Automated Safety Analysis by Minimal Path Set Detection for Multi-Domain Object-Oriented Models

Christian Schallert

Institute of System Dynamics and Control, German Aerospace Centre (DLR), Christian.Schallert@dlr.de

Abstract

This paper describes, exemplifies and substantiates a method for detection of the minimal path set of any fault-tolerant technical system that is represented as a multi-domain object-oriented model. Thus, the method automatically performs a safety or reliability analysis of the system.

Keywords: safety analysis, reliability analysis, minimal path set, graph algorithms, modelling of failures, failure probability

1 Introduction

Safety and reliability are essential in transport aircraft design and operation, as well as other technical areas. Safety analyses are therefore an inherent part of the complex process of aircraft and on-board systems development. In systems development, multi-domain object-oriented modelling and simulation have now become the state-of-the-art.

This paper describes a method that integrates safety or reliability analysis with multi-domain object-oriented modelling. In essence, the method automatically detects the minimal path set of any fault-tolerant technical system. The method is based on the simulation of normal behaviour, degradation and failure of a system. Thus, modelling of failures is supplemented to component models from generic libraries, e.g. the Modelica Standard Library, that typically represent only normal, intact behaviour.

Other approaches to automated safety or reliability analysis based on multi-domain object-oriented modelling exist. A model-based diagnosis approach has been described by (Bunus, Lunde, 2008) that uses constraints (inequalities) instead of differential equations. It is particularly dedicated to diagnosing systems, i.e. detecting and isolating faults. Another approach described by (Papadopoulos *et al.*, 2001) performs semi-automatic fault-tree synthesis based on fault annotations included in the components of a system model.

The method described in this paper differs from the existing approaches, in so far that it uses differential-algebraic equations and modelling of failures. It thus permits the conducting of all other simulation studies that initially motivated the implementation of a model, as well as it ensures a consistent safety analysis due to the modelling, not just annotating, of failures. The goal of the method is to improve the development process of fault-tolerant, safety-critical systems.

2 Modelling Approach

This section refers to the approach selected for the modelling of fault-tolerant systems and the additions necessary to enable automated safety analysis.

2.1 Modelling of Failures

The proposed minimal path set detection method requires that failure of a system can be simulated in addition to its normal behaviour. Thus, the modelling has to be supplemented by equations that reflect failures of system components and, if applicable, by operating logics that determine how a system reacts to the occurrence of component failures.

Model parameter values are changed in order to represent a failure. In doing so, the model equations remain the same (structure-invariant approach). Corresponding examples of aircraft on-board system models including component failures, e.g. electrical open circuit, mechanical disconnection or loss of hydraulic pressure, are provided in (Schallert, 2008, 2011, 2014). The proposed detection method activates component failures by directly accessing the relevant model parameters. Alternatively, a universal fault triggering network described by (van der Linden, 2014) can be used for activation of failures.

Provided that the preconditions (see subsection 3.1.2) are met, the detection method can be used also if the structure of the model equations is changed to represent failures. Such a structure-variant, multi-mode approach is described by (Elmqvist *et al.*, 2014).

Component failure rates λ_i are stored in each component model that includes failures. Since the λ_i values are used only for post-processing (see equation 2), they can be inserted also as custom annotations; a concept described by (Zimmer *et al.*, 2014).

2.2 Indication of System Status

Safety or reliability assessment requires the analyst to define criteria that indicate if a system operates normally or if it fails. Such criteria have to be

implemented in a system model, in order to compute a `sysOp` output signal that indicates system operation.

In case of a flight control surface actuation system, such as described in (Schallert, 2014), `sysOp` is computed by comparing the actual position or rate of the controlled surface with the command (model input). The capability of the system to follow commands is simulated by the minimal path set detection method for various combinations of intact and failed components. In doing so, `sysOp` is evaluated for correlation with the respective component states.

3 A Method for Minimal Path Set Detection

In this section a method is described that solves the problem of determining the failure probability of a system by detecting its minimal path set. The method is called DMP. It draws on a representation of the system model object structure as a graph and on simulation. Minimal path set analysis generally assumes that a system and its components are two-state, intact or failed, as explained in section 2.3 of (Birolini, 2007).

The DMP method is a state space simulation. The state space, in this context, denotes the set of all combinations of intact and failed components of a system to be examined for detection of its minimal path set. Evaluation of the system graph reduces the size of the state space and hence the number of simulations required.

3.1 Definitions and Preparations

3.1.1 Definitions

Definitions are provided of the terms used in the following for the DMP method:

Set. A defined collection of distinct objects, e.g. the components of a system.

Subset. `A` is a subset of `B`, `A` ⊆ `B`, if every object of `A` is also an object of `B`, e.g. {1, 2, 3} ⊆ {1, 2, 3}. If `A` is a subset of but unequal to `B`, then `A` is a proper subset of `B`, `A` ⊂ `B`, e.g. {1, 2} ⊂ {1, 2, 3}.

Superset. `A` is a superset of `B`, `A` ⊇ `B`, if every object of `B` is also an object of `A`, e.g. {1, 2, 3} ⊇ {1, 2, 3}. If `A` is a superset of but unequal to `B`, then `A` is a proper superset of `B`, `A` ⊃ `B`, e.g. {1, 2, 3} ⊃ {1, 2}.

Difference set. `A` \ `B` denotes the set of elements that are members of `A` but not of `B`, e.g. {1, 2, 3} \ {2} = {1, 3}, or {1, 2, 3} \ {4} = {1, 2, 3}.

Component. A distinct element of a system. In this paper, components are also called nodes.

Combination. A set of intact components of a system.

Path. A set of intact components that causes a system to operate.

S-T path. A Source-to-Target path in a graph.

Path set. A set of paths of a system.

Minimal path. A path that cannot be reduced without causing system failure.

Minimal path set. The set of all minimal paths of a system.

Graph. A representation of a set of objects, e.g. the components (nodes) of a system, and of the connections between them.

Node. An object in a graph. Nodes are also called components in this paper.

Edge. A link that connects a pair of nodes in a graph.

Articulation. A node in a graph (or path) that, if removed, disconnects the graph (or path) into several subgraphs.

Subgraph. A part of a graph whose set of nodes and set of edges are subsets of those of the graph, the set of edges being restricted to the subset of nodes.

Density. The density d of a graph is generally, e.g. in (Diestel, 2010), defined by

$$d(N,E) = 2E/N(N-1) \tag{1}$$

where N and E denote the numbers of nodes and edges of the graph, respectively.

Probability computation. The probability of system operation or failure is computed from the system's minimal path set in applying the reliabilities of its components. Let C_i denote the intact state of component i. Then, the probability of occurrence P of a minimal path MP is, see (Meyna, Pauli, 2003),

$$P(MP) = P(C_1 \wedge C_2 \wedge \dots) \qquad \forall C_i \in MP$$
$$P(MP) = \prod_{C_i \in MP} P(C_i) = \prod_{C_i \in MP} R_i(t), \quad R_i = e^{-\lambda_i t} \tag{2}$$

with the component reliabilities R_i, failure rates λ_i and exposure time t. Exponentially distributed lifetimes are assumed. Other lifetime models, e.g. Weibull distribution, can be used as well. The probability of system operation $R_{sys}(t)$ is computed from the probabilities of the minimal paths by

$$R_{sys}(t) = P(MP_1 \vee \dots \vee MP_r)$$
$$= \sum_{j=1}^{r} P(MP_j) - \sum_{j=1}^{r-1} \sum_{k=j+1}^{r} P(MP_j \wedge MP_k) + \dots \tag{3}$$
$$+(-1)^{r+1} \cdot P(MP_1 \wedge MP_2 \wedge \dots \wedge MP_r)$$

where r is the number of all minimal paths in the set. Equation 3 is evaluated for illustration at the end of subsection 3.2.2.

3.1.2 Properties of Minimal Paths and Requirements for Detection

This subsection explains the assumptions and requirements that apply to the minimal path set detection method DMP described in section 3.2:

1. The system behaves monotonously. This refers to a system that operates if all its components are intact and fails if all components fail. If the system

DOI 10.3384/ecp15118565

operates while not all components are intact, it continues operating if any further component becomes intact. Conversely, if the system has failed, it remains failed if any further component fails. This definition of monotony is common in safety analysis. For instance, it can be found in section 14.2 of (Meyna, Pauli, 2003).

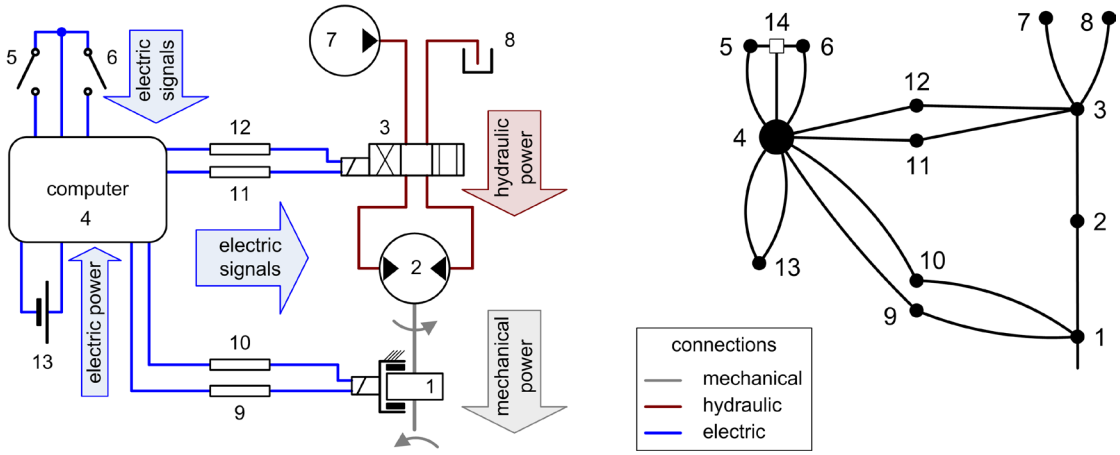

Figure 1. Exchange of power and signals across the edges in a coherent set of nodes

2. Every real world component is represented by one model object and by one node in a corresponding graph. No component is represented by two or more model objects or nodes.

DMP relies on a representation of the object structure of the system model as a graph. Nodes of the graph represent components, and edges the connections between components. The establishing of a graph is described in subsection 3.1.3. The properties of minimal paths, and in particular their situation in the graph, are explained in the following, which then proceeds to further requirements for DMP.

Depending on the system model and, if applicable, the marking of sources (S) and targets (T) in the corresponding graph, some S-T paths are minimal paths. This is true, for instance, for the electric network models shown in (Schallert, 2008, 2011), where also related detection methods are described. In general, however, what is known is only that a minimal path consists of one or more connected nodes.

This is explained by Figure 1 that depicts a part of an aircraft's flight control surface actuation system model and its accompanying graph. The edges of the graph correspond to the interfaces that exchange power, material or signals among the components (nodes) of a system. This exchange among neighboured nodes enables a system to operate. No other nodes are situated between any of those nodes that exchange power, material or signals and hence belong to a minimal path. Thus, only a coherent set of nodes can be a minimal path. The following defines a coherent set of nodes:

Definition 1. A set of nodes in a graph is coherent if any two nodes of the set are connected through a series of edges and through only those nodes that belong to the set.

Figure 2 shows coherent and incoherent sets of nodes (marked blue) for illustration. An S-T path, such as (c), is a special case of a coherent set of nodes.

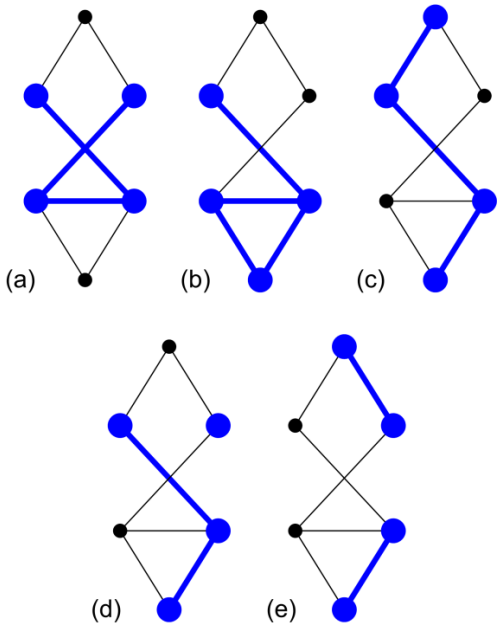

Figure 2. Coherent (a), (b), (c) and incoherent sets of nodes (d), (e) in a graph

Coherence (interconnection) of intact nodes in the system graph is a precondition for a minimal path. Removing a node from a minimal path interrupts the exchange of power, material or signals among the nodes of the minimal path. If no other minimal paths exist, the system fails. If the system operates with an incoherent set of intact nodes, nodes can be removed from the set, i.e. fail, without interrupting the exchange of power etc. Such a set of nodes is therefore a path but not a minimal path. Thus, the third assumption for method DMP is:

3. Only a coherent set of intact nodes in a system graph can be a minimal path.

Because not every coherent set of intact nodes is a minimal path, the system model is simulated to determine which ones are actually minimal paths.

3.1.3 Graph Representation of Multi-Domain Object-Oriented Models

A graph is defined by its adjacency list (array `AL`). In `AL`, each row corresponds to a node of the graph. The neighbours of a node are stored in the respective rows of `AL`, as will be illustrated. If more than one connection exists between two components of a model, this is reflected by a single edge in the graph. (That is, in each row of `AL`, any node is stored not more than once.) It is only relevant that any two nodes of the graph are connected, but it is not important whether the two nodes are connected by one or more than one edge. Additionally, the interface types are not evaluated by method DMP, so they are not reflected in the graph.

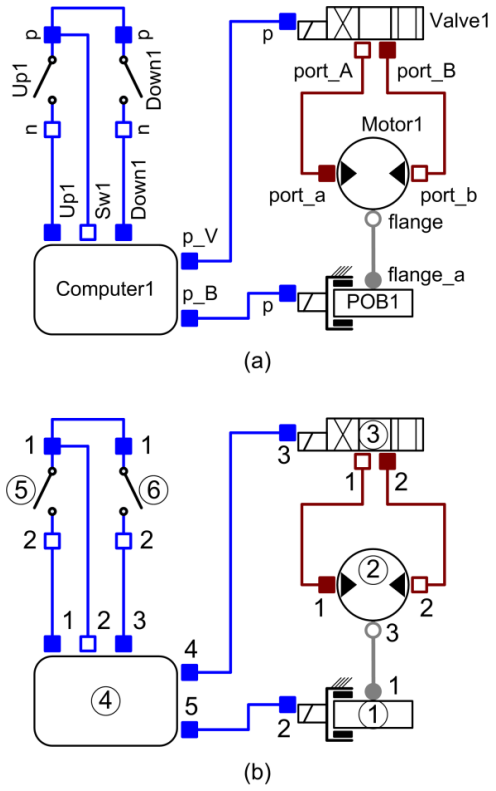

Figure 3. Components and connections in a multi-domain object-oriented model

For illustration, the adjacency list is indicated for a part of the system model depicted in Figure 1. Figure 3 (a) shows the component (node) and interface names, and the indices in (b). The numbering of nodes – encircled in (b) – corresponds to Figure 1. The algorithm that actually prepares an adjacency list is described in subsection 3.1.3 of (Schallert, 2015).

The connections via mechanical flanges, hydraulic ports, electric pins etc. are declared in the model by the `connect()` statements below. They are expressed in terms of the component and interface names (left column), and in terms of component and interface indices (right column).

1.	`connect(POB1.flange_a, Motor1.flange);`	`(1.1, 2.3)`
2.	`connect(Motor1.port_a, Valve1.port_A);`	`(2.1, 3.1)`
3.	`connect(Motor1.port_b, Valve1.port_B);`	`(2.2, 3.2)`
4.	`connect(POB1.p, Computer1.p_B);`	`(1.2, 4.5)`
5.	`connect(Valve1.p, Computer1.p_V);`	`(3.3, 4.4)`
6.	`connect(Computer1.Sw1, Up1.p);`	`(4.2, 5.1)`
7.	`connect(Up1.p, Down1.p);`	`(5.1, 6.1)`
8.	`connect(Computer1.Up1, Up1.n);`	`(4.1, 5.2)`
9.	`connect(Computer1.Down1, Down1.n);`	`(4.3, 6.2)`

A special case occurs if more than one node is directly or indirectly connected to one and the same interface of a node, as happens for the 6th and 7th connections of the example: `Computer1.Sw1 (4.2)` is connected to `Up1.p (5.1)`, and in turn `Up1.p (5.1)` is connected to `Down1.p (6.1)`. Actually, there is a direct connection between `(4.2)` and `(6.1)`. It only appears to be indirect, across `(5.1)`, because each `connect()` statement links exactly two nodes. To reflect that a direct connection exists between `(4.2)` and `(6.1)`, an auxiliary node `(14)` is introduced. Auxiliary nodes do not represent any real or model object; rather, they are introduced to ensure that coherent sets of nodes are correctly detected by method DMP. An auxiliary node is stored as an additional row in the adjacency list `AL`.

Table 1 specifies the adjacency list by the node indices. Figure 4 shows the corresponding graph.

Table 1. Adjacency list `AL` for Figure 3 (b)

1	2	4			
2	1	3			
3	2	4			
4	1	3	5	6	14
5	4	14			
6	4	14			
14	4	5	6		

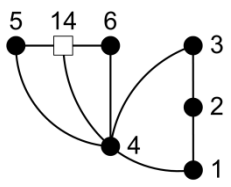

Figure 4. Graph for Figure 3 (b)

DOI 10.3384/ecp15118565

3.2 Detection of Minimal Paths

Method DMP is capable of detecting the minimal path set if conditions 1, 2 and 3 defined in subsection 3.1.2 are fulfilled. The detection starts with all system components (nodes) intact. Nodes are then successively removed from the system graph, which corresponds to component failures. The model is simulated to identify if the system still operates or fails.

Articulations can occur in the graph that, if removed, cause disconnection of the graph into several subgraphs. Since only a coherent set of intact nodes can be a minimal path, splitting up the graph at articulations reduces the state space and thus the number of simulations. The lower the density of a system graph is, the more articulations occur within it and thus fewer simulations are required. For completeness, method DMP allows that articulations can also belong to a minimal path.

3.2.1 Detection Algorithm

Figure 5 shows a flow chart of the detection algorithm. It consists of a preparation phase (steps DMP.1 - 4) and the actual, iterative detection process (steps DMP.5 - 17). Steps DMP.3, 8, 11 - 16 refer to lower level algorithms that are described in detail, including code, in (Schallert, 2015). The meaning of the symbols used is as follows:

`nr`	number of all components that can fail of a system
n_{Loop}	iteration counter of detection process
`rn`	node(s) to be removed from a path of array PS_{prev}
`PS`, PS_{prev}	arrays of path sets in the actual and previous iteration, respectively, of the detection process
`isMinPS`, $isMinPS_{prev}$	Boolean arrays that store if a path in array `PS` or PS_{prev} is minimal
`np`, np_{prev}	number of paths stored in `PS` and PS_{prev}
`SF`	array for storing combinations that cause system failure
`nsf`	number of combinations stored in array `SF`

In the preparation phase, the necessary data are retrieved from the system model (step DMP.1). Then, the model is simulated to check if the system operates for the set of initially intact components (nodes). To this end, the model output `sysOp` is evaluated. A monotonous system will operate, and the procedure is continued only in this case (step DMP.2). If the system fails, no minimal path can be detected, and the process is aborted. Next, a graph (adjacency list) of the system model is established (step DMP.3, see 3.1.3). Then, several arrays are initialised (step DMP.4) for the detection process.

At the start of an iteration, the paths detected so far, their number, as well as the information whether they are minimal are assigned to PS_{prev}, np_{prev} and $isMinPS_{prev}$. Arrays `PS`, `isMinPS` and the counter `np` are reset (step DMP.5). Then, n_{Loop} is increased by one. Next, combinations are generated from the paths in PS_{prev}. If the i^{th} path, denoted by PS_{prev}[i, :], is minimal (checked in step DMP.7), then it is not further reduced, because any subset of a minimal path causes system failure. If the i^{th} path is not minimal, then all subsets are generated that remove one intact node `rn` from the path (step DMP.8): PS_{prev}[i, :] \ {rn} for all rn ∈ PS_{prev}[i, :] and rn ∈ {1, nr}. If node `nr` is an articulation of path PS_{prev}[i, :], then the corresponding subgraphs of PS_{prev}[i, :] are generated. Articulations and subgraphs are determined by an algorithm based on depth-first search described by (Tarjan, 1972). Along with each subgraph, the non-articulations of PS_{prev}[i, :] that also belong to the respective subgraph are stored. This information is used later, in step DMP.13, to generate combinations that remove two or more non-articulations from a path, dependent on the simulation result (step DMP.11). Due to monotony of the system, any subset of a path is generated only if it is not a subset of any combination stored in `SF` that causes system failure. Thus, if no subset is generated from path PS_{prev}[i, :] in step DMP.8, then the system fails for every subset of this path; it is minimal and is marked by $isMinPS_{prev}$[i] := `true`. The generation of subsets of paths ends after every path in PS_{prev} has been processed, i.e. i > np_{prev} (step DMP.10).

Next (step DMP.11), the model is simulated for every generated combination in order to determine if the system is operating. From the simulation result (`sysOp`), it is first determined which paths in PS_{prev} are minimal. If a path is minimal, it is stored in `PS` and marked as minimal in `isMinPS`. Then, dependent on whether they cause system operation or failure, the combinations are stored either in `PS` or in `SF`, and the respective counter (`np` or `nsf`) is increased (step DMP.12).

For those paths in `PS` that were established due to an articulation and that are no superset of any other path, combinations are generated that remove two or more non-articulations from the original path in PS_{prev} (step DMP.13). This is necessary since articulations can also belong to a minimal path. The system model is then simulated for the generated combinations. Dependent on the simulation result, a combination is stored either in `PS` or `SF` (steps DMP.14 and 15).

Next, array `PS` is tidied up by deleting those paths that are a superset of any other path (step DMP.16). A path can be minimal only if it is not a superset of any other path. If every of the `np` paths in `PS` is marked as minimal (step DMP.17), the detection is complete and

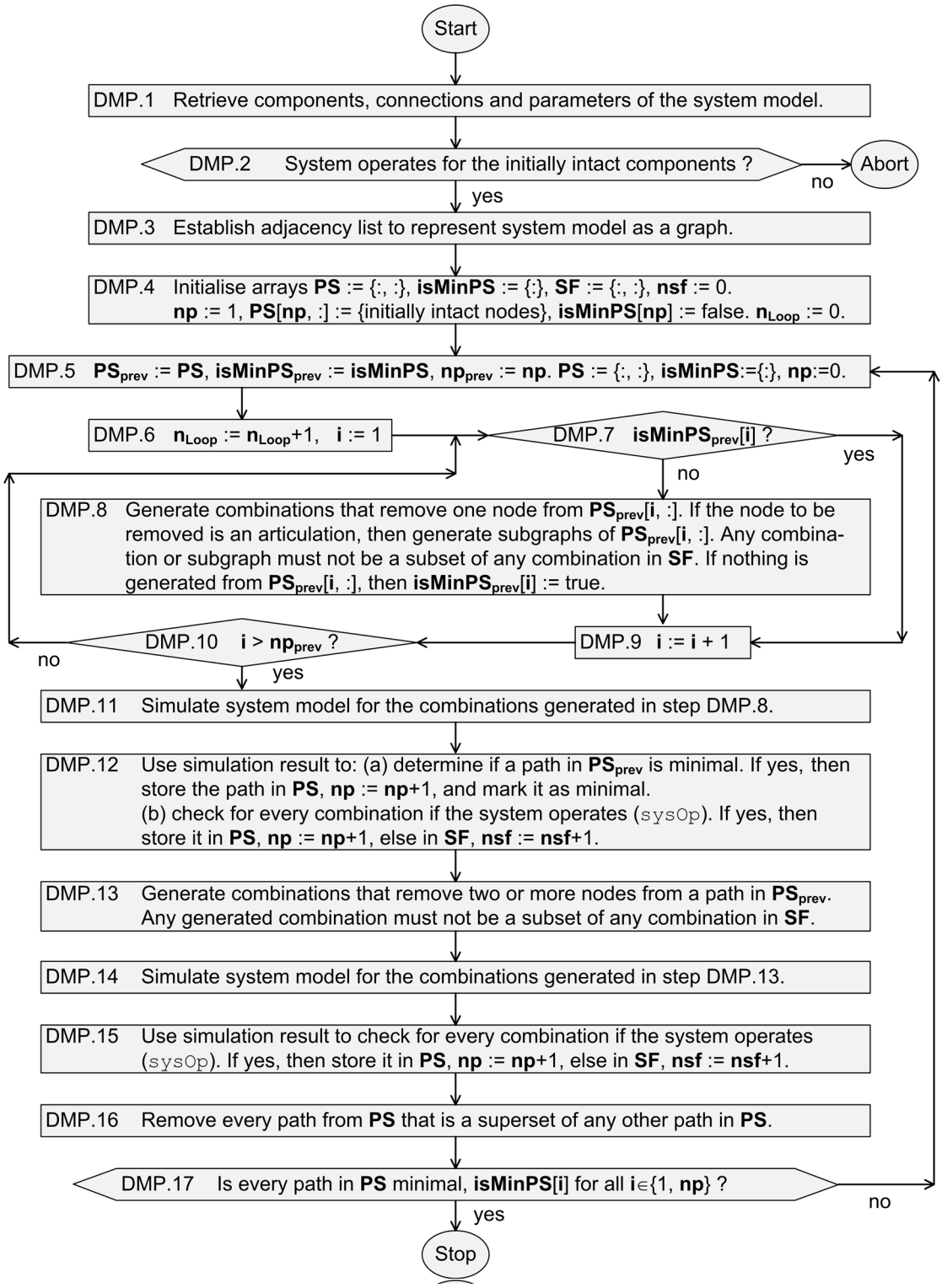

Figure 5. Flow chart of minimal path set detection algorithm DMP

the process ends. Otherwise, the process continues with a new iteration at step DMP.5.

3.2.2 A Minimal Path Set Detection Example

The detection algorithm DMP is illustrated by means of the example graph shown in Figure 6. It is assumed that this graph is deduced from the object-oriented model of any technical system. The minimal path set is assumed as PS = {{1, 2, 3}, {4, 5, 6}, {1, 3, 4, 6, 7}}.

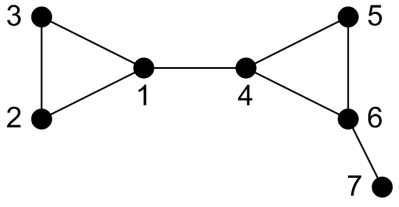

Figure 6. An example graph

DOI 10.3384/ecp15118565

The meaning of the symbols used is as follows, as yet not defined:

isArt	indicates if rn is an articulation of the respective path
comb	generated combination of intact nodes
simC	array of generated combinations, input for simulations of the system model
sysOp	array of simulation result (system operates or not) for every combination in simC
nonArt	non-articulation nodes of a path in PS_{prev} that also belong to a generated subgraph

The detection proceeds as follows. In the tables, column "row" indicates the progress of the algorithm in terms of "n_{Loop}" - "number of comb", e.g. 0-1 denotes combination 1 of iteration 0 ($n_{Loop} = 0$).

In the preparation phase it is checked if the system operates when all its components are intact (step DMP.2 in Figure 5). Thus, the simulation input simC is as indicated below in Table 2. At this initial stage, no path has yet been detected and PS_{prev} is empty.

Table 2. Combinations tested (by simulation of system model) at initial stage of detection process

row	PS_{prev}	rn	isArt	comb	comb stored in simC		sysOp	nonArt
					no	yes		
0-1	-	-	-	{1, 2, 3, 4, 5, 6, 7}	-	x	x	-

The system operates, so the set of initially intact nodes is stored as a single, non-minimal path (np = 1) in PS (step DMP.4). SF is empty (nsf = 0), Table 3.

Table 3. Path set after initial stage of detection process

PS	isMinPS
{1, 2, 3, 4, 5, 6, 7}	-

SF
-

The process continues with iteration one ($n_{Loop} = 1$). The path data are assigned to PS_{prev}, $isMinPS_{prev}$ and np_{prev}. PS, isMinPS and np are reset (step DMP.5). Combinations are then generated from path $PS_{prev}[1, :]$ as follows (step DMP.8): Node 1 is an articulation. The path splits into two subgraphs {2, 3} and {4, 5, 6, 7} due to the removal of node 1. The non-articulations of the original path $PS_{prev}[1, :]$ that also belong to the respective subgraphs are {2, 3} and {5, 7}. Node 2 is not an articulation, thus a combination is generated by removing node 2 from $PS_{prev}[1, :]$, and likewise for nodes 3, 5 and 7. Altogether, ten combinations are generated for simulation of the system model, Table 4.

The simulation result of step DMP.11 (column sysOp) indicates that path $PS_{prev}[1, :]$ is not minimal, because the system operates for subsets of it, namely for those in rows 1-2, 1-3, 1-4, 1-5, 1-7, 1-8 and 1-10. These combinations are stored as paths in PS, np = 7. The other combinations in rows 1-1 and 1-6 are stored in SF, nsf = 2 (step DMP.12). The one in row 1-9, {7}, is not stored in SF as it is a subset of {5, 6, 7}.

At this stage, two of the seven paths in PS are not supersets of any other path, namely rows 1-2 and 1-5 in Table 4 (marked bold). Other paths can exist that include some of the articulations of the original path $PS_{prev}[1, :]$. To assure that such paths are detected, further combinations that are no superset of any path in PS - in this case {4, 5, 6, 7} and {1, 2, 3} - must be generated (step DMP.13). Such combinations remove as many non-articulations from the original path as non-superset paths were deduced from it, namely two (rows 1-2 and 1-5) in the case of $PS_{prev}[1, :]$. To avoid generating supersets, one node of every set of non-articulations, {5, 7} and {2, 3}, is removed from the original path, respectively. Thus, the combinations $PS_{prev}[1, :] \setminus \{2, 5\}$, $PS_{prev}[1, :] \setminus \{2, 7\}$, $PS_{prev}[1, :] \setminus \{3, 5\}$ and $PS_{prev}[1, :] \setminus \{3, 7\}$ are generated, as listed in rows 1-11 through 1-14, Table 5.

Due to the simulation result, three more paths are stored in PS, np = 7 + 3 = 10, and one more combination in SF, nsf = 2 + 1 = 3 (steps DMP.14 and 15). The total number of simulations so far is n_{sim} = 1 + 10 + 4 = 15. Supersets of paths are removed from PS, which leads to np = 5 paths remaining (in Table 6) after completion of step DMP.16.

Table 6. Path set PS and combinations that cause system failure SF, as existent after 1st iteration of process

PS	isMinPS
{1, 2, 3}	-
{1, 2, 4, 5, 6}	-
{1, 3, 4, 5, 6}	-
{1, 3, 4, 6, 7}	-
{4, 5, 6, 7}	-

SF
{1, 2, 4, 6, 7}
{2, 3}
{5, 6, 7}

Since none of the paths in PS is marked as minimal (step DMP.17), the process continues with a second iteration ($n_{Loop} = 2$). The path data are assigned to PS_{prev}, $isMinPS_{prev}$ and np_{prev}. PS, isMinPS and np are reset (step DMP.5). Then, combinations are generated (step DMP.8) from each of the $np_{prev} = 5$ paths in PS_{prev} as listed in Table 7. Three combinations are generated from $PS_{prev}[1, :] = \{1, 2, 3\}$, but only one is stored in simC for simulation. The other two are not stored in simC because they are a subset of a combination in SF, as indicated in rows 2-1 and 2-3. If a combination causes system failure, every subset of it causes system failure as well due to system monotony.

Any combination is stored only once in simC, as indicated in row 2-12, for instance. Eight combinations are stored altogether for simulation in step DMP.11.

The simulation result (column sysOp in Table 7) indicates that PS_{prev}[1, :] = {1, 2, 3} and PS_{prev}[4, :] = {1, 3, 4, 6, 7} are minimal, because the system fails for every respective subset. These paths are stored in PS and marked as minimal in isMinPS (step DMP.12). In addition, two non-minimal paths, {4, 5, 6} in row 2-5 and {1, 4, 5, 6} in row 2-6, are stored in PS; the latter will be removed in step DMP.16.

It is not necessary in this iteration to generate combinations that remove two or more non-articulations from any path in PS_{prev}. The reason is: At most one subgraph that causes system operation is deduced from any path in PS_{prev}. In the case of PS_{prev}[2, :] = {1, 2, 4, 5, 6}, subgraphs {2}, {4, 5, 6} and {1, 2}, {5, 6} are generated due to articulations 1 and 4, respectively. The system operates only for {4, 5, 6}. In order to generate every combination from PS_{prev}[2, :] that is no superset of {4, 5, 6}, it is sufficient to remove one non-articulation from PS_{prev}[2, :]. These combinations are generated already in step DMP.8, as Table 7 shows (rows 2-9 and 2-10).

Thus, np = 4 paths are stored in PS of which two are minimal. nsf = 3 + 3 = 6 combinations are stored in SF. The total number of simulations so far is n_{sim} = 15 + 8 = 23. np = 3 paths remain in PS (see Table 8) after removal of supersets in step DMP.16.

Table 8. Path set PS and combinations that cause system failure SF, as existent after 2nd iteration of process

PS	isMinPS
{1, 2, 3}	x
{1, 3, 4, 6, 7}	x
{4, 5, 6}	-

SF
{1, 2, 4, 5}
{1, 2, 4, 6, 7}
{1, 3, 4, 5}
{1, 3, 4, 6}
{2, 3}
{5, 6, 7}

Since not all paths are marked as minimal, the process enters a third iteration, n_{Loop} = 3. Again, the path data are assigned to PS_{prev}, $isMinPS_{prev}$ and np_{prev}. PS, isMinPS and np are reset. Then, combinations are generated only for the non-minimal path PS_{prev}[3, :] = {4, 5, 6} in step DMP.8. As Table 9 shows, every generated combination is a subset of any combination in SF. This means that PS_{prev}[3, :] is also minimal, and no further simulations are necessary. Thus, the process is complete. The minimal path set of the example system (Figure 6) is correctly detected:

Table 10. Minimal path set detected after 3rd iteration

PS	isMinPS
{1, 2, 3}	x
{1, 3, 4, 6, 7}	x
{4, 5, 6}	x

Next, the probability of system operation (or failure) is computed. For illustration, equation 3 is evaluated for the detected minimal path set. With the component reliabilities $R_i = R_i(t)$, the probabilities of the minimal paths are:

$$P(PS_1) = R_1R_2R_3, \qquad P(PS_2) = R_1R_3R_4R_6R_7,$$
$$P(PS_3) = R_4R_5R_6.$$

For the 2nd order intersections, the probabilities are:

$$P(PS_1 \wedge PS_2) = R_1R_2R_3R_4R_6R_7,$$
$$P(PS_1 \wedge PS_3) = R_1R_2R_3R_4R_5R_6,$$
$$P(PS_2 \wedge PS_3) = R_1R_3R_4R_5R_6R_7.$$

The probability of the single 3rd order intersection is:

$$P(PS_1 \wedge PS_2 \wedge PS_3) = R_1R_2R_3R_4R_5R_6R_7.$$

Employing these products, equation 3 reads

$$R_{sys}(t) = R_1R_2R_3 + R_1R_3R_4R_6R_7 + R_4R_5R_6$$
$$-(R_1R_2R_3R_4R_6R_7 + R_1R_2R_3R_4R_5R_6$$
$$+R_1R_3R_4R_5R_6R_7) + R_1R_2R_3R_4R_5R_6R_7.$$

If it is assumed that $\lambda_i = 10^{-2}$/h and t = 1h, thus $R = R_i$ = 0.990, then the probability of system operation is $R_{sys}(1h) = 2R^3 + R^5 - 3R^6 + R^7 = 0.99922$ or likewise, the probability of system failure is $F_{sys}(1h) = 1 - R_{sys}(1h) = 7.8 \cdot 10^{-4}$.

3.2.3 Proof and boundary effort of detection method

The minimal path set detection method DMP gives a complete result when applied to any multi-domain object-oriented system model that fulfills the conditions 1, 2 and 3 stated in subsection 3.1.2. In addition, the method is finite which means that it terminates when applied to any such model. The completeness and finiteness are proven in the following. The upper and lower bounds of the required computing effort are also derived.

Completeness. Consider a detection method that merely exploits the monotony of the analysed system. It starts with a set of all nodes as the initial path. Every combination is generated that removes a single node from the path. It is tested for each (by simulation of the model) if the system operates. Those combinations that cause system operation constitute a set of paths. In the set of paths, only those are kept that are no superset of any other, because only a non-superset path can be minimal. Thus, a complete path set of the system exists at the end of an iteration of the detection method.

A next iteration is entered. All combinations are generated that remove a single node from every path in the set of the previous iteration. In so doing, subsets of combinations that cause system failure are omitted. Due to monotony, if a combination causes system failure, the system remains failed if any node is removed from that combination. Again, testing the generated combinations leads to a complete set of paths, a next iteration is entered with all non-superset paths of the set, and so on. At some point it is found that the system fails on removal of any node from a path. Such a path is minimal by definition. The method

DOI 10.3384/ecp15118565

continues reducing the other paths until all in the set of paths are minimal. Since every path was reduced by a single node from one iteration to the next, it is obvious that the method gives the complete minimal path set.

In addition to exploiting system monotony, method DMP benefits from the fact that only a coherent set of intact nodes can be a minimal path. Evaluation of the system graph hence reduces the number of simulations required for minimal path set detection.

A path is split into subgraphs at the articulations of the path. In this way, subgraphs are generated for every path that exists in the path set of the previous iteration of DMP. If two or more subgraphs of a path cause system operation, all combinations are generated that remove as many non-articulations from that path, as subgraphs were deduced from it that cause system operation. (Two or more nodes are removed, because all combinations that reduce a path by one of its non-articulations have been generated and tested in a preceding step.) In so doing, for the generated combinations to be no superset of any of the subgraphs of that path, each non-articulation removed from the path belongs to exactly one of its subgraphs. Thus, if any such combination causes system operation, it exists in the path set at the end of an iteration of DMP. It follows that the path set at the end of any iteration is complete, and hence method DMP is complete.

Finiteness. DMP commences with a set of all nodes of a monotonous system as the initial path. It tests if the system still operates for subsets of a (the initial or other) path. Every subset removes one or more nodes from a path. Nodes are never added to a path from one iteration to the next. DMP repeats gradually removing nodes until the system fails for all subsets of a path, i.e. if that path is minimal. For those paths not yet identified as minimal, subsets of them are generated until they be reduced no further without causing system failure, i.e. until every path is minimal. Then, the process ends. If every single node of a system constitutes a minimal path, the process ends after all nodes are failed. Due to monotony, a system fails if all its nodes fail. Thus, method DMP clearly terminates.

Effort. For illustration of the highest computing effort, consider a complete system graph that includes no articulations. Removal of any node gives a subsequent smaller complete graph. In addition, consider that every single node of the *nr* nodes of the system graph constitutes a minimal path. Then, the number of simulations in each iteration n_{Loop} of DMP is the binomial $\mathbf{C}(nr,\ n_{Loop})$. DMP iterates until all nodes are failed, i.e. $n_{Loop} = nr$. The total number of simulations is thus $\sum_{n_{Loop}=0}^{nr} \mathbf{C}(nr,\ n_{Loop}) = 2^{nr}$, the same as a "brute force" approach needs.

The lowest effort occurs if a system operates only with all its nodes intact (single minimal path that comprises all nodes of the system). Irrespective of the density of the system graph, DMP runs until iteration $n_{Loop} = 1$ is completed. The total number of simulations is hence $\sum_{n_{Loop}=0}^{1} \mathbf{C}(nr,\ n_{Loop}) = 1 + nr$.

The number of simulations required by DMP is thus bound by the upper limit 2^{nr} and lower limit $1 + nr$. Between these bounds, the actual effort depends on the density of the system graph, the number of minimal paths and number of nodes thereof. The more effort is saved, the lower the density of a system graph is.

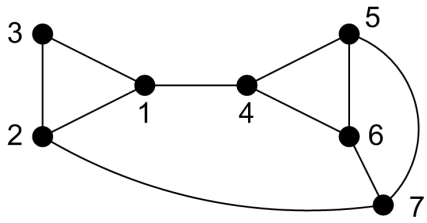

Figure 7. An example graph, higher density than Figure 6

For illustration, Table 11 lists the number of simulations n_{sim} for three detection cases. The number of edges and density (equation 1) of the respective graphs are denoted by *E* and *d*. The number of nodes is $N = nr = 7$ for all cases. n_{sim} is stated for method DMP, as well as a method that exploits the system "monotony only" (no evaluation of the graph), as described. Case 2 corresponds to the example in subsection 3.2.2. Case 1 relates to the same graph, but the system has one less minimal path. Case 3 assumes the same minimal path set as case 2, but the graph has a higher density (Figure 7). The comparison shows that the fewer minimal paths exist and the lower the density of the graph, the smaller effort required by DMP. With an increasing number of minimal paths and graph density, the effort of DMP approaches that of the "monotony only" method. A "brute force" method neither evaluates the monotony of a system nor its graph; it thus requires $2^{nr} = 128$ simulations for each case.

Table 11. Comparison of effort of minimal path set detection for three cases

case	system graph	*E*	*d*	PS	n_{sim}	
					monotony only	DMP
1	Figure 6	8	0.381	{1, 2, 3}, {4, 5, 6, 7}	35	15
2	Figure 6	8	0.381	{1, 2, 3}, {4, 5, 6}, {1, 3, 4, 6, 7}	40	23
3	Figure 7	10	0.476	{1, 2, 3}, {4, 5, 6}, {1, 3, 4, 6, 7}	40	34

4 Conclusions

This paper contributes a method called DMP for detection of the minimal path set of any fault-tolerant system that is represented as a multi-domain object-oriented model. DMP can be employed throughout the system development process to keep the safety analysis up-to-date with design iterations. This is meaningful particularly if multi-domain object-oriented modelling is used already in systems engineering. DMP enhances the scope of application of a model while permitting all other simulation studies that originally motivated implementation of the model to be conducted.

DMP belongs to the class of state-space simulations. Evaluation of the system graph reduces the number of simulations required, thus ensuring feasibility of DMP. It has been successfully tested on large, realistic models of safety relevant aircraft systems, as described in (Schallert, 2015).

It must be beared in mind that all model-based safety analysis methods capture only those phenomena that are covered in the modelling. A model is always an abstraction of a real system and might hence be incomplete. Then again, the DMP method ensures that all relevant failure conditions, at least in so far as modelled, are captured.

Acknowledgements

This research has received funding from the European Union's 7th Framework Programme (FP7/2007-2013) for the CleanSky Joint Technology Initiative under grant agreement CSJU-GAN-SGO-2008-001.

References

A. Birolini. *Reliability Engineering – Theory and Practice* (Fifth Edition). Springer-Verlag Berlin Heidelberg, 2007.

P. Bunus, K. Lunde. Supporting Model-Based Diagnostics with Equation-Based Object-Oriented Languages. *Proceedings of the 2nd International Workshop on Equation-Based Object-Oriented Languages and Tools (EOOLT)*, pp. 121-130, Paphos, Cyprus, 2008.

R. Diestel. *Graph Theory (Graduate Texts in Mathematics)*, Springer-Verlag, 2010.

H. Elmqvist, S. E. Mattsson, M. Otter. Modelica extensions for Multi-Mode DAE-Systems. *Proceedings of the 10th International Modelica Conference*, pp. 183-193, Lund, Sweden, 2014. doi: 10.3384/ECP14096183

A. Meyna, B. Pauli. *Taschenbuch der Zuverlässigkeits- und Sicherheitstechnik*. Carl Hanser Verlag München Wien, 2003. In German.

C. Schallert. Incorporation of Reliability Analysis Methods with Modelica. *Proceedings of the 6th International Modelica Conference*, pp. 103-112, Bielefeld, Germany, 2008.

C. Schallert. Inclusion of Reliability and Safety Analysis Methods in Modelica. *Proceedings of the 8th International Modelica Conference*, pp. 616-627, Dresden, Germany, 2011. doi: 10.3384/ECP11063616

C. Schallert. A Safety Analysis via Minimal Path Sets Detection for Object-Oriented Models. *Safety and Reliability: Methodology and Applications (editors: Nowakowski et al.)*, CRC Press/Balkema, ISBN: 978-1-315-73697-6, 2014.

C. Schallert. *Integrated Safety and Reliability Analysis Methods for Aircraft System Development using Multi-Domain Object-Oriented Models*, 2015 (to appear).

R. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2), pp. 146-160, 1972.

F. van der Linden. General fault triggering architecture to trigger model faults in Modelica using a standardized blockset. *Proceedings of the 10th International Modelica Conference*, pp. 427-436, Lund, Sweden, 2014. doi: 10.3384/ECP14096427

Y. Papadopoulos, J. McDermid, R. Sasse, G. Heiner. Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. *Reliability Engineering and System Safety*, Vol. 71, pp. 229 - 247, 2001.

D. Zimmer, M. Otter, H. Elmqvist, G. Kurzbach. Custom Annotations: Handling Meta-Information in Modelica. *Proceedings of the 10th International Modelica Conference*, pp. 173-182, Lund, Sweden, 2014. doi: 10.3384/ECP14096173

Table 4. Combinations tested during 1st iteration of detection process

row	PS_{prev}	rn	isArt	comb	comb stored in simC		sysOp	nonArt
					no	yes		
1-1	{1, 2, 3, 4, 5, 6, 7}	{1}	x	{2, 3}	-	x	-	{2, 3}
1-2		{1}	x	**{4, 5, 6, 7}**	-	x	x	{5, 7}
1-3		{2}	-	{1, 3, 4, 5, 6, 7}	-	x	x	-
1-4		{3}	-	{1, 2, 4, 5, 6, 7}	-	x	x	-
1-5		{4}	x	**{1, 2, 3}**	-	x	x	{2, 3}
1-6		{4}	x	{5, 6, 7}	-	x	-	{5, 7}
1-7		{5}	-	{1, 2, 3, 4, 6, 7}	-	x	x	-
1-8		{6}	x	{1, 2, 3, 4, 5}	-	x	x	{2, 3, 5}
1-9		{6}	x	{7}	-	x	-	{7}
1-10		{7}	-	{1, 2, 3, 4, 5, 6}	-	x	x	-

DOI 10.3384/ecp15118565

Table 5. Combinations tested during 1st iteration that remove two non-articulations from $PS_{prev}[1, :]$

row	PS_{prev}	rn	comb	comb stored in simC		sysOp
				no	yes	
1-11	{1, 2, 3, 4, 5, 6, 7}	{2, 5}	{1, 3, 4, 6, 7}	-	x	x
1-12		{2, 7}	{1, 3, 4, 5, 6}	-	x	x
1-13		{3, 5}	{1, 2, 4, 6, 7}	-	x	-
1-14		{3, 7}	{1, 2, 4, 5, 6}	-	x	x

Table 7. Combinations tested during 2nd iteration of detection process

row	PS_{prev}	rn	isArt	comb	comb stored in simC		sysOp	nonArt
					no	yes		
2-1	{1, 2, 3}	{1}	-	{2, 3}	⊆{2, 3}	-	-	-
2-2		{2}	-	{1, 3}	-	x	-	-
2-3		{3}	-	{1, 2}	⊆{1, 2, 4, 6, 7}	-	-	-
2-4	{1, 2, 4, 5, 6}	{1}	x	{2}	⊆{1, 2, 4, 6, 7}	-	-	-
2-5		{1}	x	{4, 5, 6}	-	x	x	{5, 6}
2-6		{2}	-	{1, 4, 5, 6}	-	x	x	-
2-7		{4}	x	{1, 2}	⊆{1, 2, 4, 6, 7}	-	-	-
2-8		{4}	x	{5, 6}	⊆{5, 6, 7}	-	-	-
2-9		{5}	-	{1, 2, 4, 6}	⊆{1, 2, 4, 6, 7}	-	-	-
2-10		{6}	-	{1, 2, 4, 5}	-	x	-	-
2-11	{1, 3, 4, 5, 6}	{1}	x	{3}	⊆{2, 3}	-	-	-
2-12		{1}	x	{4, 5, 6}	exists in simC	-	x	-
2-13		{3}	-	{1, 4, 5, 6}	exists in simC	-	x	-
2-14		{4}	x	{1, 3}	exists in simC	-	-	-
2-15		{4}	x	{5, 6}	⊆{5, 6, 7}	-	-	-
2-16		{5}	-	{1, 3, 4, 6}	-	x	-	-
2-17		{6}	-	{1, 3, 4, 5}	-	x	-	-
2-18	{1, 3, 4, 6, 7}	{1}	x	{3}	⊆{2, 3}	-	-	-
2-19		{1}	x	{4, 6, 7}	⊆{1, 2, 4, 6, 7}	-	-	-
2-20		{3}	-	{1, 4, 6, 7}	⊆{1, 2, 4, 6, 7}	-	-	-
2-21		{4}	x	{1, 3}	exists in simC	-	-	-
2-22		{4}	x	{6, 7}	⊆{5, 6, 7}	-	-	-
2-23		{6}	x	{1, 3, 4}	-	x	-	{3}
2-24		{6}	x	{7}	⊆{5, 6, 7}	-	-	-
2-25		{7}	-	{1, 3, 4, 6}	exists in simC	-	-	-
2-26	{4, 5, 6, 7}	{4}	-	{5, 6, 7}	⊆{5, 6, 7}	-	-	-
2-27		{5}	-	{4, 6, 7}	⊆{1, 2, 4, 6, 7}	-	-	-
2-28		{6}	x	{4, 5}	-	x	-	{4, 5}
2-29		{6}	x	{7}	⊆{5, 6, 7}	-	-	-
2-30		{7}	-	{4, 5, 6}	exists in simC	-	x	-

Table 9. 3rd iteration of detection process (no combinations tested)

row	PS_{prev}	rn	isArt	comb	comb stored in simC		sysOp	nonArt
					no	yes		
3-1	{4, 5, 6}	{4}	-	{5, 6}	⊆{5, 6, 7}	-	-	-
3-2		{5}	-	{4, 6}	⊆{1, 3, 4, 6}	-	-	-
3-3		{6}	-	{4, 5}	⊆{1, 2, 4, 5}	-	-	-