group so that the top event still occurs. If the literal can be switched to each of the other literals, it can be removed from the implicant so that it still remains implicant. This is done by comparing the implicant canditates with switched literal to other existing implicants. If all those implicant candidates are already found in the set of implicants, they are removed from the set and a new implicant with the literal removed is added.

If set $\{A_0(-2), B_0(-3), C_0(-3)\}$ has previously been added to the list of implicants, and $\{A_1(-2), B_0(-3), C_0(-3)\}$ is identified as new implicant, literal $A_1(-2)$ is switched to $A_0(-2)$ so that an implicant candidate $\{A_0(-2), B_0(-3), C_0(-3)\}$ is created based on $\{A_1(-2), B_0(-3), C_0(-3)\}$. Then, $\{A_0(-2), B_0(-3), C_0(-3)\}$ is compared to the existing implicants, and the same implicant can be found from the list, which means that it can be removed from the list, and $\{B_0(-3), C_0(-3)\}$ can be added as a new implicant. However, before $\{B_0(-3), C_0(-3)\}$ is added, similar analysis is also performed for it, i.e. sets $\{B_1(-3), C_0(-3)\}$ and $\{B_0(-3), C_1(-3)\}$ are searched from the list. Also, sets $\{A_1(-2), B_1(-3), C_0(-3)\}$ and $\{A_1(-2), B_0(-3), C_1(-3)\}$ are searched from the list, but cannot be found at this point.

Implicant canditates need not to be compared only to the implicants with the same length, but also shorter implicants. If now $\{A_1(-2), B_0(-3), C_1(-3)\}$ is identified as an implicant, it needs to be checked if $\{A_0(-2), B_0(-3), C_1(-3)\}$, $\{A_1(-2), B_1(-3), C_1(-3)\}$ and $\{A_1(-2), B_0(-3), C_0(-3)\}$ are also implicants. The two first sets are not implicants, but $\{A_1(-2), B_0(-3), C_0(-3)\}$ is an implicant, because it implies set $\{B_0(-3), C_0(-3)\}$ which has previously been identified as an implicant. Based on this, it is known that literal $C_1(-3)$ can be removed from implicant $\{A_1(-2), B_0(-3), C_1(-3)\}$, and $\{A_1(-2), B_0(-3)\}$ is also an implicant. Again, sets $\{A_0(-2), B_0(-3)\}$ and $\{A_1(-2), B_1(-3)\}$ need to be compared to the existing implicants, but they are not identified as implicants. Therefore, $\{A_1(-2), B_0(-3)\}$ is added to the list of implicants, and at the end of the process, the list contains implicants $\{B_0(-3), C_0(-3)\}$ and $\{A_1(-2), B_0(-3)\}$.

An algorithm for the minimization of implicants is presented as follows. It is performed for each newly identified implicant (after it has been checked that the implicant is of minimal length and has not been identified before). After completing a step in the algorithm, move to the next step if no other instructions are given.

1. Get the first literal in the new implicant P. The literal is denoted as L, the node that is represented by the literal is denoted as N and the state of the node in the literal is denoted as S.

2. Get the first state of node N, which is different from state S. This new state is denoted as S*.

3. Create a new set of literals P*, which is otherwise the same as the implicant P, except that state S in literal L is switched to state S*.

4. Check if set of literals P* can be found in the set of implicants. If it does, add this implicant P* to the list of potentionally removable implicants and go to step 6.

5. Check if a subset of set of literals P* can be found in the set of implicants. If not, go to step 10.

6. Check if all the states of node N have been examined. If not, get the next state, which is different from state S, and go to step 3. The new state is then denoted as S*.

7. Remove all implicants that are in the list of potentionally removable implicants from the set of implicants.

8. Create a new implicant PM, which is otherwise the same as the implicant P, except that literal L is left out. Write down that the implicant P will not be added to the set of implicants.

9. Perform the whole procedure starting from step 1 for the new implicant PM.

10. Check if all the literals in the implicant P have been examined. If not, empty the list of potentionally removable implicants, get the next literal in the implicant P and go to step 2. The next literal is then denoted as L, the node that is represented by the literal is denoted as N and the state of the node in the literal is denoted as S.

11. If step 8 has not been applied for the implicant P, add it to the set of implicants.

The previously described procedures are implemented in the AddMinimal function of FinPSA's MinEngine unit. See reference [15] for more information about MinEngine. Other parts of the code are the same as in regular minimal cut set search.

FinPSA solves minimal cut sets (or prime implicants) by dividing the fault tree into a set of smaller subproblems, which are solved in optimal order with regard to computation times. Minimization of implicants is applied already on the subproblem level, which is expected to be more efficient than the solution where the minimization would be performed only at the end of analysis.

## 5. Results

The simple model presented in Figure 1 was analysed with the top event that the water level is low at time steps -1 and 0. The decision tables of the model are presented in Table 1 and Appendix. The model was traced backwards 3 time steps. All prime implicants were correct, i.e. the same as presented in [16]. They are also presented in Appendix. Probabilities are also presented in the FinPSA output of Appendix, but attention should not be paid to them, since the FinPSA implementation has not been developed yet to calculate probabilities correctly in all cases.

With larger models, prime implicants could not yet be solved correctly. In some cases, FinPSA produced no prime implicants at all. Before the computational capabilities of the developed approach can be assessed, the issues with the preliminary FinPSA implementation need to be resolved.

## 6. Discussion and conclusions

This report presented how DFM models can be solved using the fault tree algorithm of FinPSA. First, a DFM model is transformed from DFM tool Yadrat to the fault tree format of FinPSA. Then, the prime implicants of the DFM model can be solved from the fault tree. The fault tree algorithm of FinPSA could not be applied to non-coherent DFM models as such. Therefore, a new part had to be added to the algorithm. The new algorithm identifies mutually exclusive events in implicants and minimizes the implicants from that basis. The algorithm presented in Section 4 is not FinPSA or fault tree specific. It could be applied in any tool that solves prime implicants of a non-coherent reliability model.

Prime implicants were solved correctly for a simple model, which gives confidence that the approach has potential. More complex models could not be solved correctly, and in some cases, FinPSA produced no prime implicants at all. The next step is to resolve these issues.

Because of problems with complex models, the computational efficiency of the implementation could not be evaluated yet. Assuming that the implementation can be corrected for larger models, it can be expected to be efficient because the implicants are minimized on the subproblem level of the analysis, which restricts the number of implicants that need to be handled. However, the implementation is only tentative. FinPSA optimizes the order in which different subproblems are solved, but the optimization has been developed only from the minimal cut set solving point of view, which means that the order could be different for a DFM model.

The most well-known DFM tool Dymonda utilises the method of generalized consensus [11, 12] to solve prime implicants. Implementation of the method of generalized consensus in FinPSA could also be considered. Especially, if it was found out that the current implementation does not produce complete set of prime implicants, the method of generalized consensus could be used to find the missing prime implicants.

Prime implicant solving algorithm has been implemented in a developer version of FinPSA. The implementation is considered tentative and experimental. Literals representing different states of a node at the same time step have been placed in the same not-group, and the minimization of implicants is performed based on the not-groups. In FinPSA, not-groups are typically used differently. Hence, when the DFM implementation will be added to the release version, mutually exclusive literals need to be defined in some other way. There are several other DFM related properties that could also be implemented in FinPSA: dynamic constaints between literals (e.g. for non-repairable components [7]), computation of probabilities, common cause failure modelling, risk importance measures, etc. Even a new DFM modelling feature could be developed in FinPSA so that there would not be need to transform models from Yadrat.

The possibility to solve a DFM model in FinPSA also makes it easier to integrate it in full scope nuclear power plant PRA. However, it does not solve all the issues related to the integration, such as how to handle component failures that appear both in the DFM model and in fault trees as basic events. The integration of DFM in PRA has been studied previously in [17].

The next step in this work is to study why prime implicants are not solved correctly for larger models and resolve the issues. Then, FinPSA implementation can be benchmarked with Yadrat to find out whether FinPSA is faster and able to solve large models. Improved computational efficiency and capacity could make DFM analysis more attractive, and inspire further DFM studies and practical applications of DFM.

## References

[1] Garrett CJ, Guarro SB, Apostolakis GE. The dynamic flowgraph methodology for assessing the dependability of embedded software systems. IEEE Transactions on Systems, Man and Cybernetics. 1995; 25:824-840.

[2] Al-Dabbagh AW, Lu L. Reliability modeling of networked control systems using dynamic flowgraph methodology. Reliability Engineering and System Safety. 2010; 95:1202-1209.

[3] Aldemir T, Guarro S, Mandelli D, Kirschenbaum J, Mangan LA, Bucci P, Yau M, Ekici E, Miller DW, Sun X, Arndt SA. Probabilistic risk assessment modeling of digital instrumentation and control systems using two dynamic methodologies. Reliability Engineering and System Safety. 2010; 95:1011-1039.

[4] Yau M, Dixon S, Guarro S. Application of the dynamic flowgraph methodology to the space propulsion system benchmark problem. Proceedings of the 12th International Probabilistic Safety Assessment and Management Conference; 2014 Jun 22-27; Sheraton Waikiki, Honolulu, Hawaii, USA.

[5] Milici A, Mulvihill R, Guarro S. Extending the dynamic flowgraph methodology (DFM) to model human performance and team effects. United States Nuclear Regulatory Commission, Division of System Analysis and Regulatory Effectiveness, Washington D.C., 2001. NUREG/CR-6710.

[6] Yau M, Apostolakis G, Guarro S. The use of prime implicants in dependability analysis of software controlled systems. Reliability Engineering and Systems Safety. 1998; 62:23-32.

[7] Tyrväinen T. Prime implicants in dynamic reliability analysis. Reliability Engineering and System Safety. 2016; 146:39-46.

[8] Björkman K. Solving dynamic flowgraph methodology models using binary decision diagrams. Reliability Engineering and System Safety. 2013; 111:206-216.

[9] VTT Technical Research Centre of Finland Ltd (VTT). FinPSA – Tool for promoting safety and reliability, 2016. https://www.simulationstore.com/finpsa.

[10] ASCA Inc. Dymonda. 2010. www.ascainc.com/dymonda/dymonda.html [Referred 17.8.2017].

[11] Quine, WV. A way to simplify truth functions. American Mathematical Monthly. 1955; 62:627-631.

[12] Cepek O, Kucera P, Kurik S. Boolean functions with long prime implicants. Information Processing Letters. 2013; 113:698-703.

[13] International Atomic Energy Agency (IAEA). Development and application of level 1 probabilistic safety assessment for nuclear power plants. Specific safety guide, No. SSG-3. Vienna, 2010. ISBN 978-92-0-114509-3.

[14] Vesely WE, Goldberg FF, Roberts NH, Haasl DF. Fault tree handbook. United States Nuclear Regulatory Commission, Washington, D.C., 1981. NUREG-0492.

[15] Niemelä I, Mätäsniemi T. FinPSA – Software design – Main unit documentation. VTT Technical Research Centre of Finland, Espoo, 2013. Restricted availability.

[16] Tyrväinen T. Risk importance measures and common cause failure in dynamic flowgraph methodology [master’s thesis]. Aalto University, Espoo, 2011.

[17] Björkman K, Tyrväinen T. Dynamic flowgraph methodology as a part of PRA. VTT Technical Research Centre of Finland, Espoo, 2015.

## Appendix

*Table A-1: The decision table of WL.*

| | Output | Inputs | |
|---|---|---|---|
| Node | WL | V | WL |
| Time lag | | 1 | 1 |
| | 1 | 0 | 0 |
| | 2 | 0 | 1 |
| | 2 | 0 | 2 |
| | 0 | 1 | 0 |
| | 0 | 1 | 1 |
| | 1 | 1 | 2 |

*Table A-2: The decision table of WLM.*

| | Output | Inputs | | |
|---|---|---|---|---|
| Node | WLM | MF | WLM | WL |
| Time lag | | 0 | 1 | 0 |
| | 0 | 0 | 0 | 0 |
| | 1 | 0 | 0 | 1 |
| | 2 | 0 | 0 | 2 |
| | 0 | 0 | 1 | 0 |
| | 1 | 0 | 1 | 1 |
| | 2 | 0 | 1 | 2 |
| | 0 | 0 | 2 | 0 |
| | 1 | 0 | 2 | 1 |
| | 2 | 0 | 2 | 2 |
| | 0 | 1 | 0 | 0 |
| | 0 | 1 | 0 | 1 |
| | 0 | 1 | 0 | 2 |
| | 1 | 1 | 1 | 0 |
| | 1 | 1 | 1 | 1 |
| | 1 | 1 | 1 | 2 |
| | 2 | 1 | 2 | 0 |
| | 2 | 1 | 2 | 1 |
| | 2 | 1 | 2 | 2 |

```
Syve test: Syve, cut set importances  170526 12:31   <TPT>
===================================

S1-sum      1.66E-06    TOP mc S1-sum      1.66E-06

 Num        Freq.       %          Cumul       Prob        Name

   1        4.00E-07    24.12      24.12       1.00E-06    VF=help_-2_1
                                               4.00E-01    V_-3_0

   2        2.64E-07    15.92      40.03       2.00E-06    MF=help_-1_1
```

```
                                                1.00E+00   MF=help_-2_0
                                                4.00E-01   V_-3_0
                                                3.30E-01   WL_-3_0

    3        2.64E-07     15.92      55.95      1.00E+00   MF=help_-2_0
                                                2.00E-06   VF=help_-1_1
                                                4.00E-01   V_-3_0
                                                3.30E-01   WL_-3_0

    4        1.36E-07     8.20       64.15      1.00E-06   MF=help_-2_1
                                                4.00E-01   V_-3_0
                                                3.40E-01   WLM_-3_1

    5        1.32E-07     7.96       72.11      1.00E-06   MF=help_-2_1
                                                4.00E-01   V_-3_0
                                                3.30E-01   WLM_-3_0

    6        1.09E-07     6.57       78.67      1.00E-06   MF=help_-2_1
                                                1.00E+00   VF=help_-2_0
                                                3.30E-01   WLM_-3_0
                                                3.30E-01   WL_-3_2

    7        8.98E-08     5.41       84.08      2.00E-06   MF=help_-1_1
                                                4.00E-01   V_-3_0
                                                3.40E-01   WLM_-3_1
                                                3.30E-01   WL_-3_0

    8        8.98E-08     5.41       89.50      2.00E-06   VF=help_-1_1
                                                4.00E-01   V_-3_0
                                                3.40E-01   WLM_-3_1
                                                3.30E-01   WL_-3_0

    9        8.71E-08     5.25       94.75      2.00E-06   MF=help_-1_1
                                                4.00E-01   V_-3_0
                                                3.30E-01   WLM_-3_0
                                                3.30E-01   WL_-3_0

   10        8.71E-08     5.25       100.00     2.00E-06   VF=help_-1_1
                                                4.00E-01   V_-3_0
                                                3.30E-01   WLM_-3_0
                                                3.30E-01   WL_-3_0
 End of output
```