SAFEST

Static/Dynamic Fault Trees
& Reward Event Trees Analysis Tool

www.safest.dgbtek.com

User Manual

Version 1.0.0

Dr. Falak Sher,
Asst. Prof. Matthias Volk,
Prof. Joost-Pieter Katoen,
Prof. Mariëlle Stoelinga

Prepared by
DGB Technologies LLC
Wyckoff, NJ 07481, USA

DGB Technologies is a multimission company specializing in formal verification tools and technologies for stochastic analysis of mission and life-critical systems.

DGB TECHNOLOGIES

DGB Technologies LLC
Risk & Reliability Analysis
Wyckoff, NJ 07481, USA

Static/Dynamic Fault Trees & Reward Event Trees Analysis Tool
www.safest.dgbtek.com

Dr. Falak Sher,
DGB Technologies LLC
Asst. Prof. Matthias Volk,
Eindhoven University
Prof. Joost-Pieter Katoen,
RWTH Aachen University
Prof. Mariëlle Stoelinga,
Twente University

Abstract

SAFEST is a powerful tool for modeling and analyzing both static and dynamic fault trees as well as reward event trees. Dynamic fault trees (DFTs) extend standard fault trees by providing support for faithfully modeling spare management, functional dependencies, and order-dependent failures. Reward event trees (RETs) extend traditional event trees with state rewards as well as non-deterministic decision-making at states, thus providing upper and lower bounds on the analysis results.

The SAFEST tool provides an efficient and powerful analysis of DFTs/RETs via probabilistic model checking – a rigorous, automated analysis technique for probabilistic systems. The backbone of the analysis is based on efficient state space generation. Several optimization techniques are incorporated, such as exploiting irrelevant failures, symmetries, and independent modules. Probabilistic model checking allows us to analyze the resulting state space with respect to a wide range of measures of interest. In addition, an approximation approach is provided that builds only parts of the state space and allows to iteratively refine the computations up to the desired accuracy.

The SAFEST tool allows embedding DFTs within RETs thus providing probabilities of transition branches of RETs. It provides a graphical user interface for creating, generating, simulating, simplifying, and visualizing the results of fault trees/event trees.

SAFEST is a state-of-the-art analysis tool, as demonstrated by an experimental evaluation and comparison with existing tools. A variety of case studies, including vehicle guidance systems, train operations in railway station areas, and energy systems such as (nuclear) power plants have been done. This document explains how to use the SAFEST.

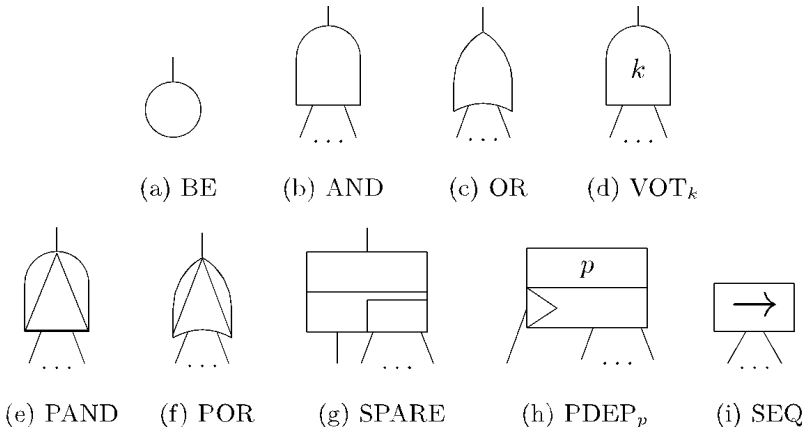
Contents

1. Introduction

Probabilistic risk assessment (PRA) is of critical importance to ensure the safe and reliable operation of today's systems in areas such as transportation, infrastructure, power generation, space exploration, etc. Fault trees and event trees are popular models in PRA and are recommended by many standards and regulatory bodies.

Dynamic Fault Trees (DFTs)

While standard (or static) fault trees (SFT) are widely used and well supported by PRA tools, their modeling capabilities are limited. Several extensions to fault trees have been proposed



over the years to overcome these limitations. Dynamic fault trees (DFT) were introduced by Dugan and are one of the most prominent extensions. DFTs introduce new gate types that allow for more faithful modeling by providing explicit support for spare management, functional dependencies, and order-dependent failures. DFT models have been successfully applied for e.g., aerospace systems, autonomous driving, railway engineering, and analysis of spacecraft via the COMPASS toolset.

Reward Event Trees (RETs)

Reward event trees extend standard event trees with reward models for states as well as non-deterministic decision-making at states. Unlike standard event trees, the analysis of RETs not only provides the probabilities/frequencies of different outcomes/consequences (for an initiating/accidental event) but also provides upper and lower bounds on the results in the case of decision-making. Furthermore, it allows the evaluation of expected values of different quantities of interest e.g. radionuclide emission, etc. Details about event trees can be read at the link.

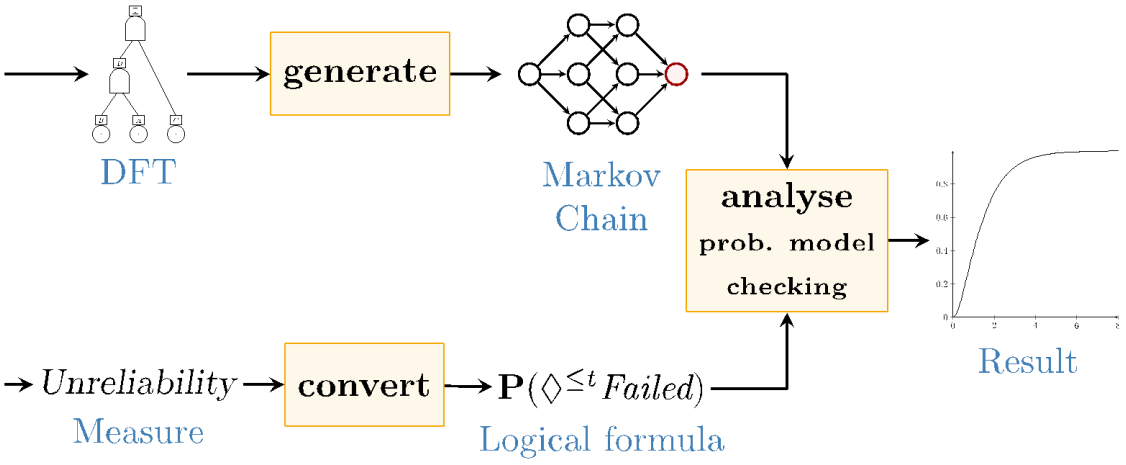
We have developed SAFEST, a modern, state-of-the-art tool for modeling and analyzing both standard (static) and dynamic fault trees and event trees. SAFEST comes with a web-based graphical user interface that allows efficient modeling, visualization, simplification, and interactive simulation of fault trees using a graphical editor. Moreover, it allows embedding

DFTs within RETs thus providing probabilities of transition branches of RETs. The analysis of dynamic fault trees/event trees is enabled via an efficient translation to Markov models and the use of state-of-the-art techniques from probabilistic model checking.

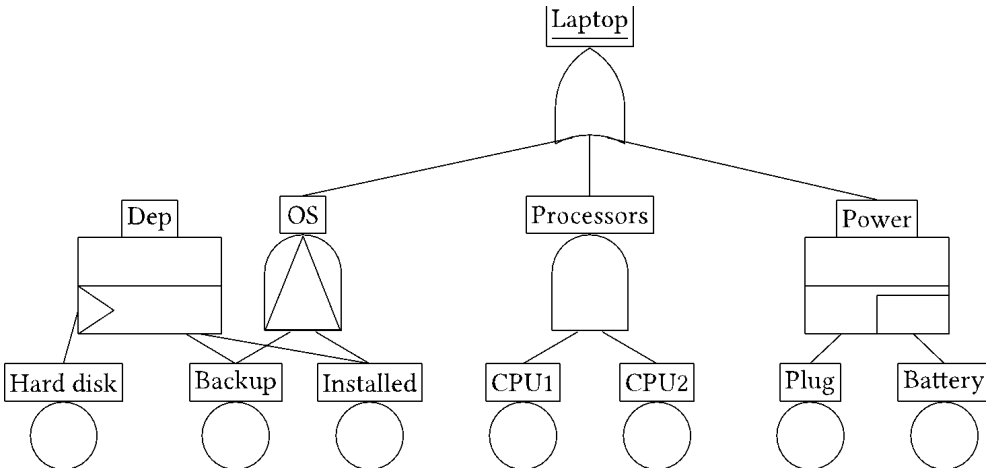
Probabilistic Model-checking

Model checking is a rigorous technique for checking whether a given model satisfies a specification given as a logical formula. Model checking uses highly optimized techniques to efficiently analyze the state space. Probabilistic model checking considers probabilistic systems that capture random behavior, such as Markov models. The approach uses tailored numerical algorithms and answers queries such as “What is the probability of a system failure within a year?” or “What is the expected time to failure when the system has entered a degraded state?”.

We apply probabilistic model checking to DFT/DET analysis. The input is a DFT/DET and a measure of interest, for instance, the unreliability or the mean-time-to-failure (MTTF). From the DFT/DET, a Markov chain is generated. The measure is converted to a logical formula. Both the Markov chain and the logical formula are fed into a probabilistic model checker which computes the results.



The use of probabilistic model checking has several advantages for the DFT/DET analysis. First, model checking supports a wide range of complex logical formulas that can be checked out of the box. Thus, we use a one-for-all analysis approach instead of having to develop algorithms tailored to each type of measure. Second, our approach uses model checking as a black box. This means we can easily change the underlying analysis tools and directly incorporate and benefit from recent advances in model-checking algorithms without changing the overall approach. Third, unlike approaches based on Monte Carlo simulation, probabilistic model checking yields exact results and, in particular, is agnostic to rare events.



The key innovation of our approach is an efficient generation of the state space to combat possible state space explosion. To this end, we developed several optimizations that exploit irrelevant failures of events, symmetric structures, and independent modules in the DFT. Furthermore, we incorporated efficient analysis techniques for static fault trees based on binary decision diagrams (BDD). Finally, we developed an approximation approach based on only building the most relevant parts of the state space. All of these techniques allow for efficient analysis of DFTs.

Our experimental evaluation shows that our tool significantly outperforms existing DFT analysis tools and can handle DFTs with several hundred elements. We have demonstrated the performance of our tool and the modeling capabilities of DFTs in several industrial case studies. Examples include DFT models for vehicle guidance systems and analyzing the impact of infrastructure failures on train operations in railway station areas.

There are two main modules of the SAFEST tool:
Fault Tree Analysis Module: It allows the building of failure models of different failure scenarios of systems that may arise during the life cycle of a system.
Bowtie Analysis Module: This allows the building of bowtie models that allow for analysis of the probabilities/frequencies of different outcomes/consequences based on an initiating/accidental event.

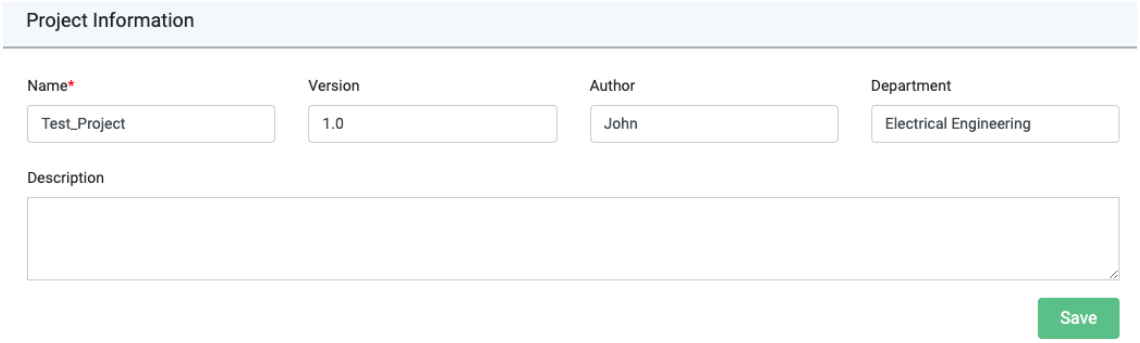
2. SAFEST Project

In the SAFEST tool, multiple failure scenarios (fault trees) of a system as well as bowtie models (event trees) can be created under the Fault Tree Analysis and Bowtie Analysis modules.

Creation

Click on “New Project” in the File menu of the toolbar. The following window will appear.

Fill up mandatory fields, and click the “Save” button. The following page will appear, where users can make any changes if required.



Open

Click the “Open Project” in the File menu of the toolbar to open an already existing SAFEST project from your drive.

Export

Click the “Export Project” in the File menu of the toolbar to export the current SAFEST project on your drive.

View

Project views can be selected from the View menu of the toolbar. There are two types of views:

- Simple View. It is for simple users. Under this view, a user cannot create Failure events in models. Reliability metrics can only be computed for top-level events (TLE or system_failed). Moreover, only basic reliability measures like *reliability/unreliability*, *mean time to failure,* and *average failure probability per hour* can be computed for the TLE.
- Advance View. It is for advanced users or researchers. This view gives the full functionality of the SAFEST tool.

Help

In the Help menu of the toolbar, we have:

- Documentation: It contains a link to the user manual of the SAFEST tool, as well as the grammar for expressions used in the models as parameters.
- Activation key: Here you can add a license key and activate the SAFEST tool functionality.

Status

In the status bar, the information about the view of the currently selected project is shown along with the status of the Analysis engine, which is either Running or Stopped.

3. Fault Tree Analysis Module

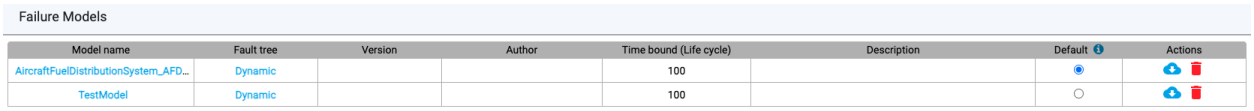
It contains all failure models (fault trees) along with their parameter sets, metrics, and computing methods.

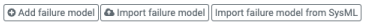
Failure Models

Each failure model comprises a fault tree and its failure events.

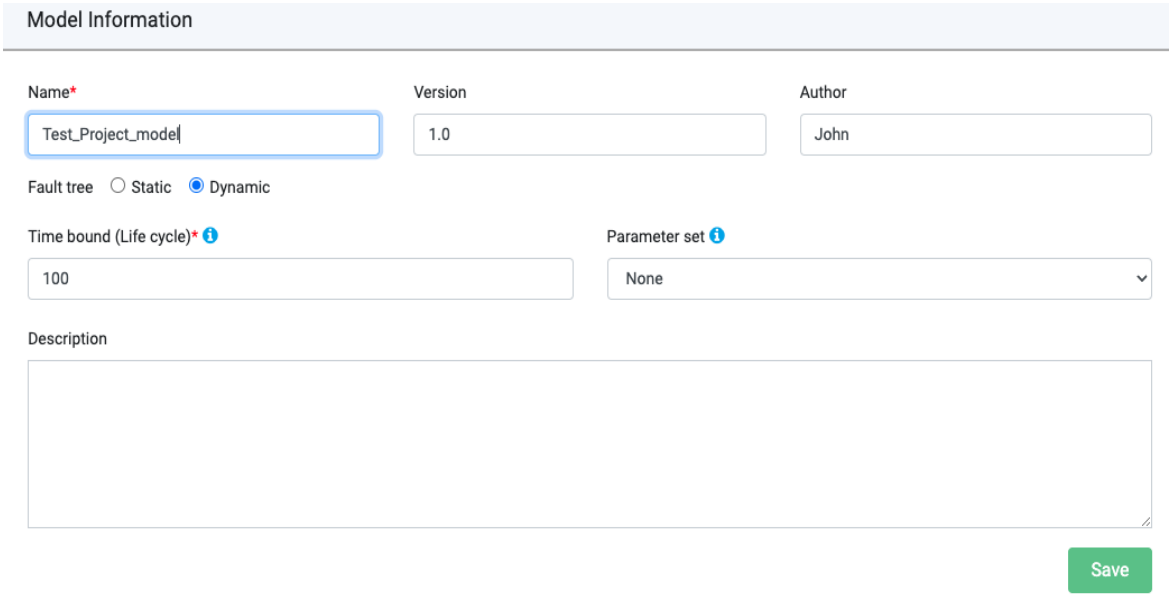
View

Click on the “Failure Models” under “Fault Tree Analysis” in the left panel to display all failure models in a table. A failure model that is worked upon the most can be selected as a default model by selecting the corresponding radio button.





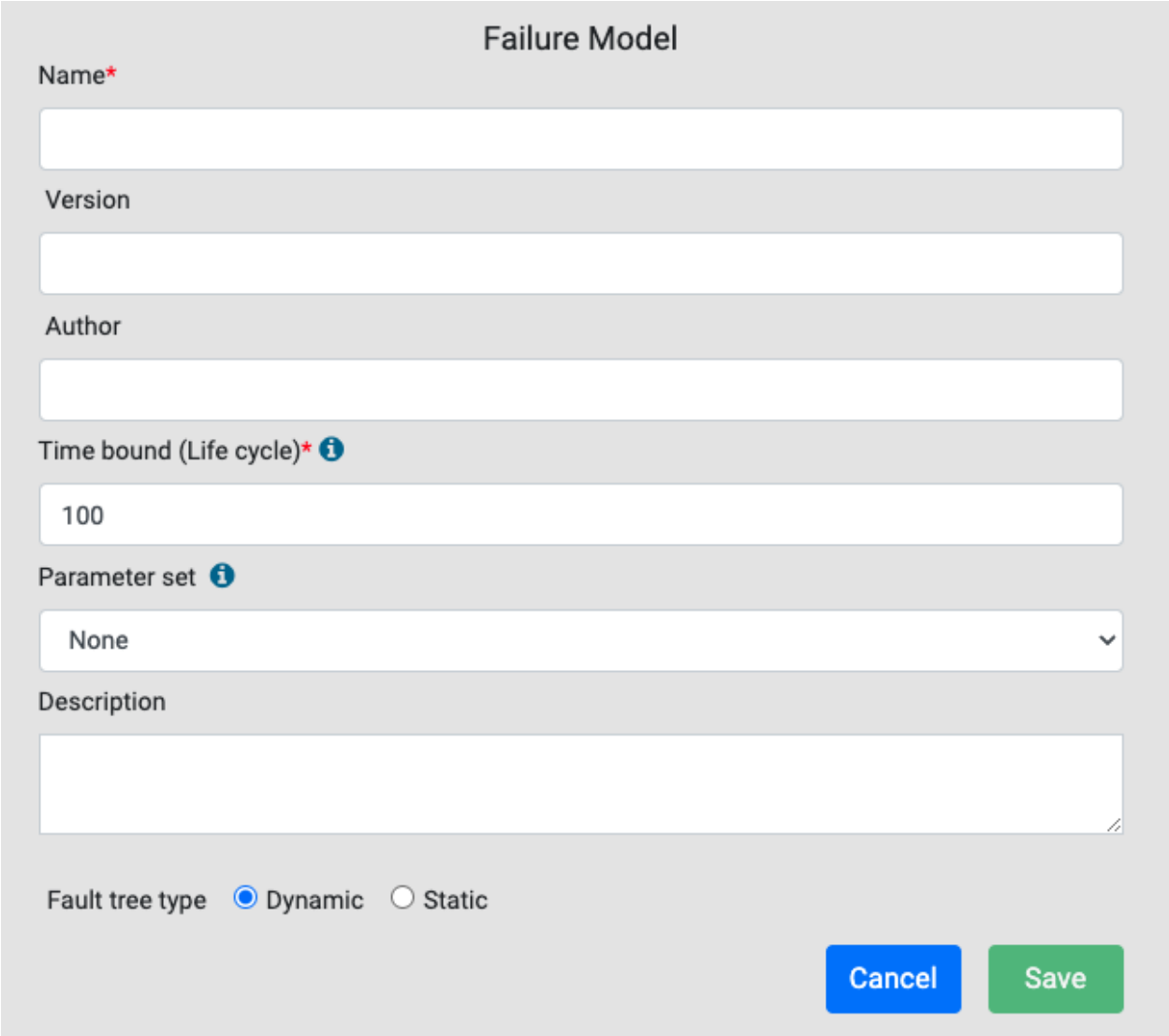
Click on the failure model name in the left panel to see details about the model.



Change fields and click the “Save” button.

Creation

Click on the “Add failure model” button on the “Failure Models” page to create a new model.

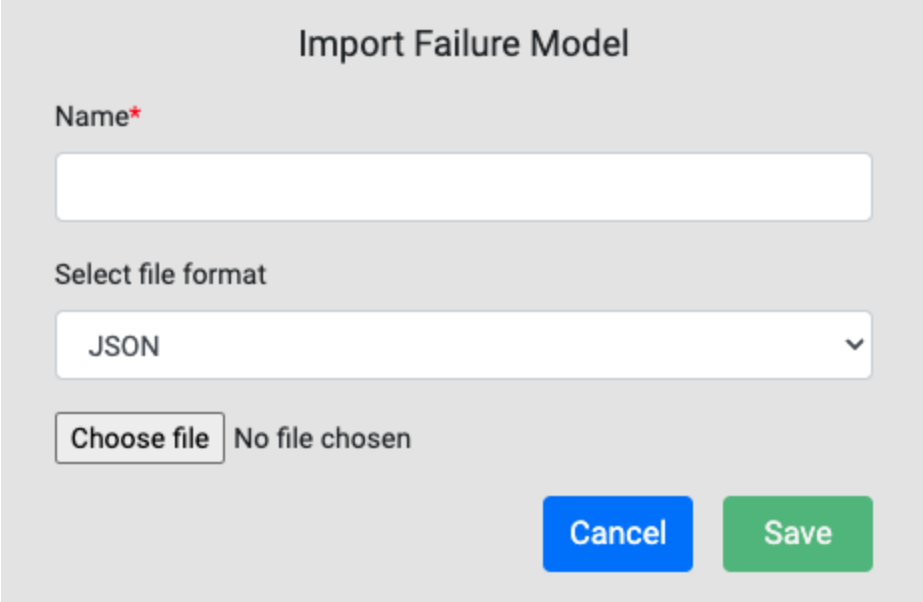


- A time bound for which the model is to be analyzed may be inserted. This value can be changed at the time of analysis as well.
- A parameter set in which parameters to be used in the model are defined may be selected. It can also be changed at the time of analysis.
- The type of fault tree can be selected as “Static” or “Dynamic”.

  Fill up the mandatory fields and click the “Save” button.

Import

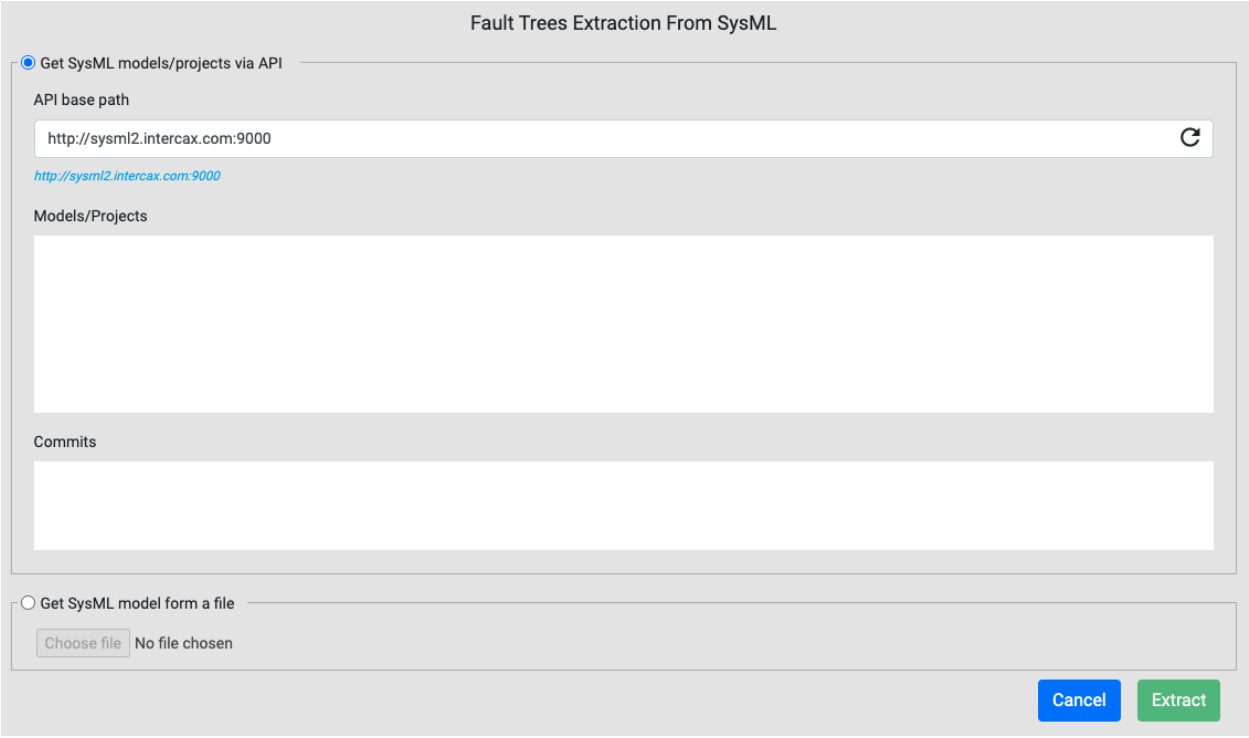
Click on the “Import failure model” button on the “Failure Models” page to import an already created model.



Select the format of the model and a file from the drive. Click the “Save” button. We support “JSON” “Galileo” and “Failure Model” formats. Note that Failure Model (.fm) is the format in which we allow to import/export of failure models in the SAFEST tool.

Import models from SysML models:

Users have the option to extract failure models from SysML 2.0 models. The SysML model has to be annotated with safety information in order to generate DFTs automatically out of it. Click the “Import failure model from SysML” button on the “Failure Models” page.



The user has two options to extract DFTs out of SysML models

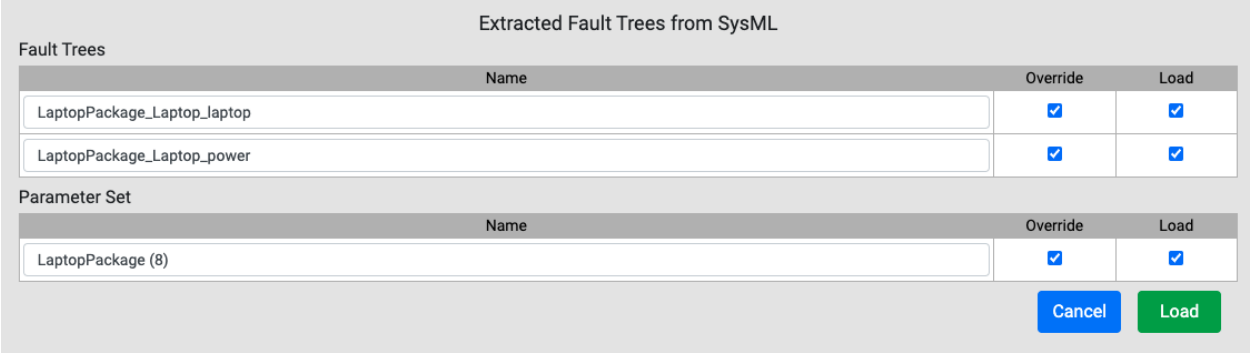
Get the SysML model from a file

Compile a SysML model, which is annotated with safety information – please read the **Annotate SysML Models with Safety Information section** below for further details on how to annotate SysML models with safety information in order to prepare them for automatic extraction of DFTs out of them, in a Jupyter notebook. And run the following command to export the model in JSON format. **Currently, we support the latest version – v0.33.0 – of SysML 2.0:**

```
%export <package_name>
```

After downloading, DFTs can be extracted from it. Select the radio button “Get SysML model from a file”, select the downloaded file (in JSON format), and click the “Extract” button.

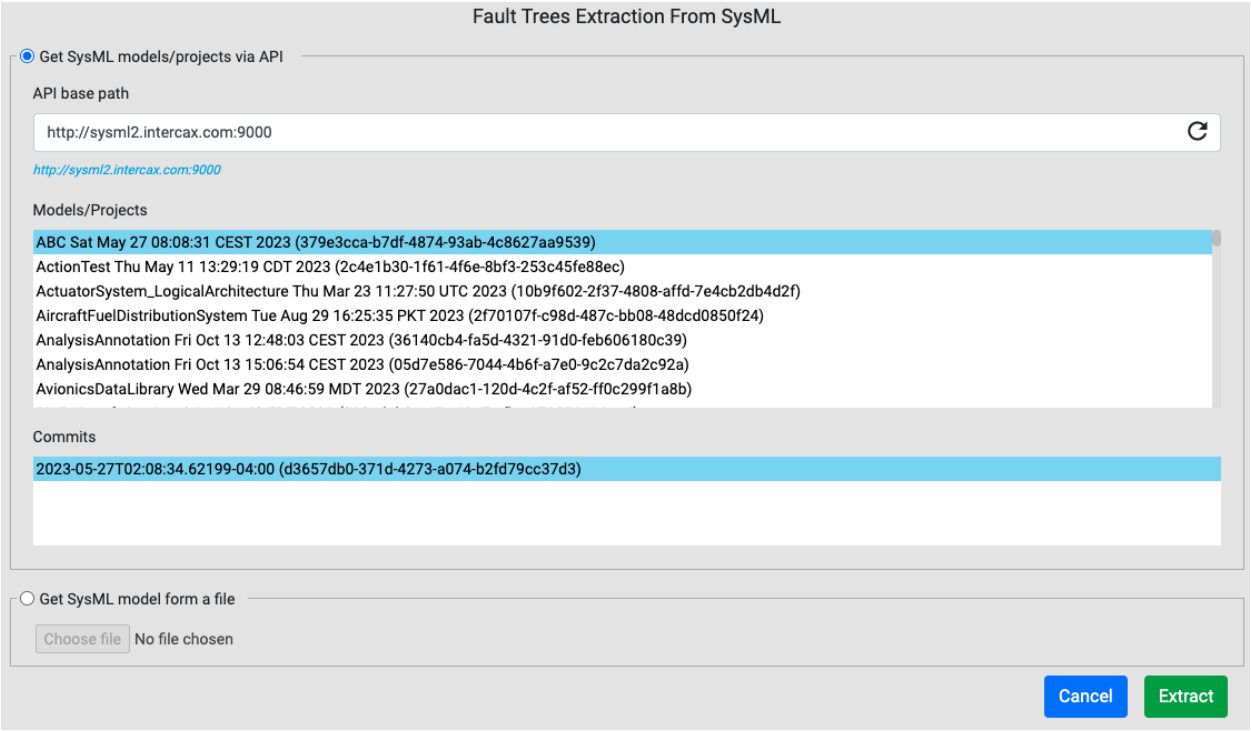
The following popup will appear that contains fault trees of all failure scenarios that have been mentioned in the SysML model. It also contains parameters that have been given in the SysML mode e.g. failure rates of components inside the SysML.



The user can select the failure models as well as parameters to be included in the project.

Get the SysML model/project via API

SysML projects are normally uploaded to some central repository. One has the option to connect to that repository using its API and upload the model from there. Select the “Get SysML models/projects via API” radio button, enter a path in the “API base path” and click the circular arrow.

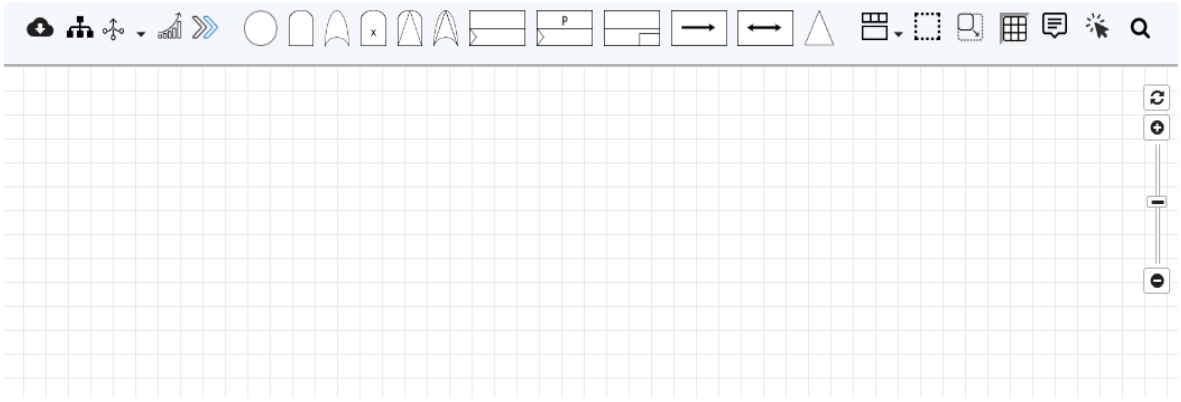


Select a project and its commit, and then click the “Extract” button to upload the SysML model. The rest of the steps are the same as above. For testing purposes, we have uploaded our “LaptopPackage” project to the above repository.

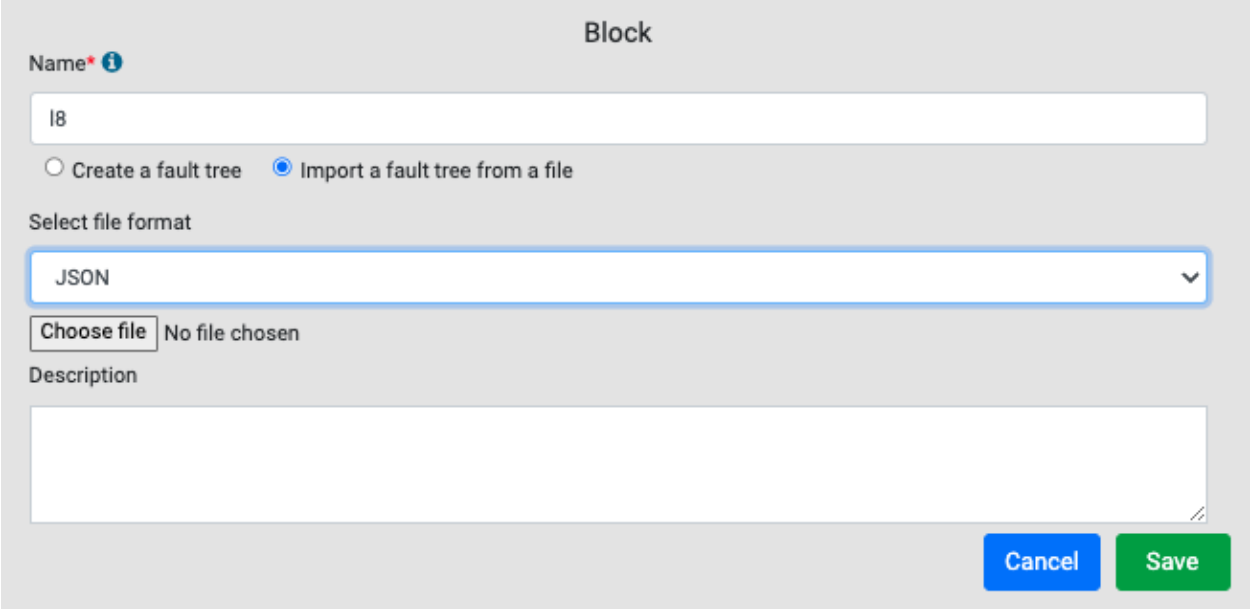
The algorithm will automatically generate fault trees and show them on the “Failure Models” page. Please read **Annotate SysML Models with Safety Information section** below for further details.

Update

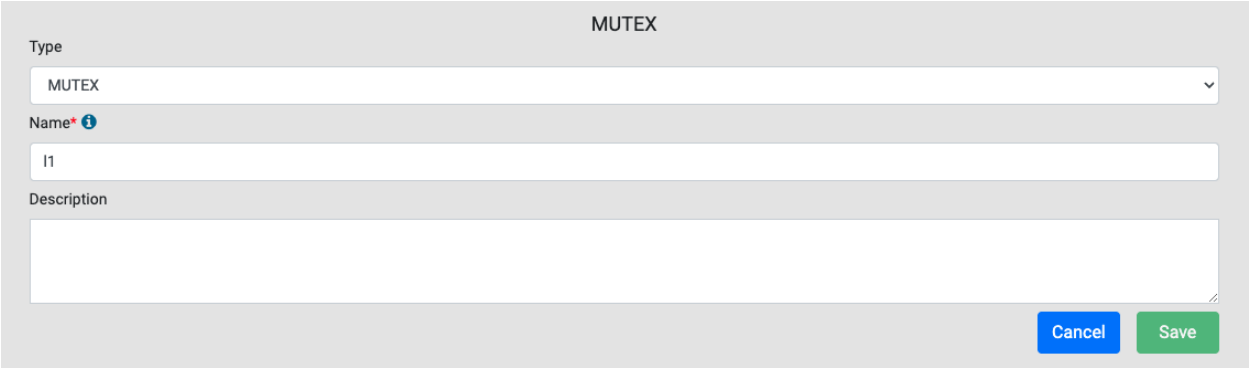
Click on the “Fault Tree” of a failure model to open a drag-and-drop grid.

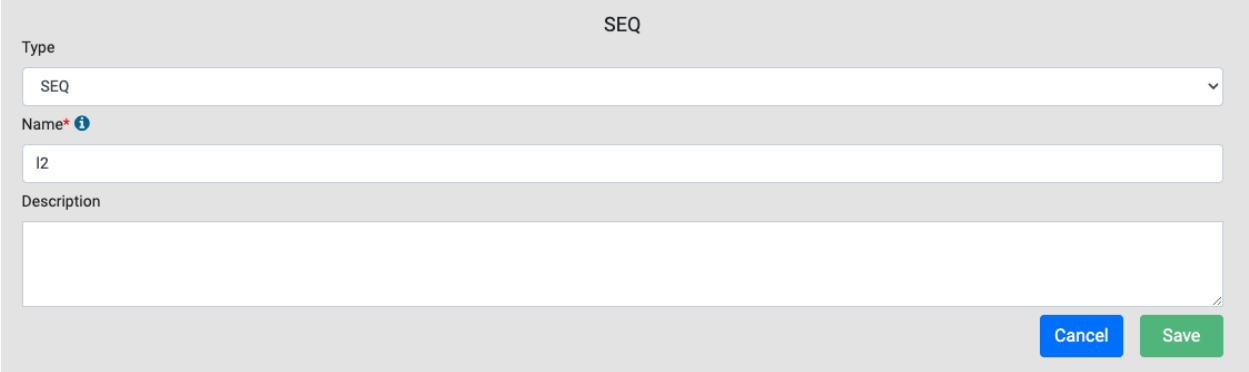


- Draw your tree by dragging different elements from the toolbar.
- Click on an element to update its information in the corresponding popup window

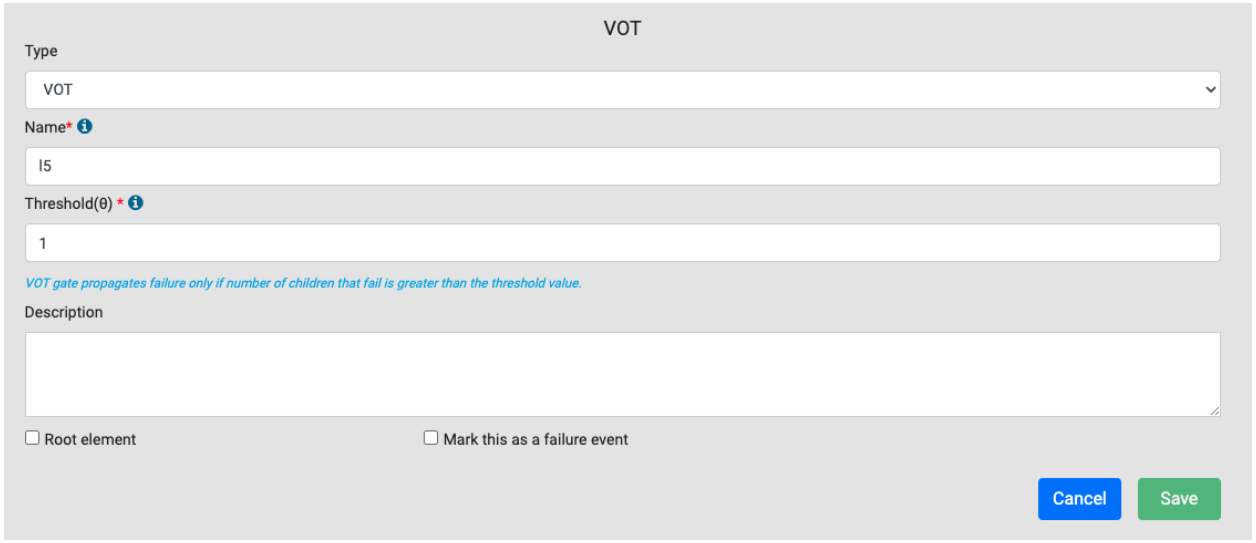


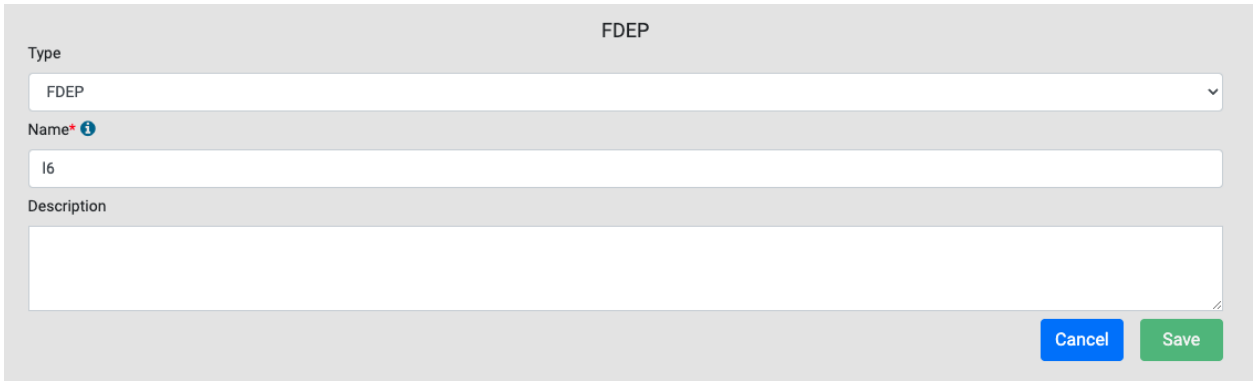
- A block can be created from scratch or uploaded from the drive. To upload, select the file (in JSON or Galileo format) and click the save button.

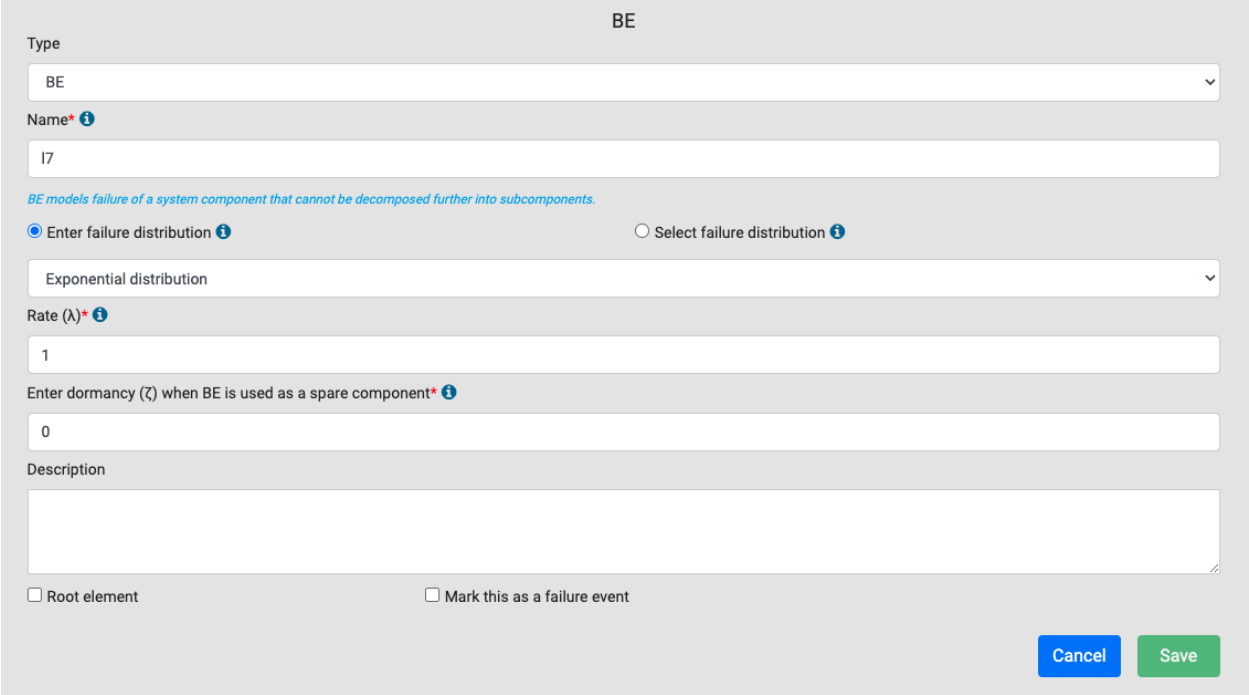




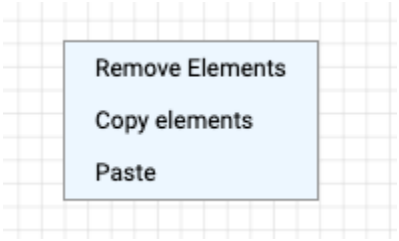

OR
Type
OR
Name*
I3
OR gate propagates failure if any of its children will fail.
Description
Root element
Mark this as a failure event
Cancel
Save


VOT
Type
VOT
Name*
I5
Threshold(θ) *
1
VOT gate propagates failure only if number of children that fail is greater than the threshold value.
Description
Root element
Mark this as a failure event
Cancel
Save


FDEP
Type
FDEP
Name*
I6
Description
Cancel
Save



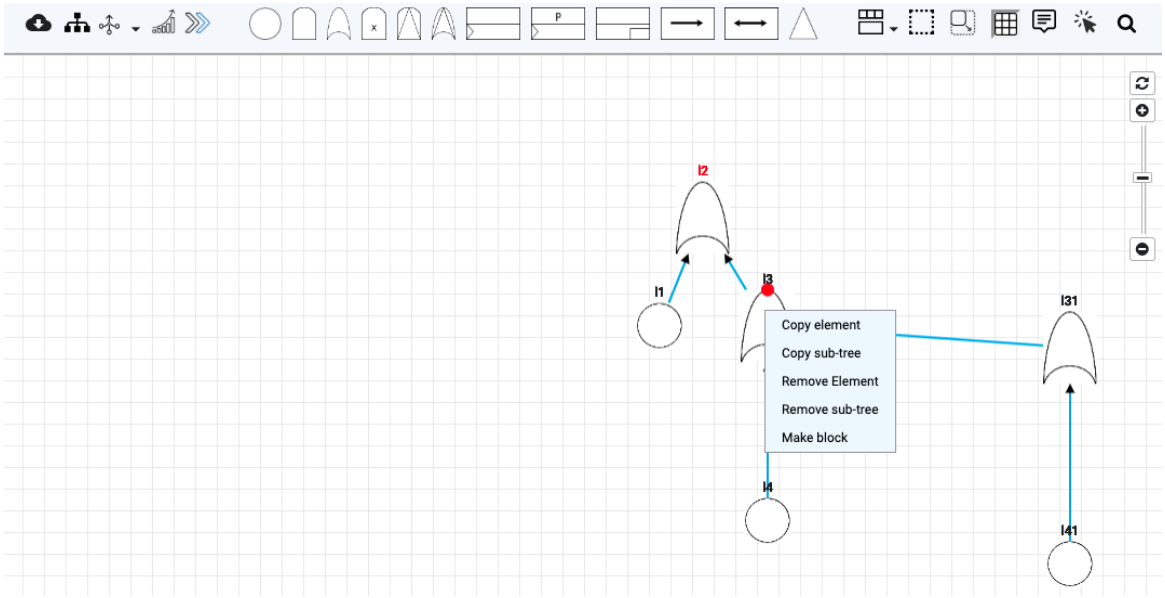
- Only one element can be selected as a top-level element in a tree.
- The values of some of the fields e.g. failure rates, probabilities, etc can be given as parameters (defined later) instead of constant values.
- The “Mark this as a failure event” checkbox is only visible in the advance view.
- The “Mark this as a failure event” checkbox is available only for those elements that propagate failure to their parents. If this checkbox is selected for an element, it is added to “Failure Events” under the failure model (in the left panel) as a basic event. This feature is only available in “Advance View”.
- Elements can be connected with each other by drawing edges between them.
- Click on the icon  to enable the selection of multiple elements in the grid view. This



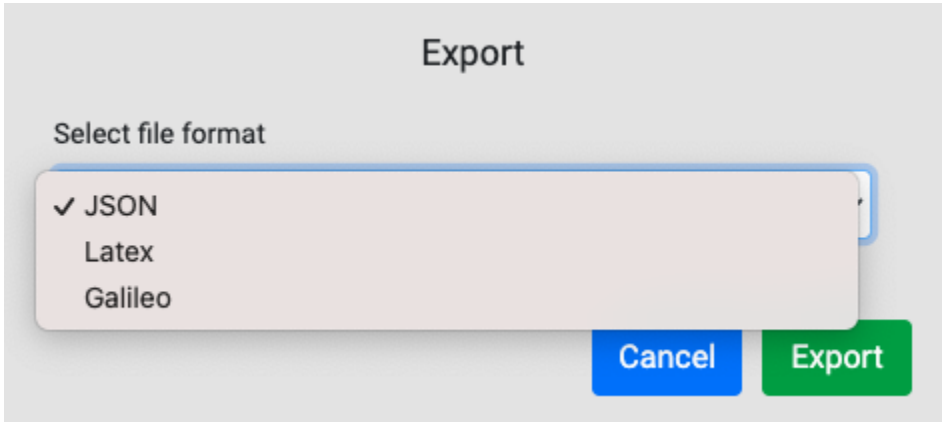
  can be done by clicking on a screen and then drawing the mouse. All elements with a rectangle will be selected, which can then be moved together.
- Right-click on the canvas and the following popup will appear:

  - By clicking “Remove Elements” all selected elements will be removed.
  - By clicking “Copy elements” all selected elements will be copied.
  - By clicking “Paste” previously copied elements will be pasted.

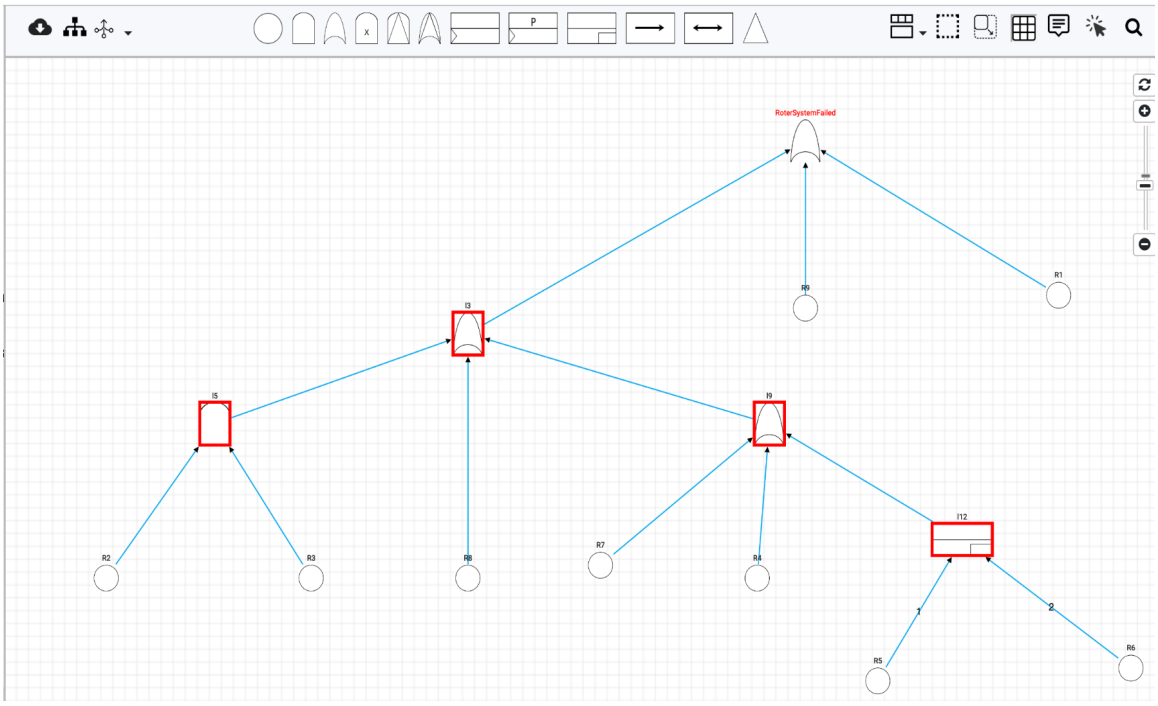
- On right-clicking on an element, a popup comes up that allows you to copy the element, copy the sub-tree under it, delete the element, delete the sub-tree under it, or convert the sub-tree under it into a block (if possible)



- Click on the download icon , a popup comes up to download the tree in JSON, Latex, and Galileo formats.



- Click on the icon to highlight the elements that, along with their children, can be converted into modules (independent sub-trees). In order to convert an element and its children into a module, right-click on it and then click the “Make block”.



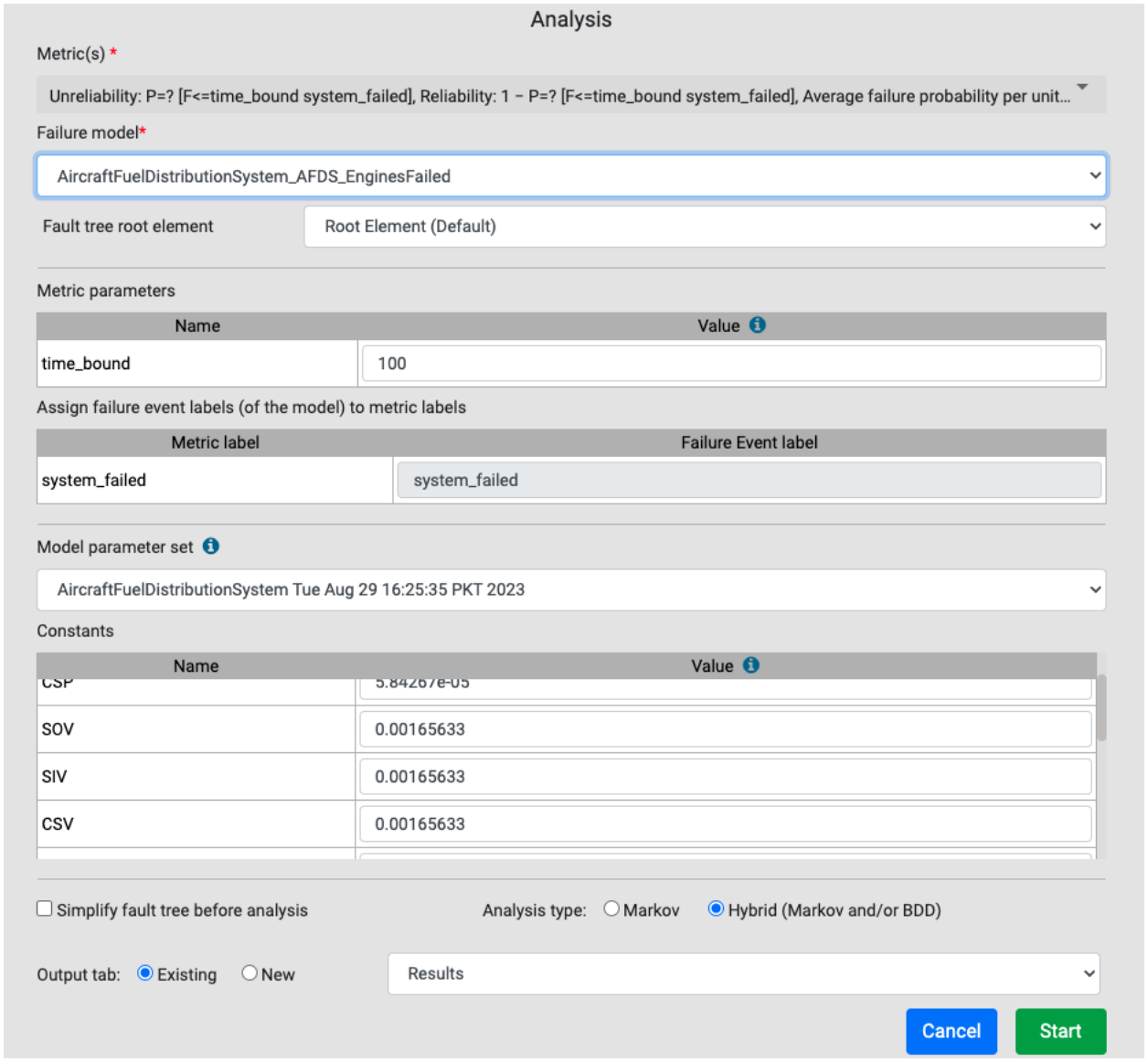
- In order to simplify a fault tree click on the down arrow along the icon  . It will give three options for tree simplification:
    - Simplify by all rules. It will apply all rules recursively for simplification.
    - Default rules. It will apply a selected set of rules (most commonly used) for simplification. They are:
        - SPLIT_FDEPS – Split FDEPs with two or more children into single FDEPs with only one child.
        - MERGE_BES – Try to merge BEs under an OR-gate into one BE.
        - TRIM – Trim parts of the DFT in place that do not contribute to the top-level element.
        - REMOVE_SINGLE_SUCCESSOR – Remove gates with just one successor. These gates will fail together with this child, so they can directly be eliminated.
        - FLATTEN_GATE – Flattening of AND-/OR-/PAND-gates.
    - Custom rules. It allows users to select rules that are to be applied for simplification.

Simplification Rules

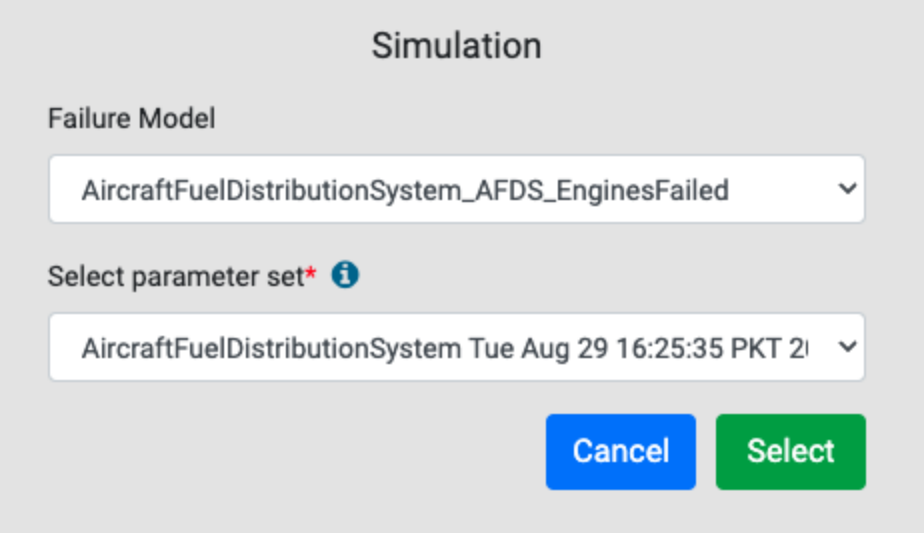
	Name	Description
☐	SPLIT_FDEPS	Split FDEPs with two or more children into single FDEPs with only one child.
☐	MERGE_BES	Try to merge BEs under an OR-gate into one BE.
☐	TRIM	Trim parts of the DFT in place which do not contribute to the top level element.
☐	REMOVE_DEPENDENCIES_TLE	Try to remove superfluous dependencies. These dependencies have a trigger which already leads to failure of the top level element.
☐	MERGE_IDENTICAL_GATES	Try to merge gates with the same type and identical successors. These gates surely fail simultaneously and thus, one gate can be removed.
☐	REMOVE_SINGLE_SUCCESSOR	Remove gates with just one successor. These gates will fail together with this child, so they can directly be eliminated.
☐	FLATTEN_GATE	Flattening of AND-/OR-/PAND-gates.
☐	SUBSUME_GATE	Subsumption of OR-gate by AND-gate or of AND-gate by OR-gate.
☐	REPLACE_FDEP_BY_OR	Eliminate FDEPs by introducing an OR-gate. Let A be the trigger and B be the dependent element. Both must be connected to the top level element. B must have only one predecessor and no SPARE or PAND/POR in its predecessor closure.
☐	REMOVE_SUPERFLUOUS_FDEP	Eliminate superfluous FDEP from AND or OR. This FDEP is triggered after the failure of the dependent element and thus, it does not influence anything else.
☐	REMOVE_SUPERFLUOUS_FDEP_SUCCESSO...	Eliminate FDEP between two successors of an OR or PAND. Only supports FDEPs with one common predecessor.

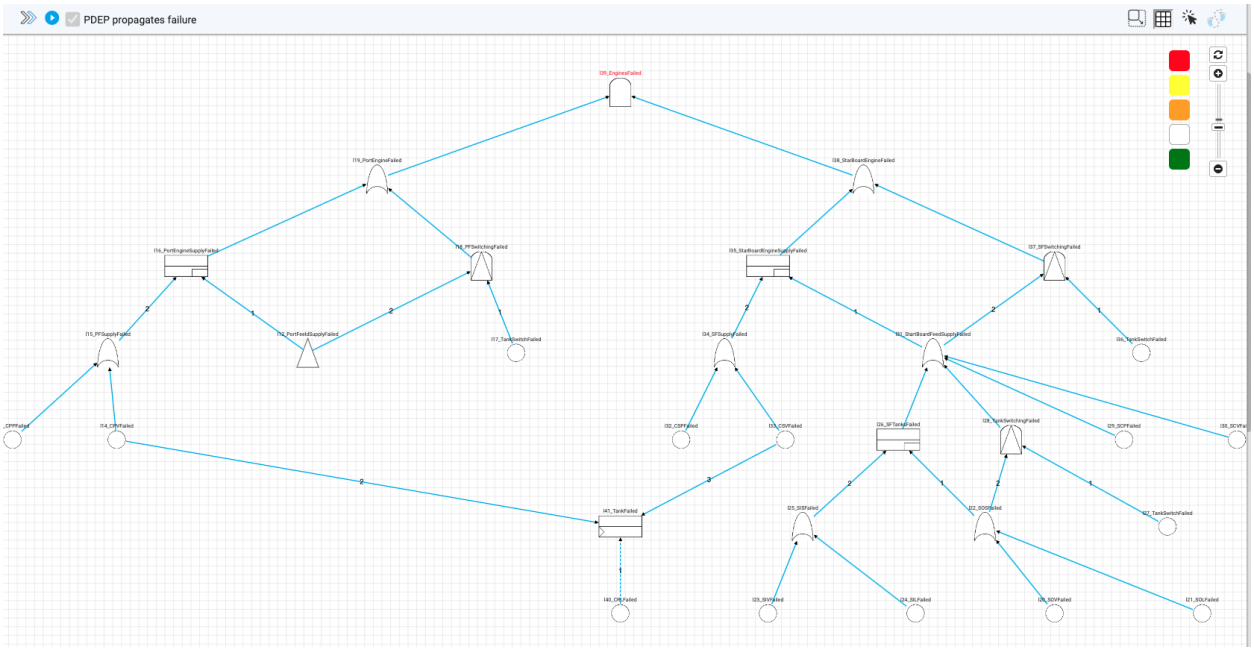
Cancel Simplify

- Click on the icon  (not visible when sub-trees are in focus) to do a basic analysis that includes reliability, mean-time-to-failure, and average-failure-probability per hour. It will take the user to the “Analysis” window under “Computing” in the left panel.



- Click on the icon  to start the interactive simulation. It will take the user to the “Interactive Simulation” window under “Computing” in the left panel. A popup will appear to select/change the model and the parameter set. On clicking the “Select” button the simulation window appears.





- Click on the down arrow new the icon  , it will give three options to display a tree:
  - Canvas view. It shows the tree in the grid.
  - Tabular view. It will show the tree in a tabular form
  - Textual view. It shows the tree in Galileo format.

Tabular View

Fault Tree (AircraftFuelDistributionSystem_AFDS_EnginesFailed)

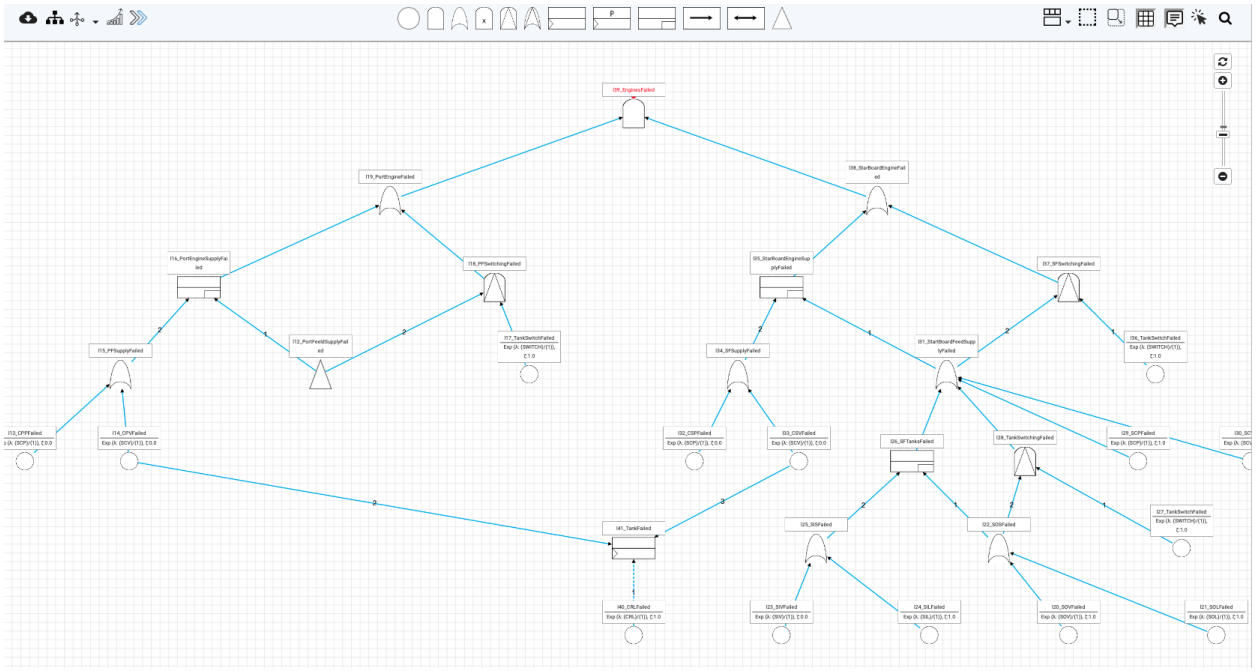
Expand ALL | Collapse All | BE: 14 | Static Gates: 8 | Dynamic Gates: 7 | Blocks: 1

	Name	Type	Information	Description
1	I39_EnginesFailed	AND	Failure Event: true, Root Element: true	AircraftFuelDistributionSystem_AFDS_EnginesFailed
2	I41_TankFailed	FDEP		AircraftFuelDistributionSystem_AFDS_centralReservation_TankFailed
3	I40_CRLFailed	BE	Exp (λ: (CRL)/(1)), ζ: 1.0	AircraftFuelDistributionSystem_AFDS_centralReservation_crt_CRLFailed
4	I38_StarBoardEngineFailed	OR		AircraftFuelDistributionSystem_AFDS_StarBoardEngineFailed
5	I37_SFSwitchingFailed	PAND		AircraftFuelDistributionSystem_AFDS_SFSwitchingFailed
6	I36_TankSwitchFailed	BE	Exp (λ: (SWITCH)/(1)), ζ: 1.0	AircraftFuelDistributionSystem_AFDS_sfSwitch_TankSwitchFailed
7	I35_StarBoardEngineSupplyFailed	SPARE		AircraftFuelDistributionSystem_AFDS_StarBoardEngineSupplyFailed
8	I34_SFSupplyFailed	OR		AircraftFuelDistributionSystem_AFDS_centralReservation_SFSupplyFailed
9	I33_CSVFailed	BE	Exp (λ: (SCV)/(1)), ζ: 0.0	AircraftFuelDistributionSystem_AFDS_centralReservation_csv_CSVFailed
10	I32_CSPFailed	BE	Exp (λ: (SCP)/(1)), ζ: 0.0	AircraftFuelDistributionSystem_AFDS_centralReservation_csp_CSPFailed
11	I31_StartBoardFeedSupplyFailed	OR		AircraftFuelDistributionSystem_AFDS_starboardFeed_StartBoardFeedSupplyFailed
12	I30_SCVFailed	BE	Exp (λ: (SCV)/(1)), ζ: 1.0	AircraftFuelDistributionSystem_AFDS_starboardFeed_scv_SCVFailed
13	I29_SCPFailed	BE	Exp (λ: (SCP)/(1)), ζ: 1.0	AircraftFuelDistributionSystem_AFDS_starboardFeed_scp_SCPFailed
14	I28_TankSwitchingFailed	PAND		AircraftFuelDistributionSystem_AFDS_starboardFeed_TankSwitchingFailed
15	I27_TankSwitchFailed	BE	Exp (λ: (SWITCH)/(1)), ζ: 1.0	AircraftFuelDistributionSystem_AFDS_starboardFeed_tankSwitch_TankSwitchFailed
16	I26_SFTanksFailed	SPARE		AircraftFuelDistributionSystem_AFDS_starboardFeed_SFTanksFailed
17	I25_SISFailed	OR		AircraftFuelDistributionSystem_AFDS_starboardFeed_sis_SISFailed
18	I24_SILFailed	BE	Exp (λ: (SIL)/(1)), ζ: 1.0	AircraftFuelDistributionSystem_AFDS_starboardFeed_sis_sit_SILFailed
19	I23_SIVFailed	BE	Exp (λ: (SIV)/(1)), ζ: 0.0	AircraftFuelDistributionSystem_AFDS_starboardFeed_sis_siv_SIVFailed
20	I22_SOSFailed	OR		AircraftFuelDistributionSystem_AFDS_starboardFeed_sos_SOSFailed
21	I21_SOLFailed	BE	Exp (λ: (SOL)/(1)), ζ: 1.0	AircraftFuelDistributionSystem_AFDS_starboardFeed_sos_sot_SOLFailed
22	I20_SOVFailed	BE	Exp (λ: (SOV)/(1)), ζ: 1.0	AircraftFuelDistributionSystem_AFDS_starboardFeed_sos_sov_SOVFailed
23	I19_PortEngineFailed	OR		AircraftFuelDistributionSystem_AFDS_PortEngineFailed
24	I18_PFSwitchingFailed	PAND		AircraftFuelDistributionSystem_AFDS_PFSwitchingFailed
25	I17_TankSwitchFailed	BE	Exp (λ: (SWITCH)/(1)), ζ: 1.0	AircraftFuelDistributionSystem_AFDS_pfSwitch_TankSwitchFailed
26	I16_PortEngineSupplyFailed	SPARE		AircraftFuelDistributionSystem_AFDS_PortEngineSupplyFailed
27	I15_PFSupplyFailed	OR		AircraftFuelDistributionSystem_AFDS_centralReservation_PFSupplyFailed
28	I14_CPVFailed	BE	Exp (λ: (SCV)/(1)), ζ: 0.0	AircraftFuelDistributionSystem_AFDS_centralReservation_cpv_CPVFailed
29	I13_CPPFailed	BE	Exp (λ: (SCP)/(1)), ζ: 0.0	AircraftFuelDistributionSystem_AFDS_centralReservation_cpp_CPPFailed
30	I12_PortFeeldSupplyFailed Expand Block	BLOCK		AircraftFuelDistributionSystem_AFDS_portFeed_PortFeeldSupplyFailed

Galileo

```
param G7;
param M5;
param R6;
param G1;
toplevel "OWTFailed";
"OWTFailed" or "I6_I6" "FW7" "FW1" "FW2" "FW3" "WindTurbine_WindTurbineFailed" "MooringSystem_MooringSystemFailed" "Tower_TowerFailed" "SemiFloatingFoundation_SFFFailed";
"I6_I6" or "I6_FW4" "I6_FW5" "I6_FW6";
"FW7" lambda=FW7 dorm=1.0;
"FW1" lambda=FW1 dorm=1.0;
"FW2" lambda=FW2 dorm=1.0;
"FW3" lambda=FW3 dorm=1.0;
"WindTurbine_WindTurbineFailed" or "WindTurbine_W2" "WindTurbine_W3" "WindTurbine_W4" "WindTurbine_W7" "WindTurbine_W8" "WindTurbine_W9" "WindTurbine_W10" "WindTurbine_W11"
"WindTurbine_W12" "WindTurbine_RoterSystem_RoterSystemFailed" "WindTurbine_Sensors_SensorsFailed" "WindTurbine_Generator_GeneratorFailed";
"MooringSystem_MooringSystemFailed" or "MooringSystem_M7" "MooringSystem_I3" "MooringSystem_I4";
"Tower_TowerFailed" or "Tower_I2" "Tower_T1" "Tower_T2" "Tower_T3" "Tower_T10" "Tower_T4" "Tower_T9";
"SemiFloatingFoundation_SFFFailed" or "SemiFloatingFoundation_F1" "SemiFloatingFoundation_I3" "SemiFloatingFoundation_I4" "SemiFloatingFoundation_F9";
"I6_FW4" lambda=FW4 dorm=1.0;
"I6_FW5" lambda=FW5 dorm=1.0;
"I6_FW6" lambda=FW6 dorm=1.0;
"WindTurbine_W2" lambda=8e-06 dorm=1.0;
"WindTurbine_W3" lambda=1e-05 dorm=1.0;
"WindTurbine_W4" lambda=1e-05 dorm=1.0;
"WindTurbine_W7" lambda=1.6e-05 dorm=1.0;
"WindTurbine_W8" lambda=2e-06 dorm=1.0;
"WindTurbine_W9" lambda=3.9e-05 dorm=1.0;
"WindTurbine_W10" lambda=1.3e-05 dorm=1.0;
"WindTurbine_W11" lambda=4e-06 dorm=1.0;
"WindTurbine_W12" lambda=3.3e-05 dorm=1.0;
"WindTurbine_RoterSystem_RoterSystemFailed" or "WindTurbine_RoterSystem_R9" "WindTurbine_RoterSystem_I3" "WindTurbine_RoterSystem_R1";
"WindTurbine_Sensors_SensorsFailed" vot2 "WindTurbine_Sensors_W61" "WindTurbine_Sensors_W62" "WindTurbine_Sensors_W63";
```

- Click on the icon  to enable the navigator at the bottom of the screen.
- Click on the icon  to show the grid on the screen.
- Click on the icon  to show the summary information about each element on the screen.

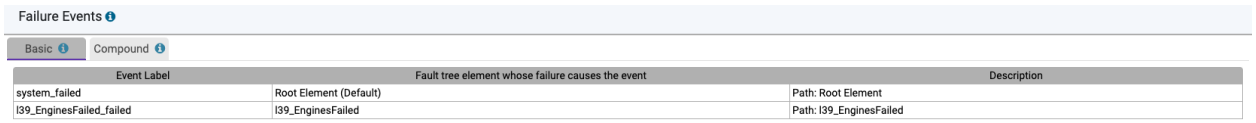


- Click on the icon  to display summary information about an element on hovering.
- Click on the icon  to search for any element on the screen.

Failure Events

Failure events are basic and compound events (in the form of boolean equations on elements of a fault tree) that are important/relevant and their probabilities can be computed.

In the advanced view, “Failure Events” are visible in the left panel under each failure model. The “Failure Events” page has two tabs.

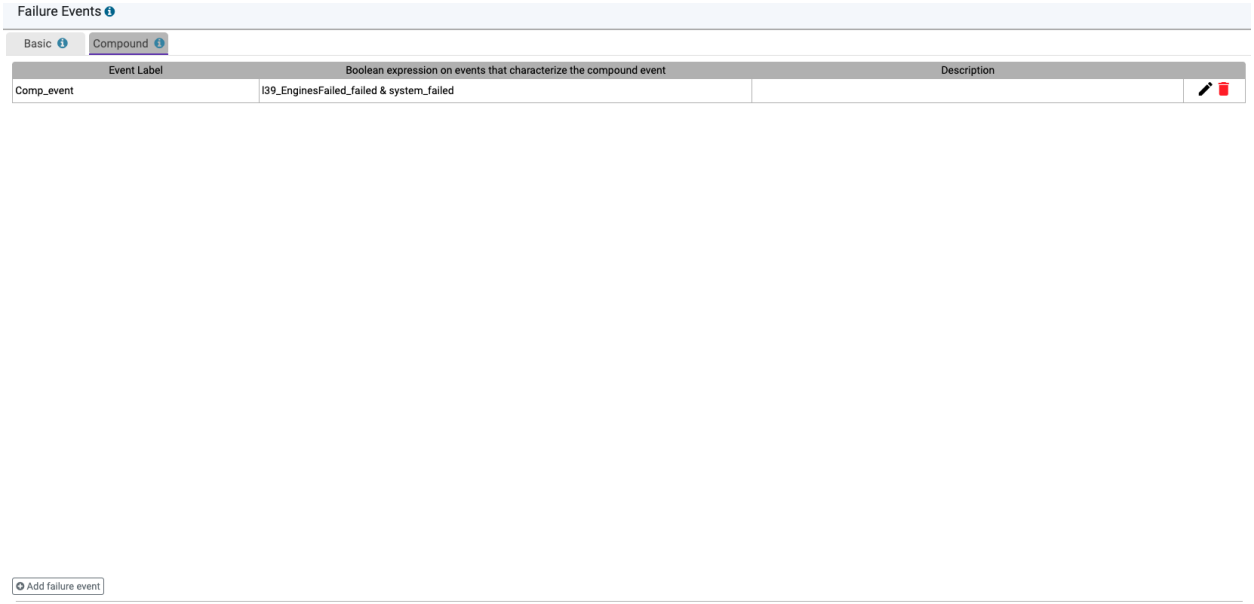


Basic

It will display all elements of the fault tree whose “Mark this as a failure event” checkbox is selected. Failures of such elements are important events whose probabilities can be computed.

Composite

On this tab, we can create compound events by writing boolean expressions on the other events (basic and compound).



Click on the “Add failure event” to show a popup

Enter event label and boolean expression on existing events (shown below in the field “The above expression can use the following Event labels”). On filling in mandatory fields, click the “Add” button.

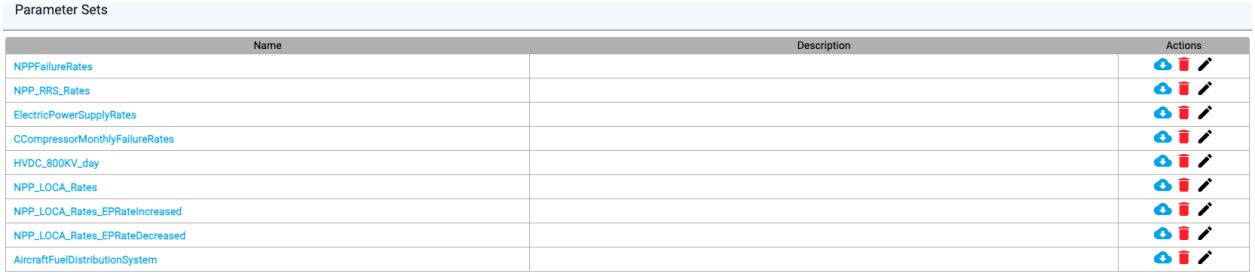
Parameter Sets

Each parameter set contains a list of parameters that are key-value pairs. They are used to specify values of e.g. probabilities, failure rates, etc. in fault tree models. By changing their values several variants of fault trees can be generated, which can then be compared with each other based on the results of metrics of interest. Each parameter set comprises:

- Constants
- Real-valued expressions
- Probability distributions (Exponential, Erlang, Weibull, Log-normal)
- Empirical probability distributions – failure distributions generated from data sets.

View

Click the “Parameter Sets” under “Fault Tree Analysis” in the left panel to view all existing parameter sets.



Creation

Click the “Add parameter set” button to create a new parameter.

Import

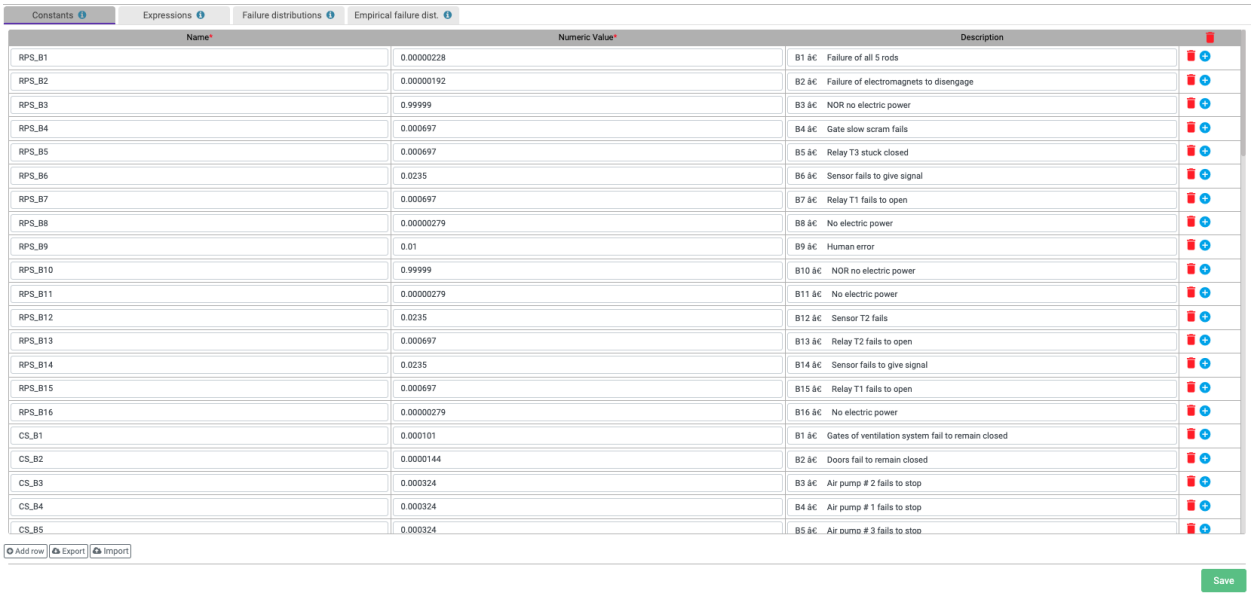
Click the “Import parameter set” to import the parameter set in .ps format (a format in which parameter sets are imported/exported in our tool).

Export

Click the icon to export a parameter set in .ps format.

Update

Click a parameter set to update its details



Constants

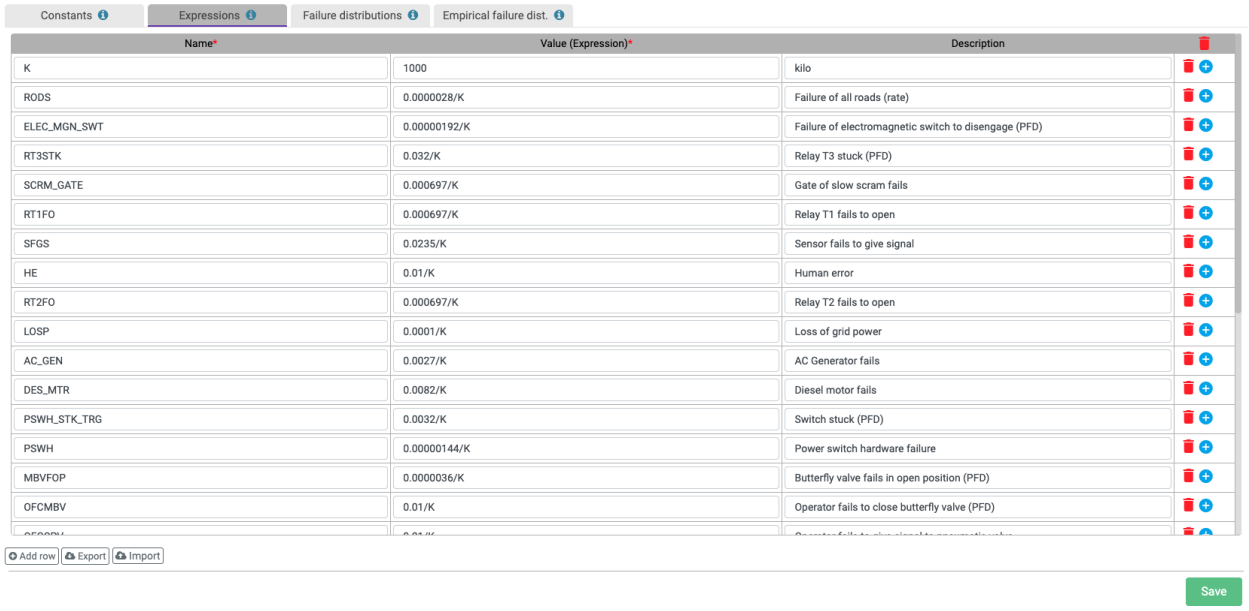
Constants can only be numeric e.g. 4, 2.3, 4e-6 etc. Their value can be changed at the time of analysis. For example, graphs can be plotted for matric results against ranges of values of constants.

- Click “Add row” to enter a new row in the table.
- Click “Export” to export constants in a csv format.
- Click “Import” to import constants from a csv file.
- Click the icon in the last column of any row to add a new row above it.
- Click the “Save” button to save the data. This action will save data in all the tabs.

Real-value Expressions

These are non-negative, real-value expressions, which can use constants (defined above) e.g. x + 2 where x is a constant. The grammar of the expression is given here.

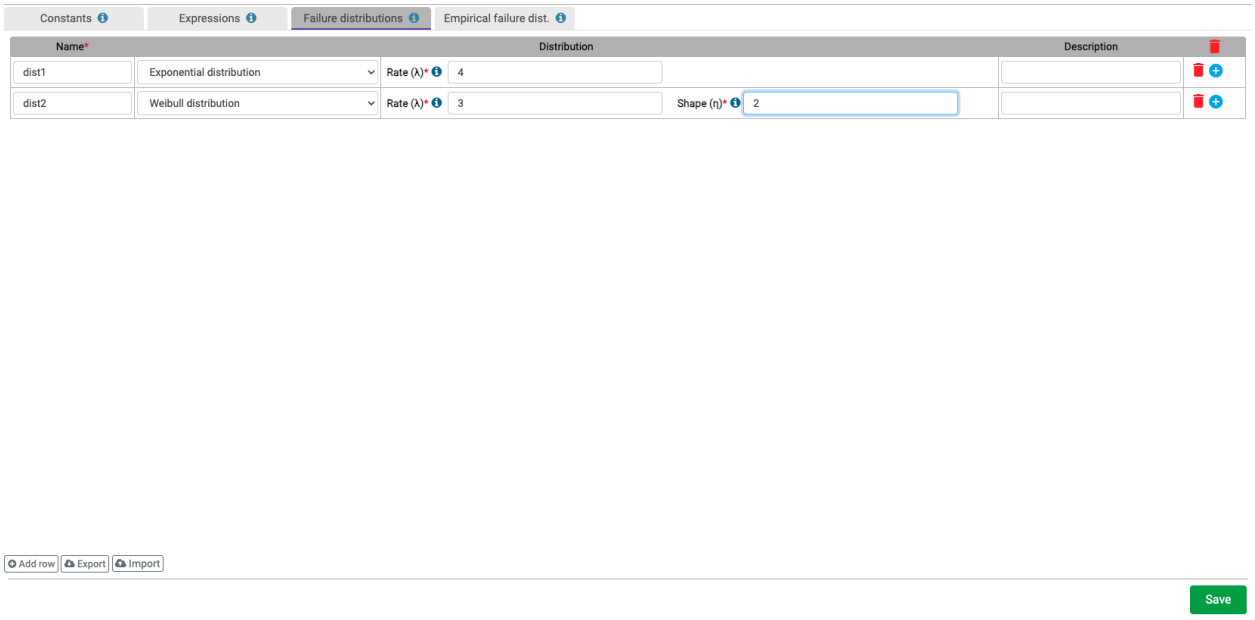
- Click “Add row” to enter a new row in the table.
- Click “Export” to export expressions in a csv format.
- Click “Import” to import expressions from a csv file.
- Click the icon in the last column of any row to add a new row above it.
- Click the “Save” button to save the data. This action will save data in all the tabs.



Failure distributions

Failure distributions can be exponential, erlang, Weibull, log-normal, and constant probability. Multiple distributions can be added by pressing the add row.

- Click “Add row” to enter a new row in the table.
- Click “Export” to export failure distributions in a csv format.
- Click “Import” to import failure distributions from a csv file.
- Click the  icon in the last column of any row to add a new row above it.
- Click the “Save” button to save the data. This action will save data in all the tabs.



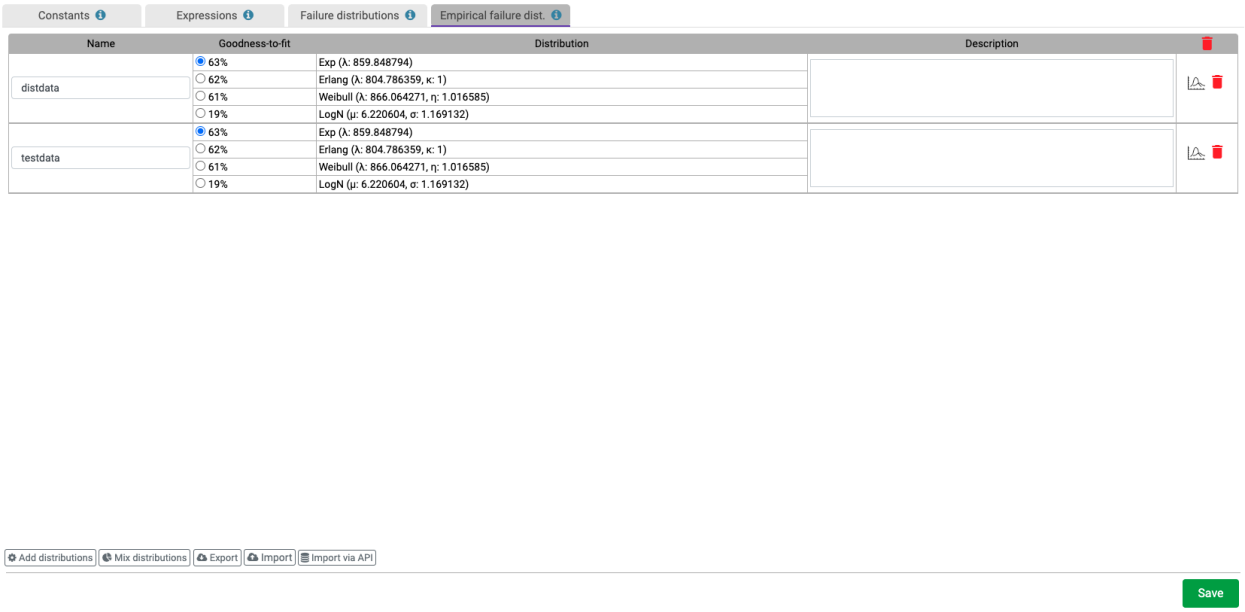
Empirical failure distributions

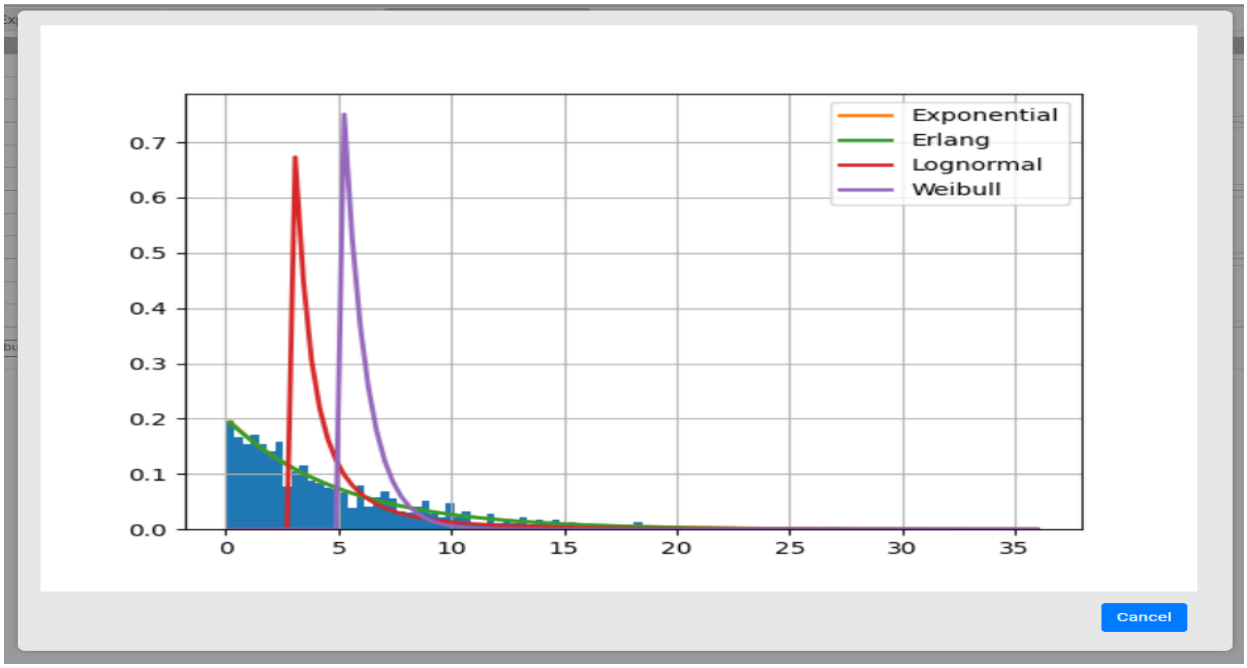
Empirical distributions are calculated from a data set using statistical methods. For that, historical failure data of a component is used to estimate the tentative failure probability distributions, which might have generated it, sorted according to their goodness-to-fit (GTF) values -- GTF value indicates the chance the data was generated by the corresponding distribution.

You can see how these distributions fit the data by clicking on the graph icon in the last column of their corresponding rows. Using Empirical distribution:

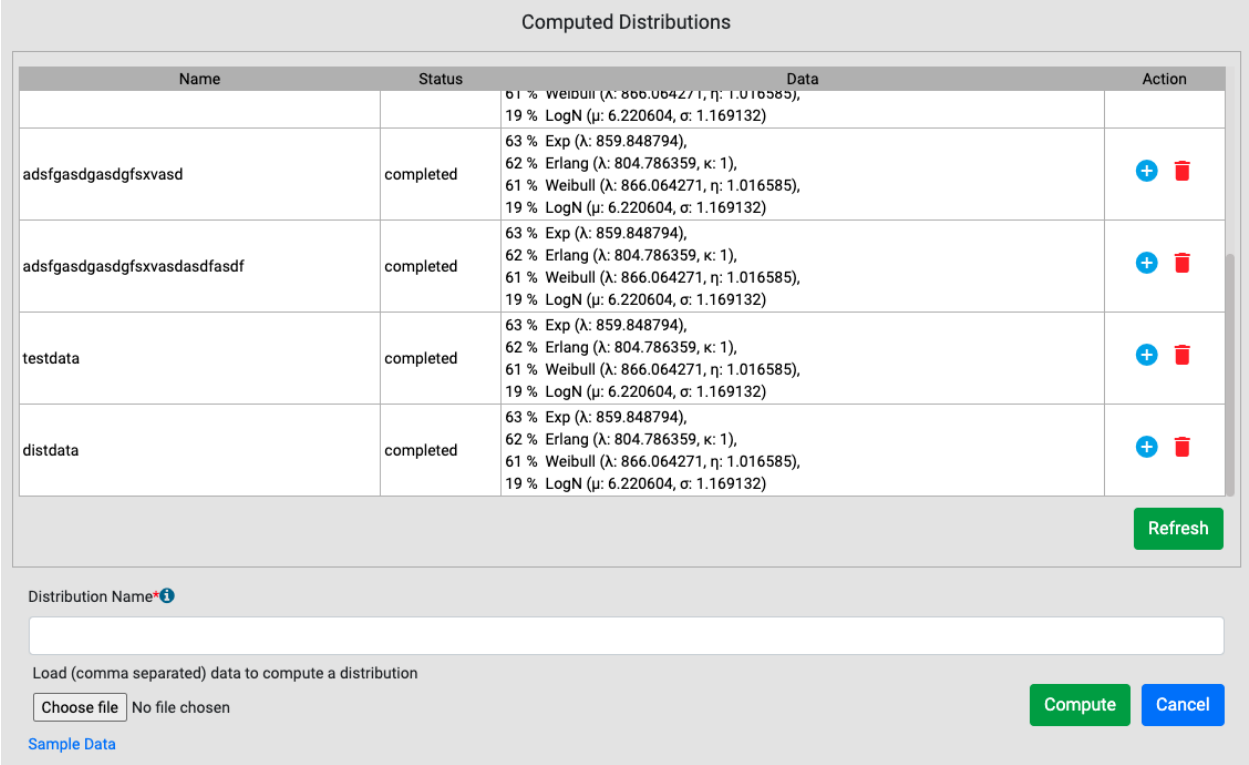
Each data may fit on multiple distributions, which are sorted according to their goodness-to-fit values, therefore we provide a radio button to select any distribution that we want to use.

Moreover, you can import and export empirical distributions (csv format) and use them in other projects.

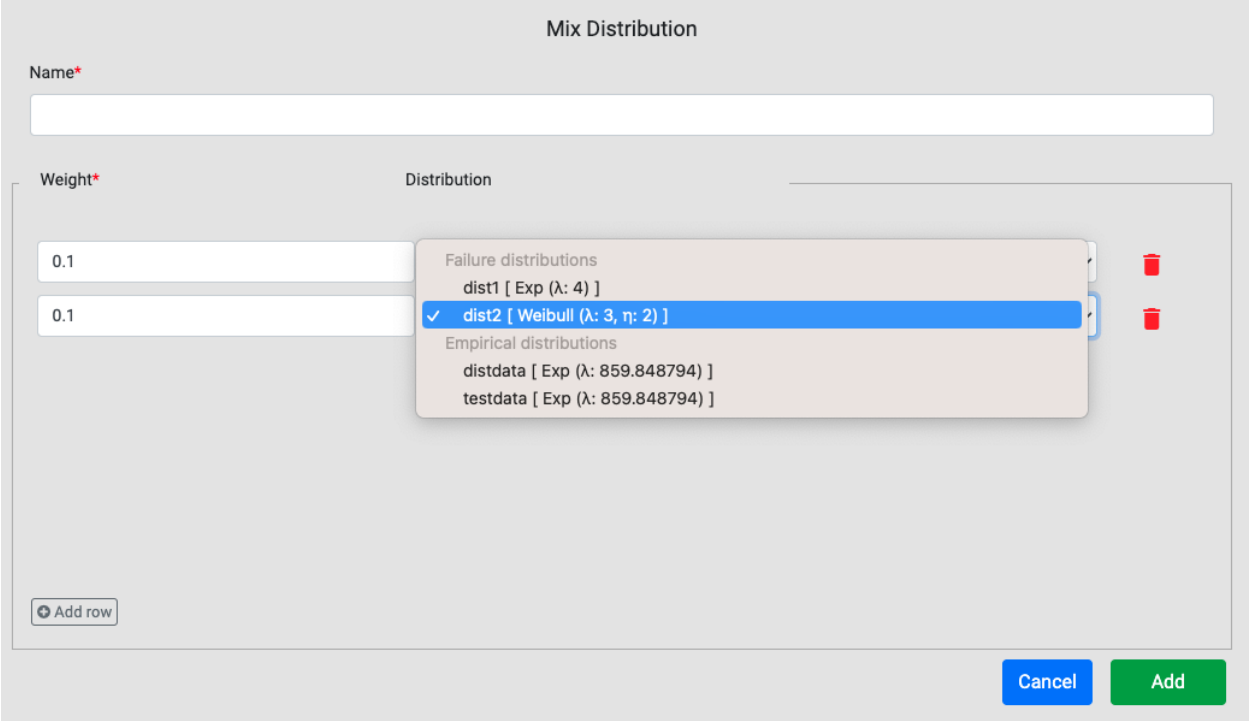




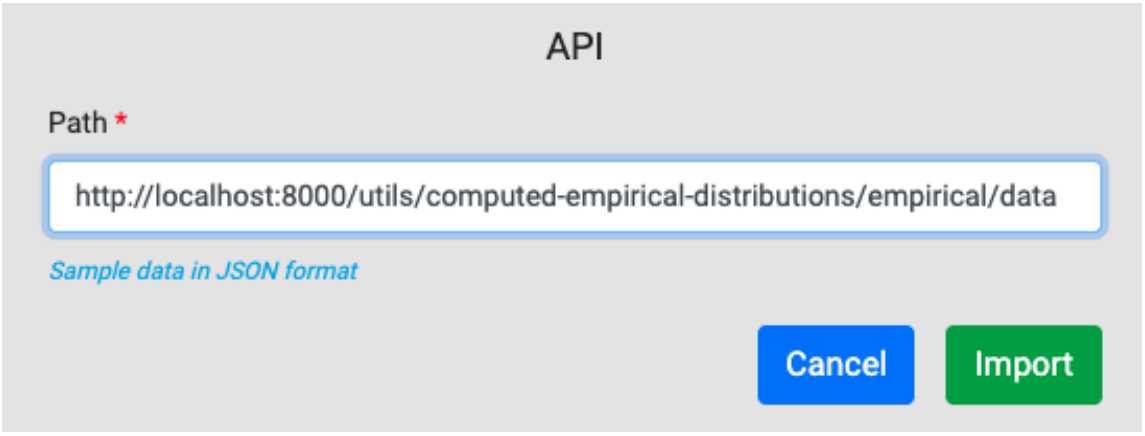
- **Generating empirical distribution.** An empirical distribution can be added by clicking the “Add distribution” button. It will display all empirical distributions that have been generated previously and stored on the server side. One can compute a new distribution by specifying the file that contains data on which the distribution is to be approximated.
    - In the “Computed Distributions” table, Click the icon in a row to add the distribution in the “Empirical Distributions” table in the main tab.
    - To generate a new distribution from data, enter the name of the distribution in the “Distribution Name” field, choose the data file from your drive, and click the “Compute” button. A new row will appear in the “Computed Distributions” table with a “running” status.
    - Click the “Refresh” button to check whether the distributions with “running” status have been computed.



- **Generating mixture distribution:** a mixture distribution can also be generated by clicking the “Mix distributions” button. It will show a popup where distributions along with their weights can be added. For example, d3 = 0.3*d1 + 0.7*d2 is a mixture distribution, where d1 and d2 are existing (empirical) failure distributions in the “Failure distributions” and “Empirical failure distributions” tabs. Click the “Add row” button if you want to add more distributions to the mixture.



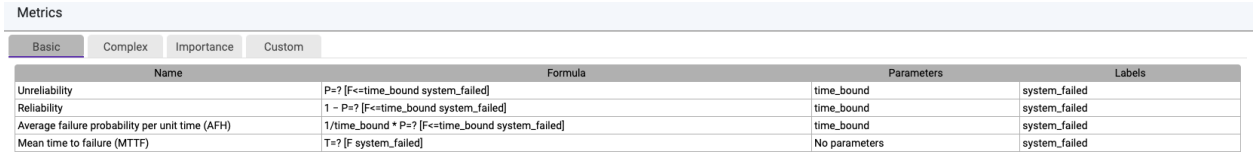
- Click the “Export” button to export empirical distributions in a JSON format (that includes the image data as well).
- Click the “Import” button to import empirical distributions in a JSON format (that includes the image data as well).
- Click the “Import via API” button to get empirical distributions that have been generated externally at a given API. The same data format can be accessed by clicking “Sample data in JSON format” on the popup.



Metrics

Metrics help us formally specify properties of interest we want to verify about failure models. There is a list of predefined metrics (classified into basic, complex, and Importance metrics). For advanced users, it is possible to define custom metrics using continuous stochastic logic (CSL).

In advance view “Metrics” link is visible in the left panel. Click on it, and a screen with four tabs: Basic, Complex, Criticality, and Custom will be visible.



Name	Formula	Parameters	Labels
Unreliability	P=? [F<=time_bound system_failed]	time_bound	system_failed
Reliability	1 − P=? [F<=time_bound system_failed]	time_bound	system_failed
Average failure probability per unit time (AFH)	1/time_bound * P=? [F<=time_bound system_failed]	time_bound	system_failed
Mean time to failure (MTTF)	T=? [F system_failed]	No parameters	system_failed

Basic

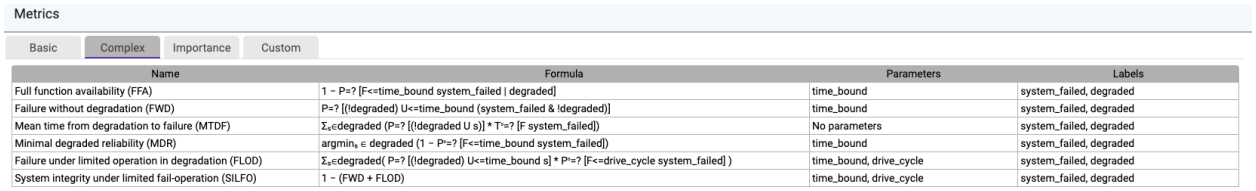
It contains four important metrics that are verified in most of the reliability analysis cases:

- Mean-time-to-failure. Expected time to system failure or scenario occurrence.
- Reliability: Probability of failure in a given time bound.
- Unreliability: The complement of reliability (1- Reliability).
- Average failure probability per hour.

Complex

It contains six metrics. These metrics cannot be verified directly by the Storm model-checker. They need some additional computations at the back end for their verification.

- Full Function Availability (FFA) describes the time-bounded probability that the system provides full functionality, i.e., it has neither failed nor degraded. It is described as the complement of the time-bounded reachability of a failed or degraded state.
- Failure Without Degradation (FWD) describes the time-bounded probability that the system fails without being degraded first. It is the time-bounded reach-avoid probability of reaching a failed state without reaching a degraded state.
- Mean Time from Degradation to Failure (MTDF) describes the expected time from the moment of degradation to system failure. It is obtained by taking the expected time of failure for each degraded state and scaling it with the probability of reaching this state while not being degraded before.
- Minimal Degraded Reliability (MDR) describes the criticality of degraded states by giving the worst-case failure probability when using the system in a degraded state. For all degraded states, the time-bounded reachability of a TLE failure is computed. The MDR is the minimum over the complement of this result for all degraded states.
- Failure under Limited Operation in Degradation (FLOD) describes the probability of failure when imposing a time limit for using a degraded system. For all degraded states, the time-bounded reachability probability of a failed state is computed within the restricted time-bound given by a drive cycle. This value is scaled by the time-bounded reach-avoid probability of reaching a degraded state without degradation before.
- System Integrity under Limited Fail-Operation (SILFO) considers the system-wide impact of limiting the degraded operation time. SILFO is split into two parts considering failures without degradation (FWD) and failures with degradation (FLOD).

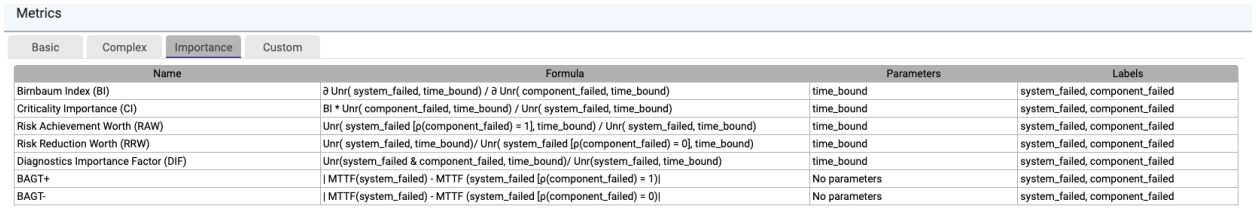

Metrics

Basic | Complex | Importance | Custom

Name	Formula	Parameters	Labels
Full function availability (FFA)	1 − P=? [F<=time_bound system_failed | degraded]	time_bound	system_failed, degraded
Failure without degradation (FWD)	P=? [(!degraded) U<=time_bound (system_failed & !degraded)]	time_bound	system_failed, degraded
Mean time from degradation to failure (MTDF)	Σ_s∈degraded (P=? [(!degraded U s)] * T=? [F system_failed])	No parameters	system_failed, degraded
Minimal degraded reliability (MDR)	argmin_s ∈ degraded (1 − P=? [F<=time_bound system_failed])	time_bound	system_failed, degraded
Failure under limited operation in degradation (FLOD)	Σ_s∈degraded(P=? [(!degraded) U<=time_bound s] * P=? [F<=drive_cycle system_failed])	time_bound, drive_cycle	system_failed, degraded
System integrity under limited fail-operation (SILFO)	1 − (FWD + FLOD)	time_bound, drive_cycle	system_failed, degraded

Importance

It contains seven metrics. These metrics cannot be verified directly by the Storm model-checker. They need some additional computations at the back end for their verification.


Metrics

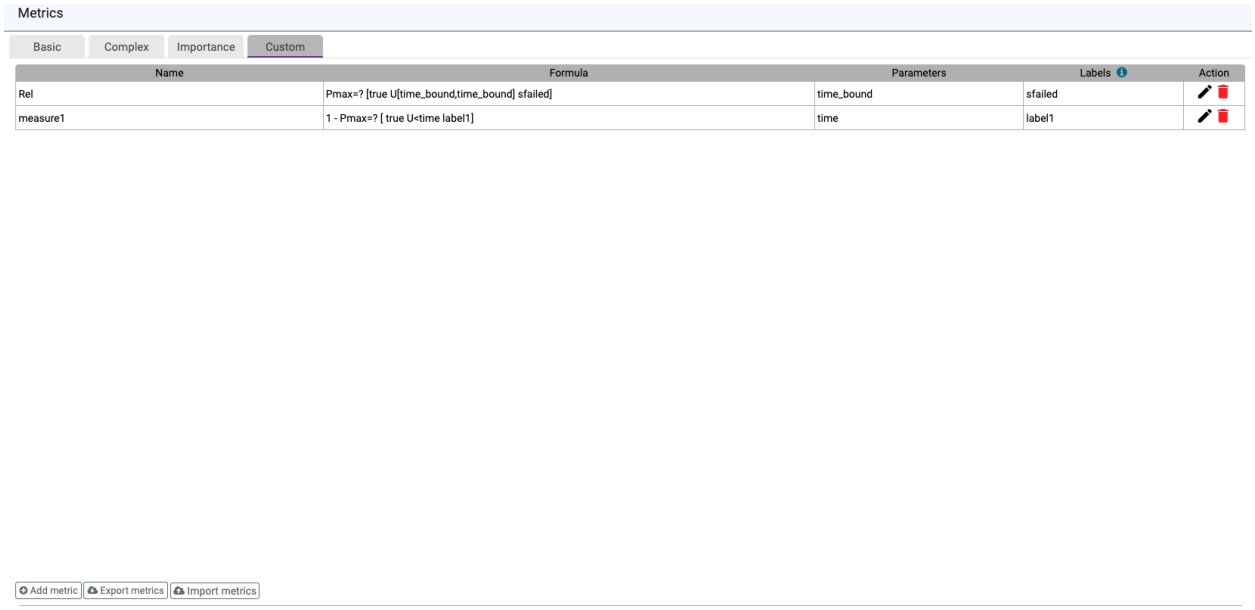
Basic | Complex | Importance | Custom

Name	Formula	Parameters	Labels
Birnbaum Index (BI)	∂ Unr(system_failed, time_bound) / ∂ Unr(component_failed, time_bound)	time_bound	system_failed, component_failed
Criticality Importance (CI)	BI * Unr(component_failed, time_bound) / Unr(system_failed, time_bound)	time_bound	system_failed, component_failed
Risk Achievement Worth (RAW)	Unr(system_failed [p(component_failed) = 1], time_bound) / Unr(system_failed, time_bound)	time_bound	system_failed, component_failed
Risk Reduction Worth (RRW)	Unr(system_failed, time_bound)/ Unr(system_failed [p(component_failed) = 0], time_bound)	time_bound	system_failed, component_failed
Diagnostics Importance Factor (DIF)	Unr(system_failed & component_failed, time_bound)/ Unr(system_failed, time_bound)	time_bound	system_failed, component_failed
BAGT+	| MTTF(system_failed) - MTTF (system_failed [p(component_failed) = 1)|	No parameters	system_failed, component_failed
BAGT-	| MTTF(system_failed) - MTTF (system_failed [p(component_failed) = 0)|	No parameters	system_failed, component_failed

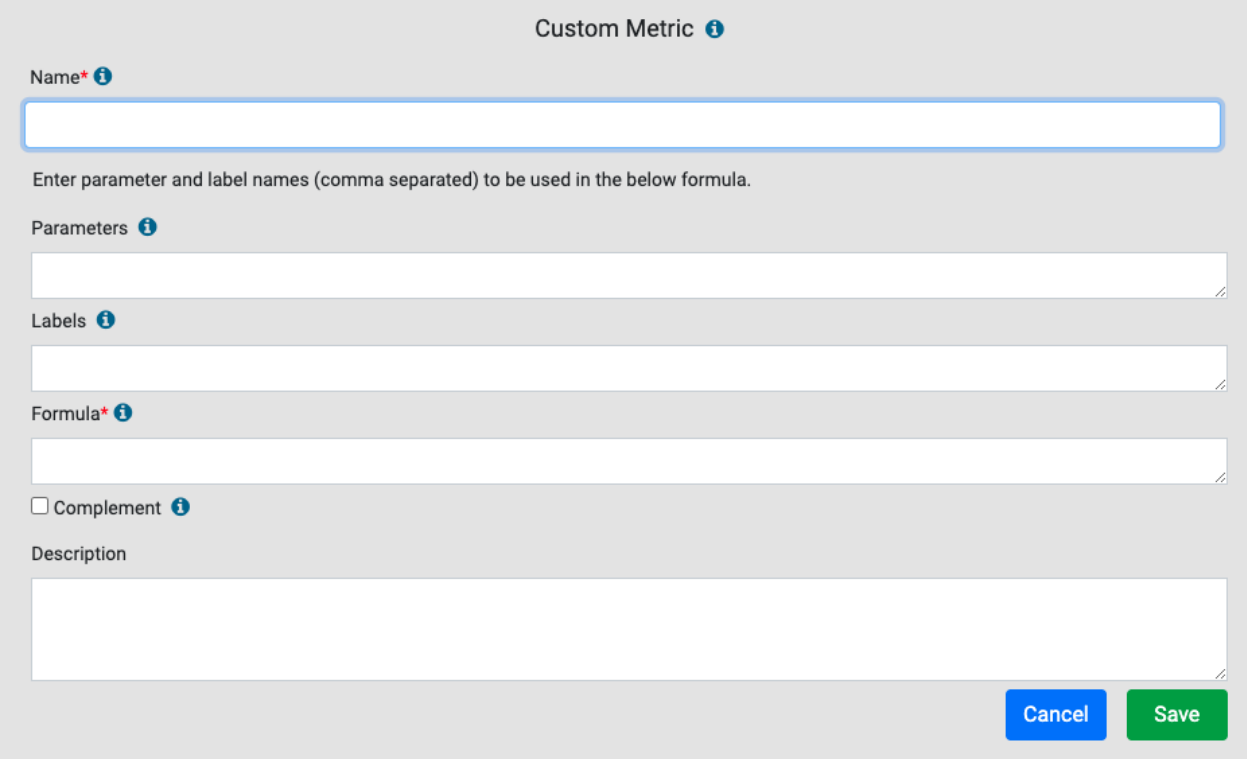
- Birnbaum Index (BI): How much the unreliability of the system depends on the unreliability of a specific component.
- Criticality Importance (CI): How much the unreliability of the system depends on the unreliability of a specific component scaled by the ratio of component and system unreliability.
- Risk Achievement Worth (RAW): The impact of the total degradation of the component on the system unreliability.
- Risk Reduction Worth (RRW): The impact of making the component fully reliable on the system's unreliability.
- Diagnostics Importance Factor (DIF): How often a component fails in states where the system has failed.
- BAGT+: Change in MTTF if the component fully degrades.
- BAGT-: Change in MTTF if the component is fully reliable.

Custom

One can create custom metrics on the “Custom” tab. It allows specifying metrics using continuous stochastic logic (CSL).



- Click the “Add metric” button to add a new metric.



  - Parameters and labels used inside metrics formulae must have unique names among themselves, starting with a letter or underscore (_) followed by underscores, letters, and/or numbers. They must not be from the list of keywords

- - true, false, Pin, Pmax, Smin, Smax, Tmin, Tmax, LAmin, LRAmax, P, R, T, S, LRA, min, max, G, U, F, W, C, I, failed.
    - The formula can be defined using probabilistic computation tree logic (PCTL)/continuous stochastic logic (CSL). For example, P = ? [true U <=10000 (failed & ! mode1)], where failed and mode1 represent quantifiable states. The grammar of expressions is given here.
    - The “Complement” checkbox can be selected in order to calculate the complement of a property mentioned in the formula field.
    - Note that metric parameters which are entered on the above screen are exclusively dedicated to metrics. Their values are not taken from the parameter set that is attached to a failure model. However, their values can be changed at the time of analysis, and ranges for their values can be provided to draw plots.
    - Click the “Export metrics”/”Import metrics” in order to export/import metrics in the .metrics format (a format supported by our SAFEST tool).

Computing

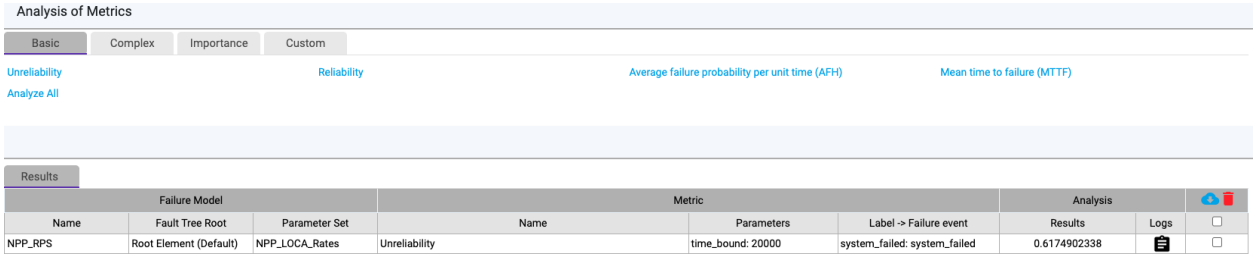
Failure models can be analyzed in different ways:

- Analysis – the exact results of different metrics can be computed.
- Bounded Analysis – the exact values of metrics can be bounded from above and below in a graphical way.
- Graphs – the exact results of different metrics can be graphed against e.g. time.
- Interactive Simulation – failure propagation in fault trees can be shown in an interactive way.
- Minimal Cut Sets – for static fault trees MCS can be computed and displayed graphically.

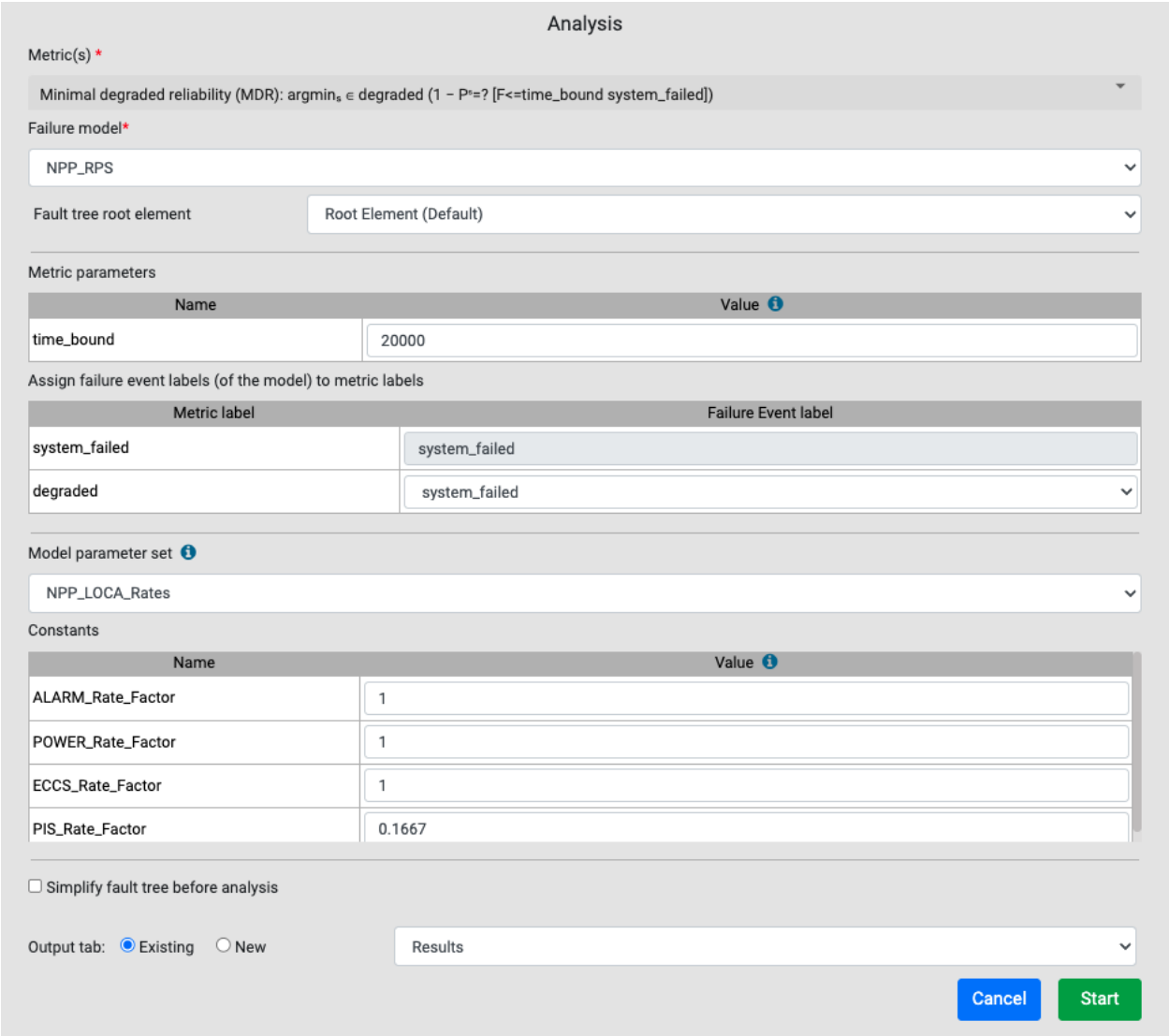
(Exact) Analysis

Complex systems usually have dynamic behavior because of e.g. spare components, failure sequence among components, functional dependencies, etc. The analysis of such systems is quite complex which is usually based on simulation or generalization techniques. Unlike others, we implement formal verification techniques e.g. probabilistic model-checking, and thus provide exact results on measures of interest.

Click on the “Analysis” link under “Computing” in the left panel. The following window will appear with four tabs for different classes of metrics.



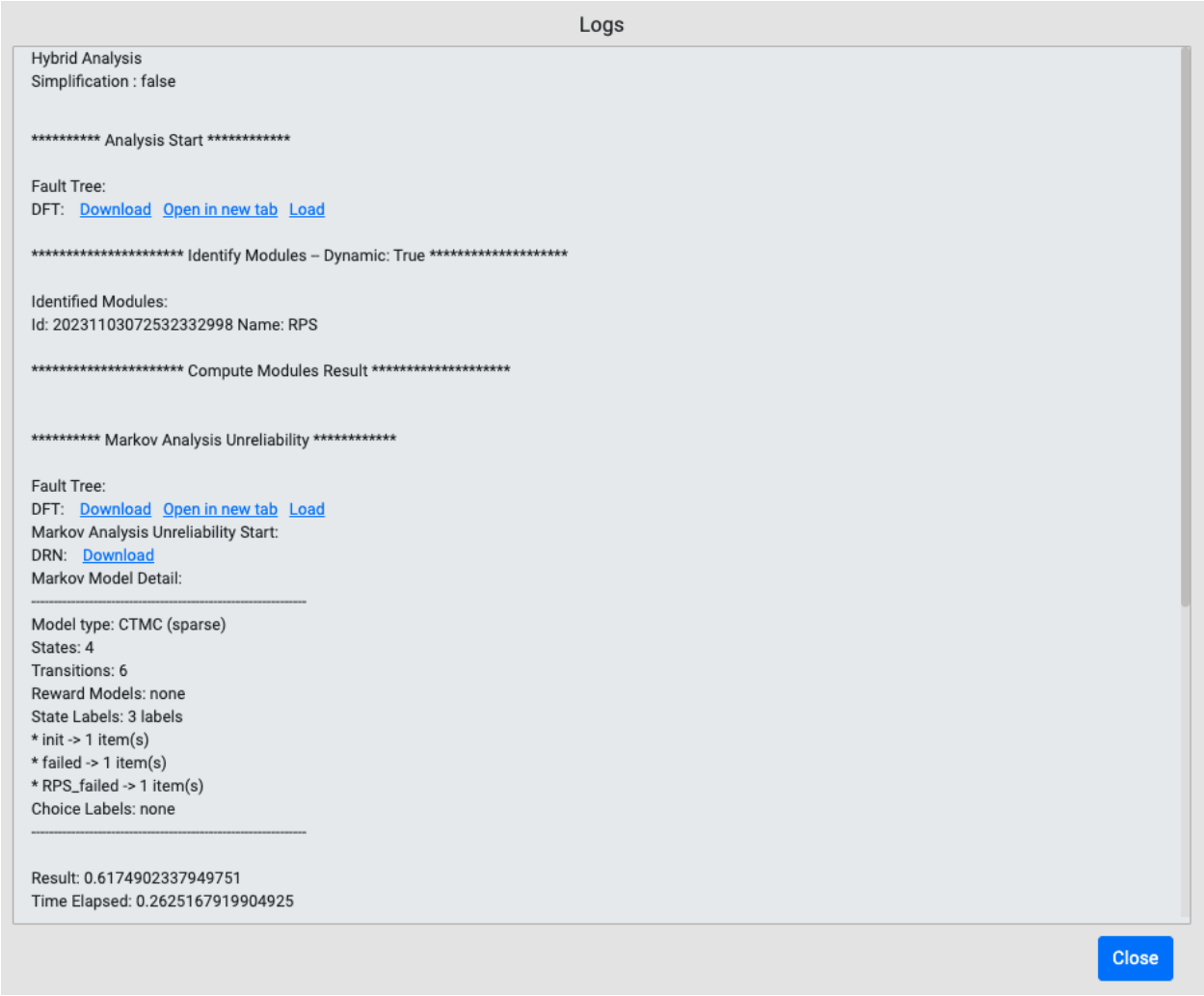
- One can verify a metric on each tab, the mechanism is more or less the same. For example, click on the “Minimal degraded reliability (MDR)” link on the complex tab. The following window will appear:



- “Failure model” dropdown: a model that is selected as a default model in the “Failure Models” window is automatically selected.
- “Fault tree root element”: the root element of the above-selected failure model’s fault tree is selected by default. One can change the root element by selecting any other element in the dropdown. Note that the dropdown contains only those elements of the fault tree whose “Mark this as a failure event” checkbox is selected in the fault tree.
- “Metric parameters”: Note that metric parameters cannot take values defined in the parameter set attached to the above-selected failure model. Each metric parameter has to be assigned a value. For the “time-bound” metric parameter, a default value is assigned to it that is associated with the above-selected failure model.
- “Assign failure event labels (of the model) to metric labels”: Assign failure events of the models to metric labels.

- A parameter set that is attached to the selected failure model (above) is automatically selected. It can be changed at this point to generate another variant of the failure model.
- Optionally, change the values of “Constants” defined in the selected parameter set. Note that values of other elements of the parameter set (real-value expressions, (empirical) failure distributions) cannot be changed at the time of analysis.
- The analysis method is selected automatically based on the selected metric. However, for some metrics both Markov and Hybrid (BDD and/or Markov) analysis are possible. One can decide the analysis type by selecting the corresponding radio button.
- “Simplify fault tree before analysis”: select this checkbox if the fault tree has to be simplified by applying simplification rules before analysis.
- Finally, select a tab on which the result of the analysis has to be displayed. It can be an existing tab or a new tab.

- Click the  icon to view the log of the analysis.



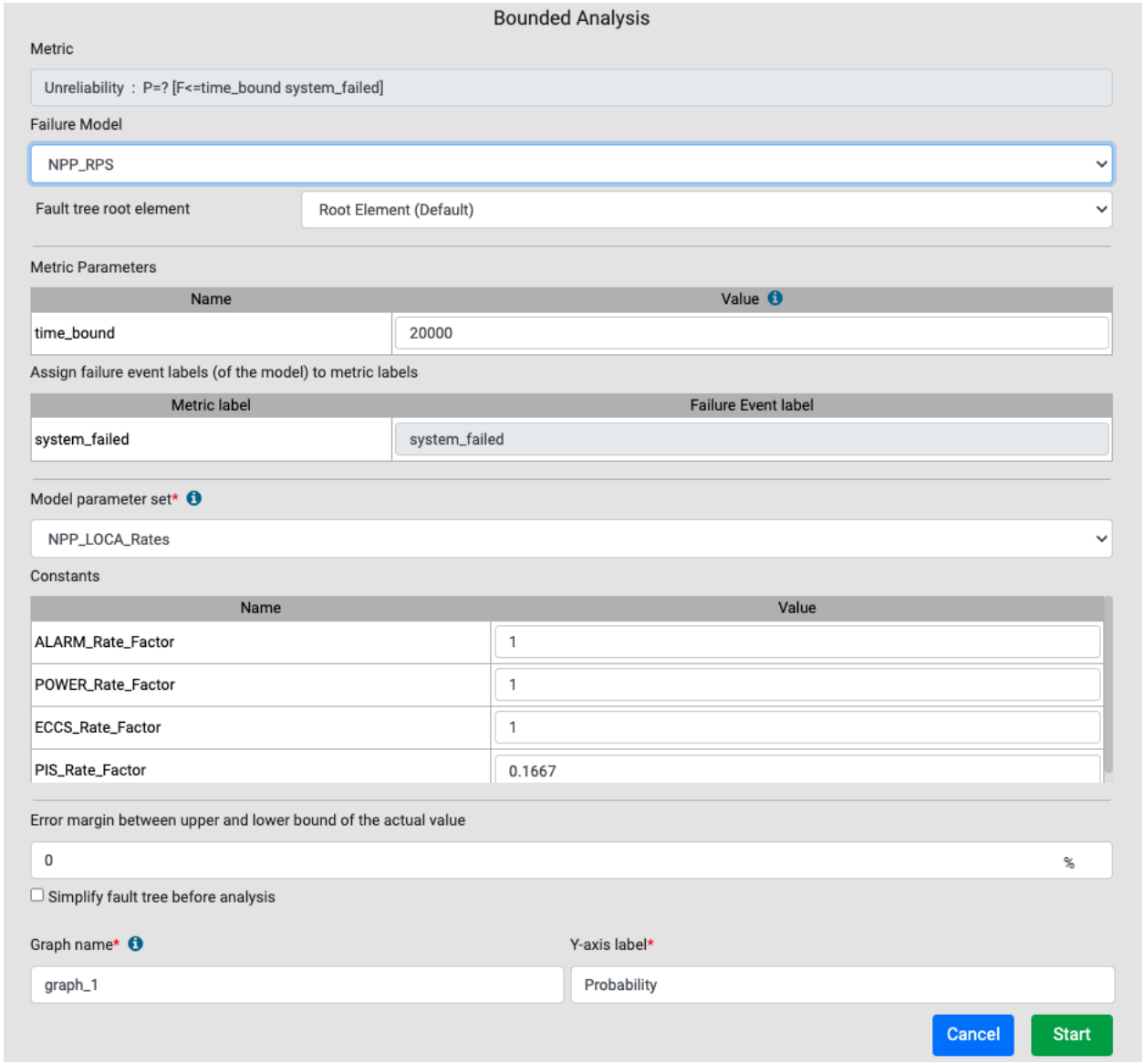
- Click the “Download” link to download the generated artifact (DFT/DRN).

  - Click the “Open in new tab” link to open the generated artifact in a new browser tab.
  - Click the “Load” link to load the generated DFT as a failure model in the current project.
- Click  icon to download the results in the selected rows in a csv format.

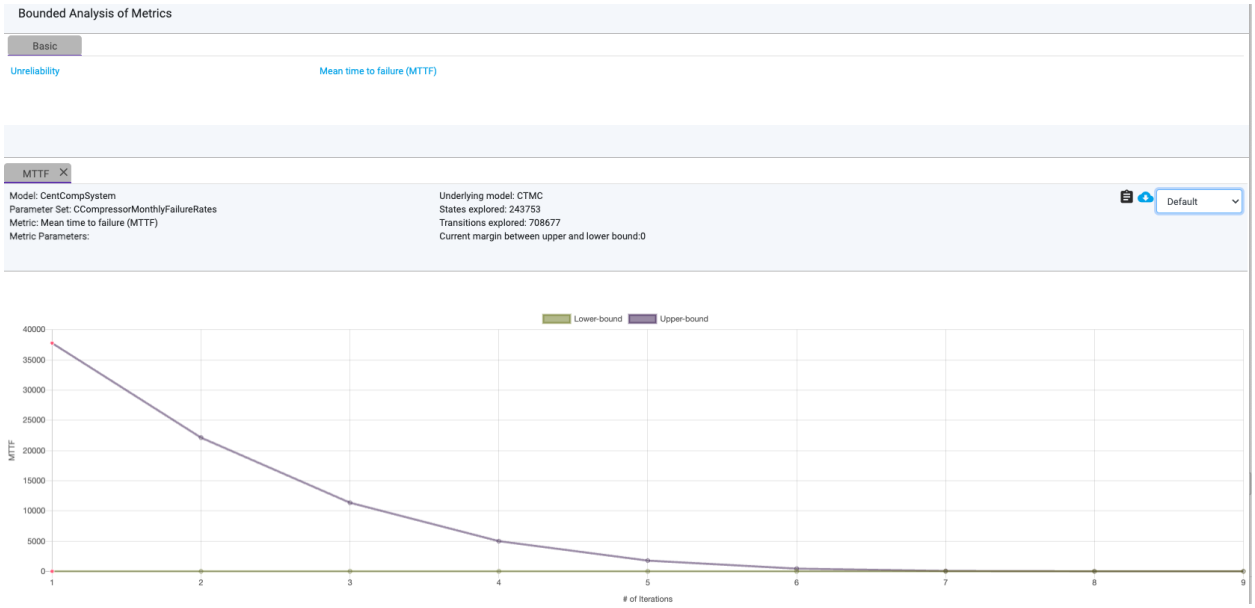
Bounded analysis

In order to compute exact results for measures using Markov analysis, first of all, the full state space is constructed and then analyzed. However, many states in the state space only marginally contribute to the result. If one is interested in an approximation of the MTTF (or the reliability), these states are of minor interest. We implemented the algorithms, proposed by Dr. Matthias Volk et. al., that generate state space on-the-fly, and then compute an upper and a lower bound to the exact results on a partially unfolded system, which might be much smaller as compared to the fully unfolded system. The approximation is sound ensuring the exact result lies between these two bounds.

Click on the “Bounded Analysis” link under “Computing” in the left panel and then click e.g. “Mean-time-to-failure” link. The following window will appear:



- All fields are filled up as described in the “Analysis” case with a few additions:
- “Error margin between upper and lower bound of the actual value”: A percentage error margin is entered in this field.
- Optionally enter “Graph name” and the label of its Y-axis. Note X-axis will always represent the number of iterations in this case.
- Bounded analysis is always done by the Markov technique.
- Click the “Start” button, the results will be displayed as:

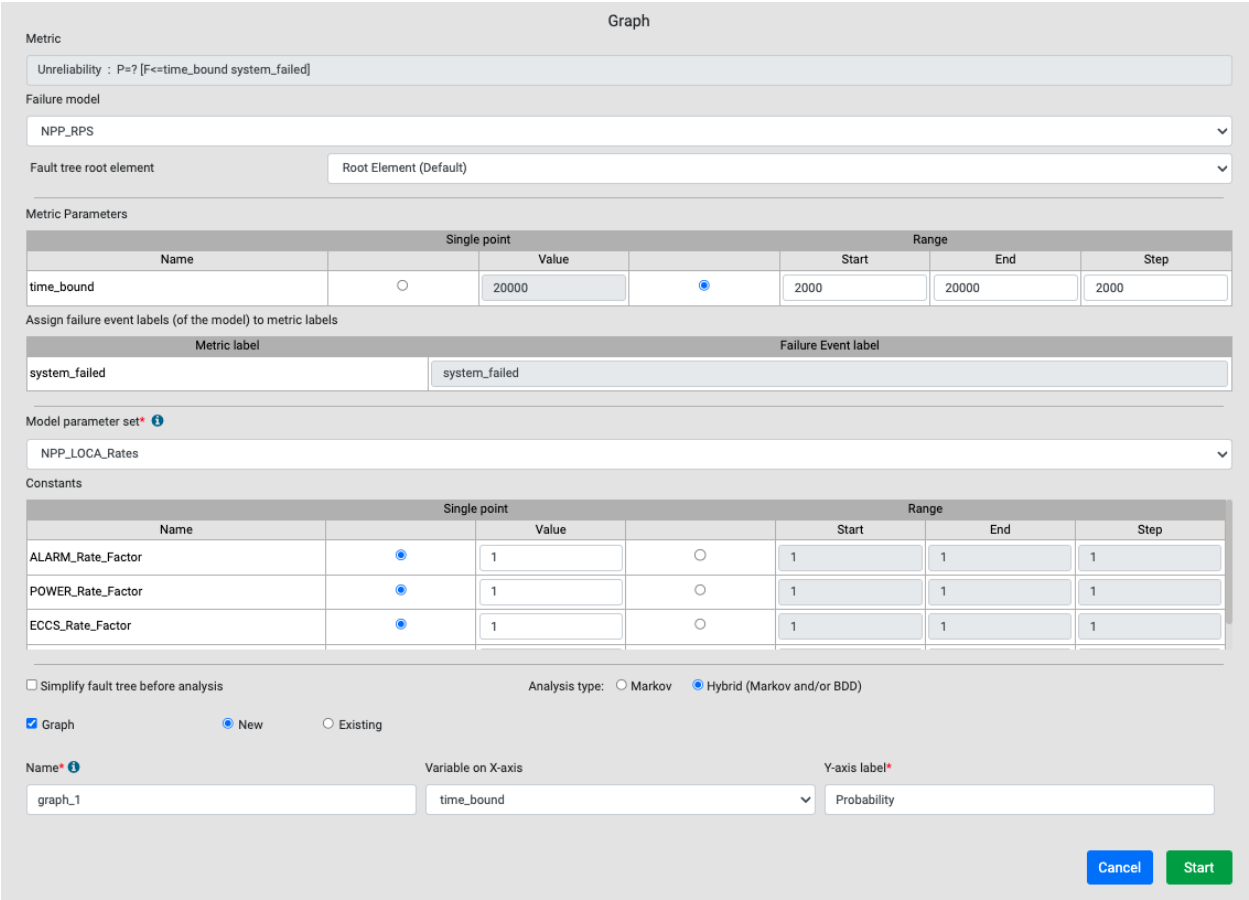


The upper line in the graph shows the upper bound whereas the lower line shows the lower bound on the actual value of the metric.

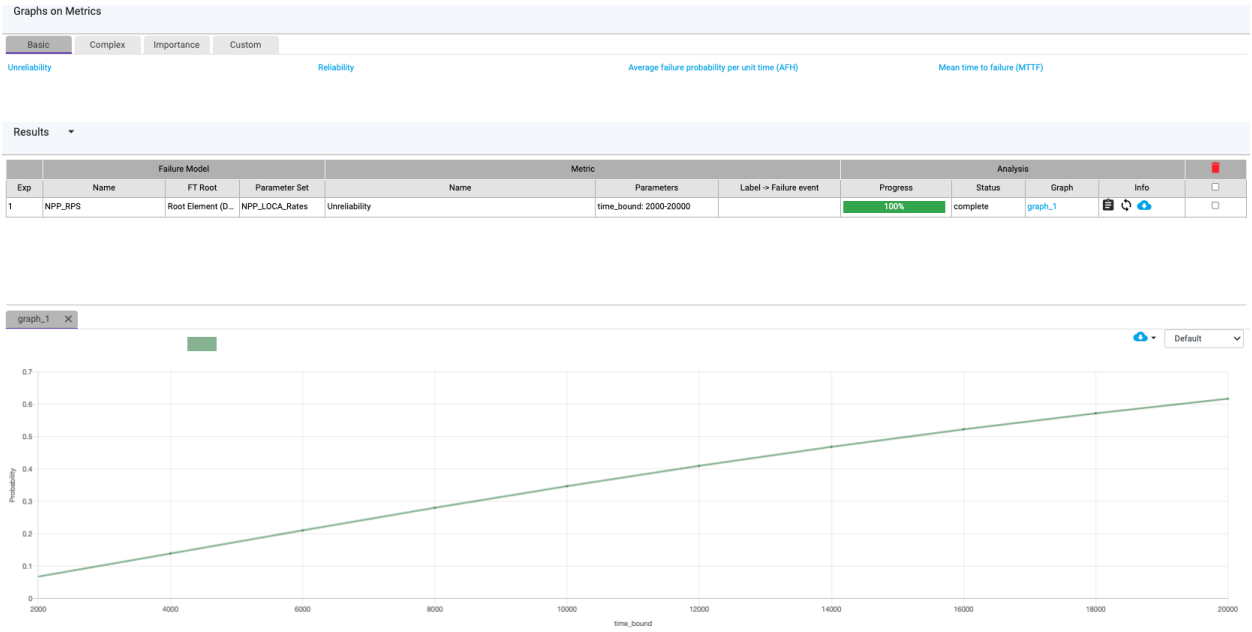
- In addition, we show the number of generated states and the transitions explored so far.
- In case you are interested in further reducing the error margin, insert a new value in the text field and click the play button .
- One can apply a log function on the values of the Y-axis by selecting it on the right side of the graph.
- The graph values can be downloaded by clicking on the icon.

Graphs

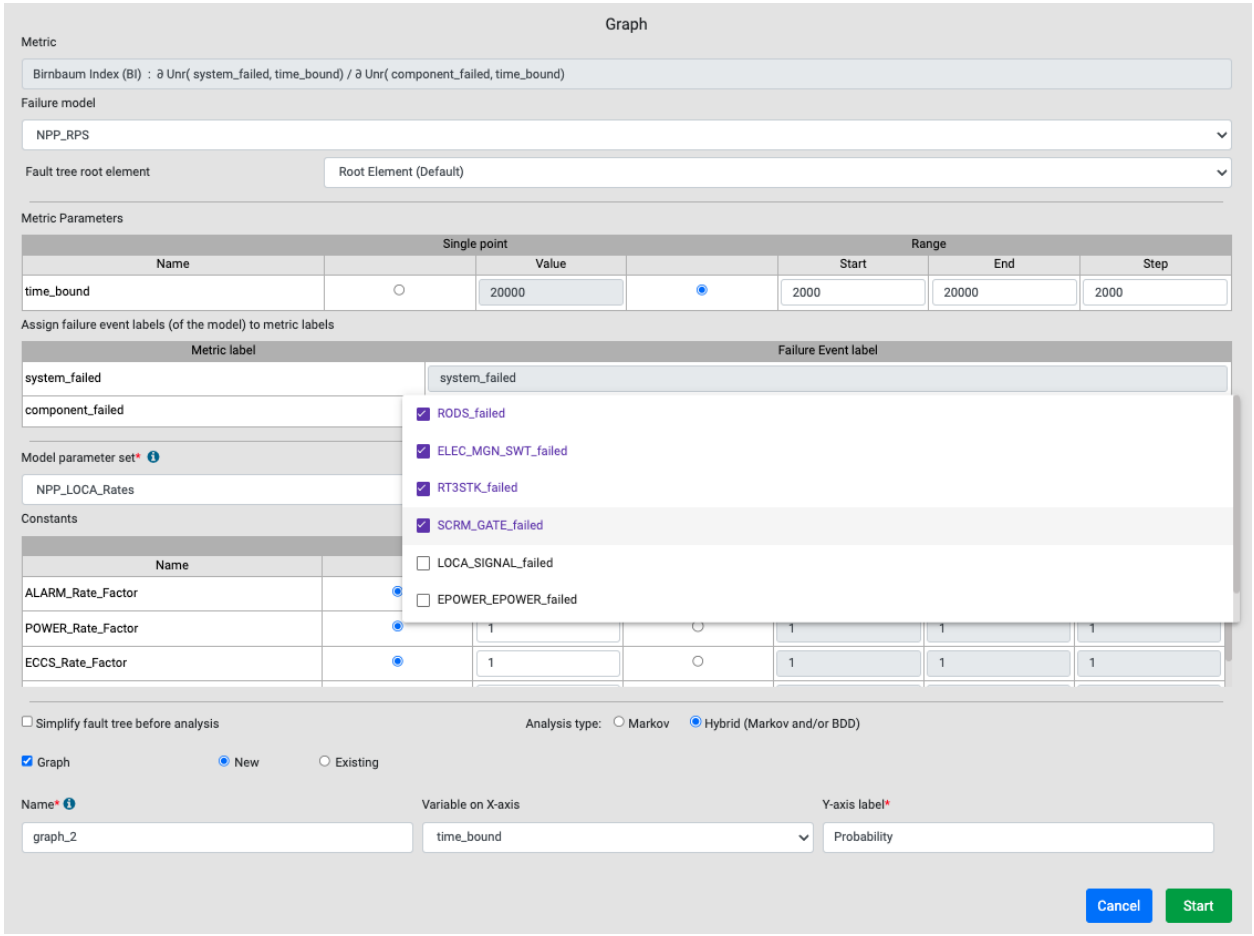
We provide an interface to plot and compare measures of interest e.g. reliability, MTTF, etc. against different parameters of interest. Click on the “Graphs” link under “Computing” in the left panel and then click “Reliability”. The following window will appear:

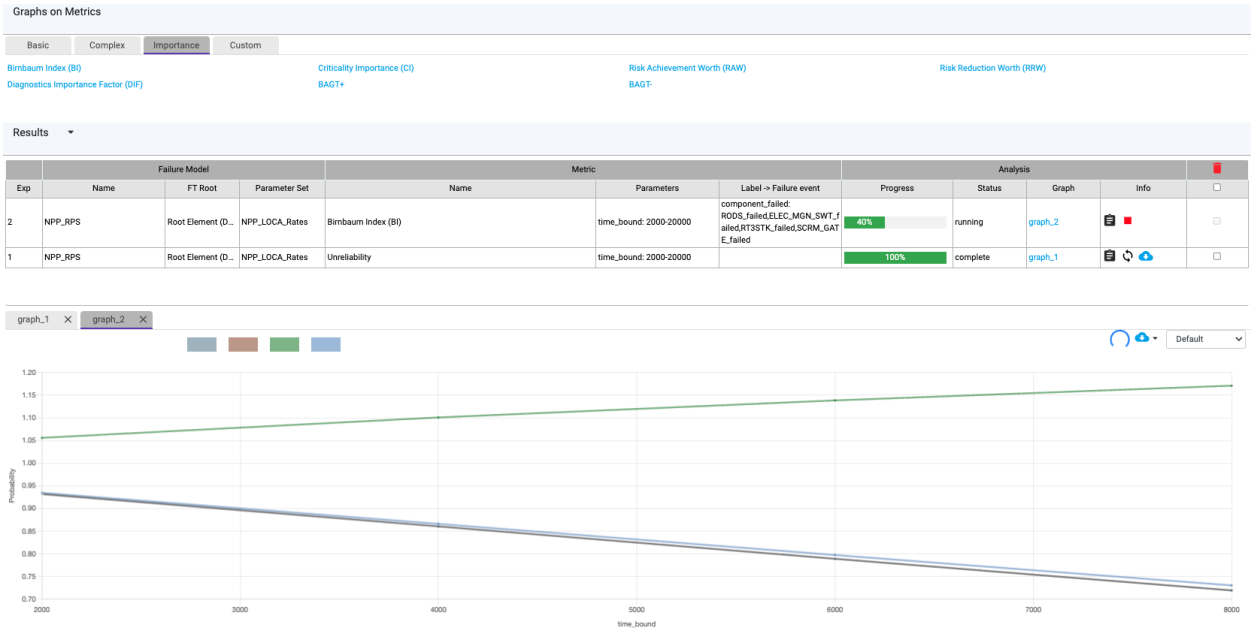


- All fields are filled up as described in the “Analysis” case with a few additions:
- One can specify a range of values of metric parameters as well as of Constants defined in the selected parameter set.
- A graph can be plotted on an existing graph as well that has the same variable on the X-axis.
- The variable on the X-axis of the graph can be selected either from the metric parameters or from the Constans of the selected parameter set.
- Click the “Start” button to display the graph:



- Click the icon to rerun the analysis and draw a graph.
- Click the icon to stop the running analysis.
- In the case of Importance measures, one can draw a plot for multiple components at the same time:


Graph
Metric
Birnbaum Index (BI) : ∂ Unr(system_failed, time_bound) / ∂ Unr(component_failed, time_bound)
Failure model
NPP_RPS
Fault tree root element
Root Element (Default)
Metric Parameters
Single point
Range
Name
Value
Start
End
Step
time_bound
20000
2000
20000
2000
Assign failure event labels (of the model) to metric labels
Metric label
Failure Event label
system_failed
system_failed
component_failed
RODS_failed
ELEC_MGN_SWT_failed
RT3STK_failed
SCRM_GATE_failed
LOCA_SIGNAL_failed
EPOWER_EPOWER_failed
Model parameter set*
NPP_LOCA_Rates
Constants
Name
ALARM_Rate_Factor
POWER_Rate_Factor
ECCS_Rate_Factor
Simplify fault tree before analysis
Analysis type:
Markov
Hybrid (Markov and/or BDD)
Graph
New
Existing
Name*
Variable on X-axis
Y-axis label*
graph_2
time_bound
Probability
Cancel
Start

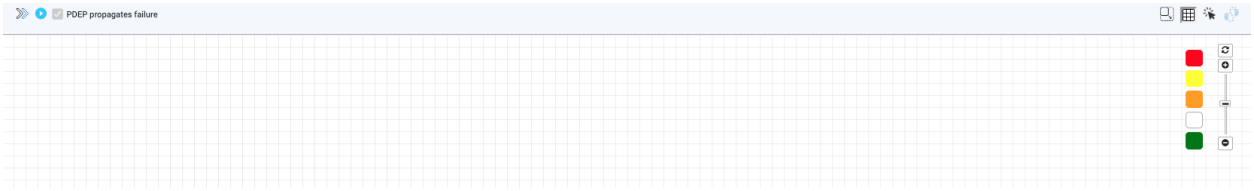

Graphs on Metrics
Basic
Complex
Importance
Custom
Birnbaum Index (BI)
Criticality Importance (CI)
Risk Achievement Worth (RAW)
Risk Reduction Worth (RRW)
Diagnostics Importance Factor (DIF)
BAGT+
BAGT-
Results
Failure Model
Metric
Analysis
Exp
Name
FT Root
Parameter Set
Name
Parameters
Label -> Failure event
Progress
Status
Graph
Info
2
NPP_RPS
Root Element (D...
NPP_LOCA_Rates
Birnbaum Index (BI)
time_bound: 2000-20000
component_failed: RODS_failed,ELEC_MGN_SWT_failed,RT3STK_failed,SCRM_GATE_failed
40%
running
graph_2
1
NPP_RPS
Root Element (D...
NPP_LOCA_Rates
Unreliability
time_bound: 2000-20000
100%
complete
graph_1

graph_1
graph_2
Default
Probability
time_bound

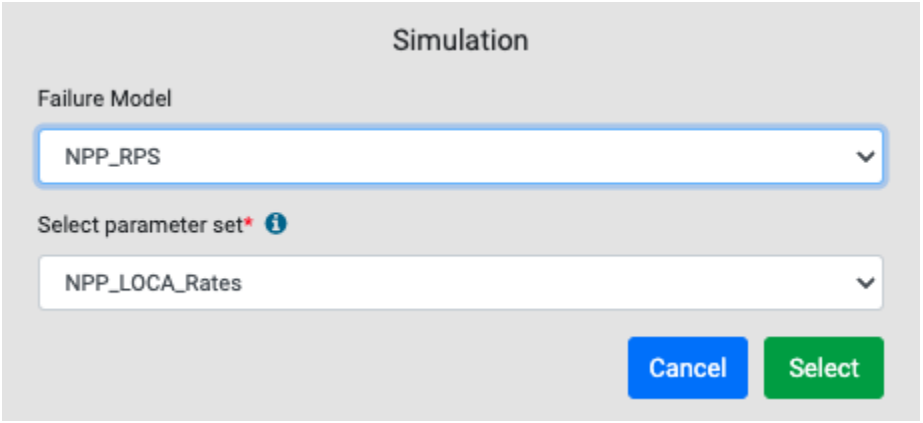
Interactive simulation

The idea is to interactively visualize a sequence of failures in a DFT. The user would start with a usual DFT and could select one of the basic events (BE) that should fail first. Based on this, the status of each DFT element (failed, operational, fail-safe, claiming in SPAREs, etc.) is redetermined and then visualized. Afterward, another BE can be selected to fail, and so forth. The main benefit of this feature is that the semantics of DFTs become much clearer as users can try out the behavior by themselves.

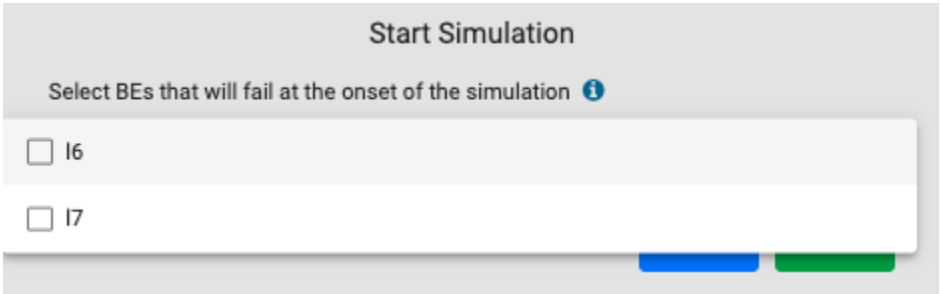
Click on the “Interactive Simulation” link under “Computing” in the left tab. The following screen will appear.

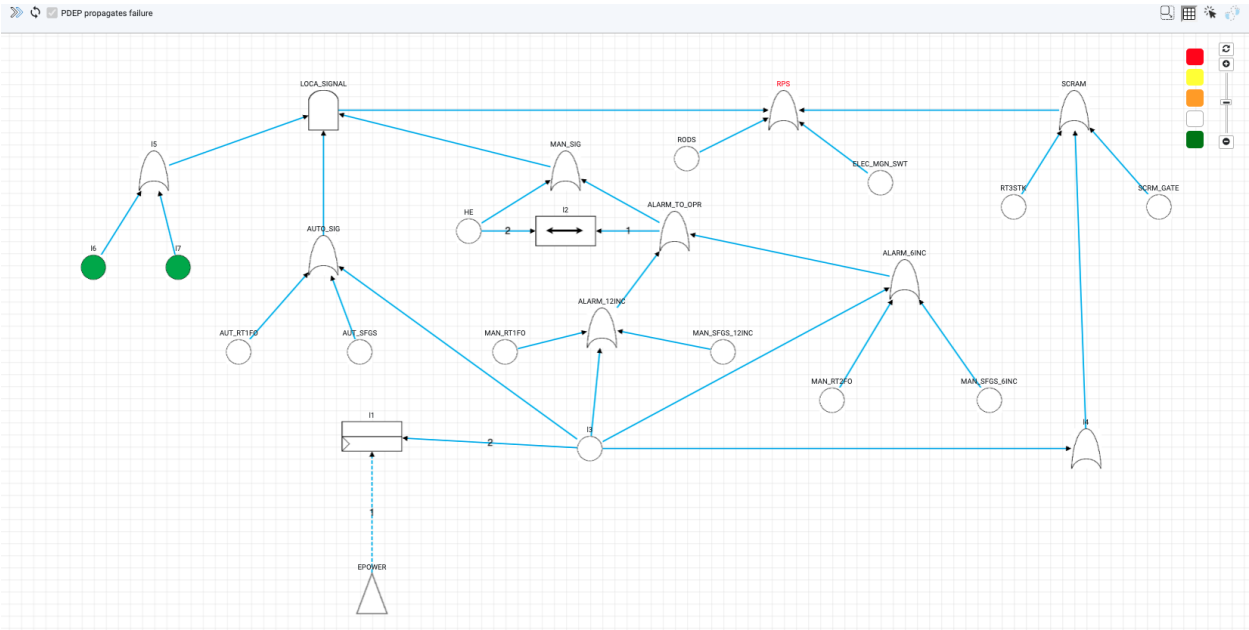


- Click on the icon  to start the simulation. The user will be prompted to select a failure model and a parameter set as:

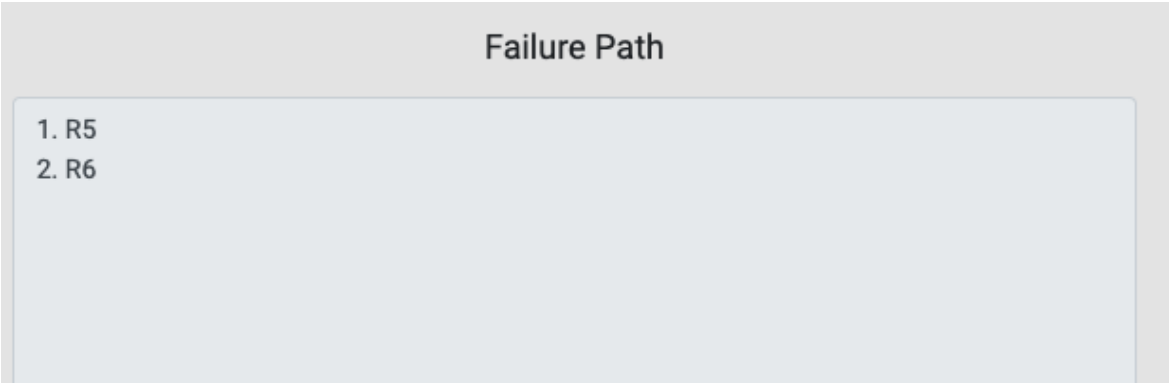


- Click the icon to start the simulation. The user will be prompted to select BEs, having constant probability distribution, that will be failed on start. On clicking the “Start” button, the simulation will start.





- “PDEP propagates failure” checkbox is selected. That means if the trigger of any PDEP fails, the gate will fail all its dependent BEs. Currently, the checkbox is disabled, which will be enabled in the next releases to give more flexibility in simulation.
- All basic events (BEs) that can fail are shown in green.
- The user clicks any green BE to fail it. Its color will be turned into Red. After this, BEs that are operational and cannot fail currently remain White, those that are in a fail-safe state are Orange, and those that are in a dont-care state are Yellow.
- The user keeps on failing green BEs, and in return, the failure keeps on moving up the tree until the top-level event turns Red showing the failure of the top-level event.
- The sequence of failures can be shown by clicking on the icon  :



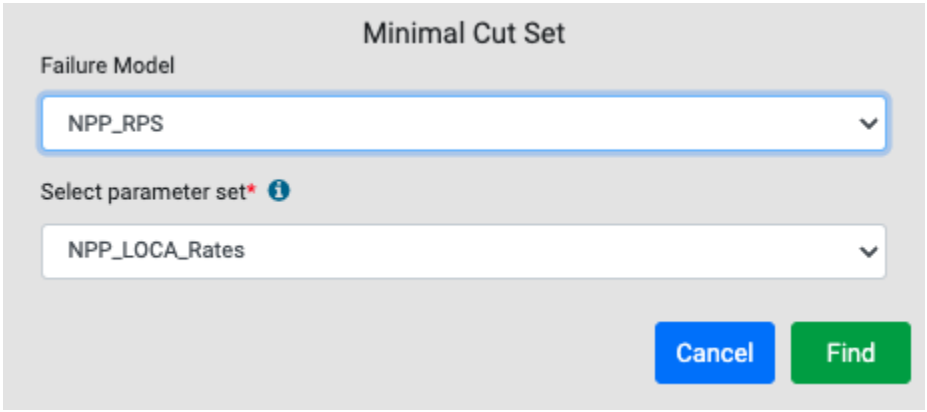
- Users can restart the simulation by clicking on the icon  .

Minimal cut set (for static fault trees)

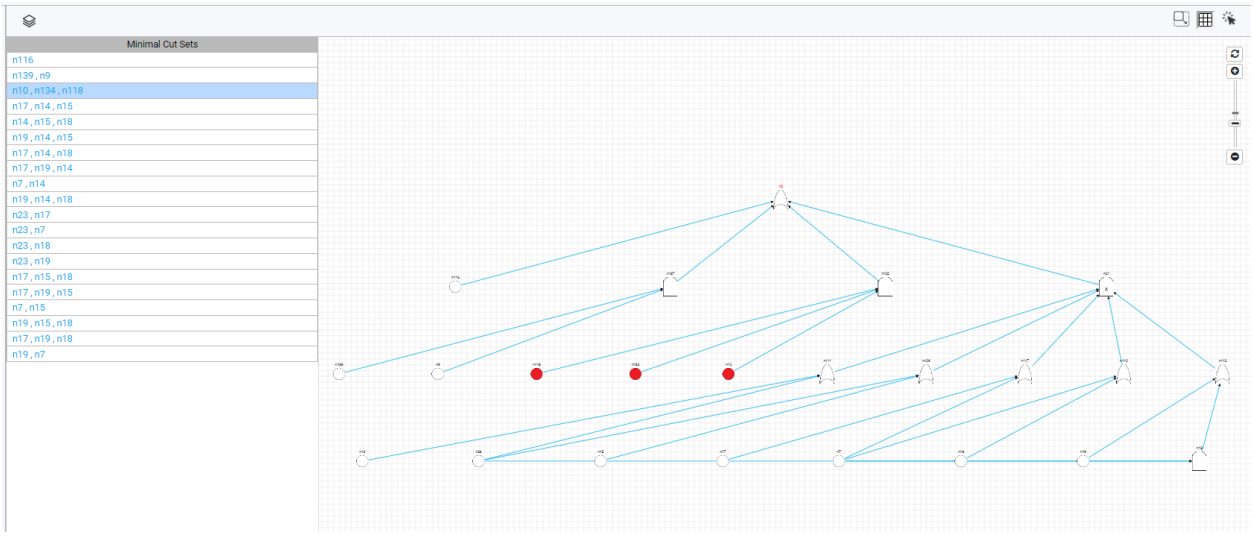
Cut sets represent sets of BEs whose failure leads to the failure of the top-level element of a fault tree. A minimal cut set is a set whose proper subset cannot be a cut set itself. Cut sets cannot be calculated for dynamic fault trees because of the dynamic nature of the system.

Click on the “Minimal cut set” link under “Computing” in the left tab. The following screen will appear.

Click on the icon  to start. The user will be prompted to select a failure model and a parameter set as:



On clicking “Find”, minimal cut sets are computed and displayed on the screen as:



- All minimal cut sets will be shown on the left of the screen.
- On clicking a cut set, the corresponding BEs will be highlighted (in Red) in the tree.

4. Bowtie Analysis Module

It contains all bowtie models (event trees) along with their parameter sets, loss sets, consequence sets, and computing methods.

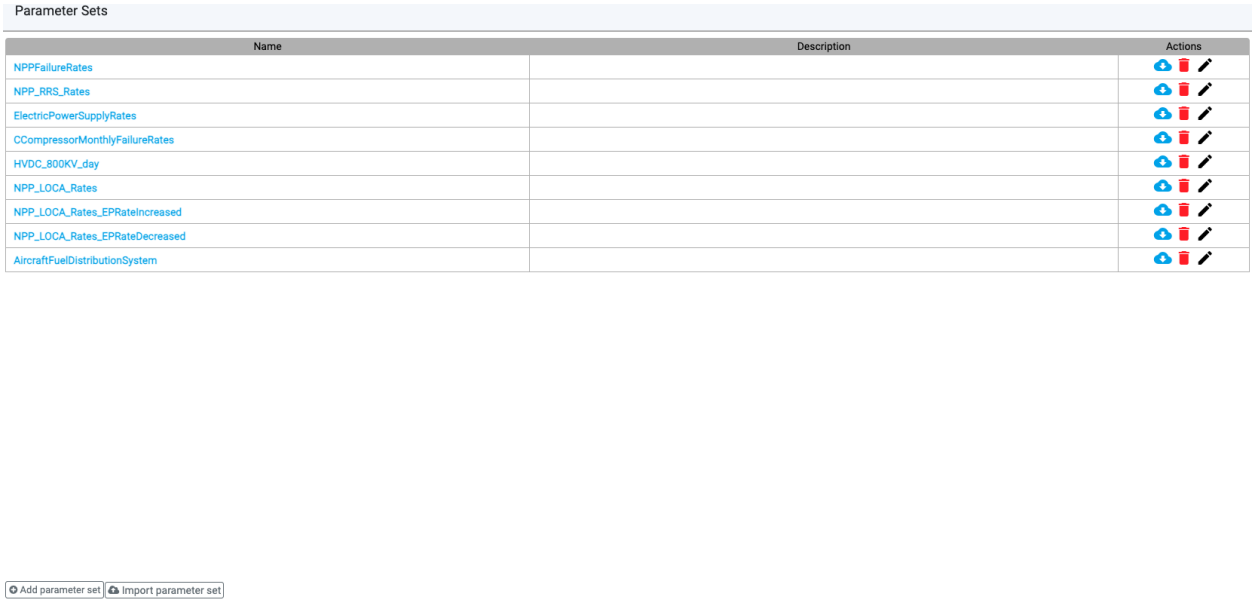
Parameter Sets

Each parameter set contains a list of parameters that are key-value pairs. They are used to specify values of e.g. probabilities, failure rates, etc. in fault tree models. By changing their values several variants of fault trees can be generated, which can then be compared with each other based on the results of metrics of interest. Each parameter set comprises:

- Constants
- Real-valued expressions

View

Click the “Parameter Sets” under “Fault Tree Analysis” in the left panel to view all existing parameter sets.



Creation

Click the “Add parameter set” button to create a new parameter.

Import

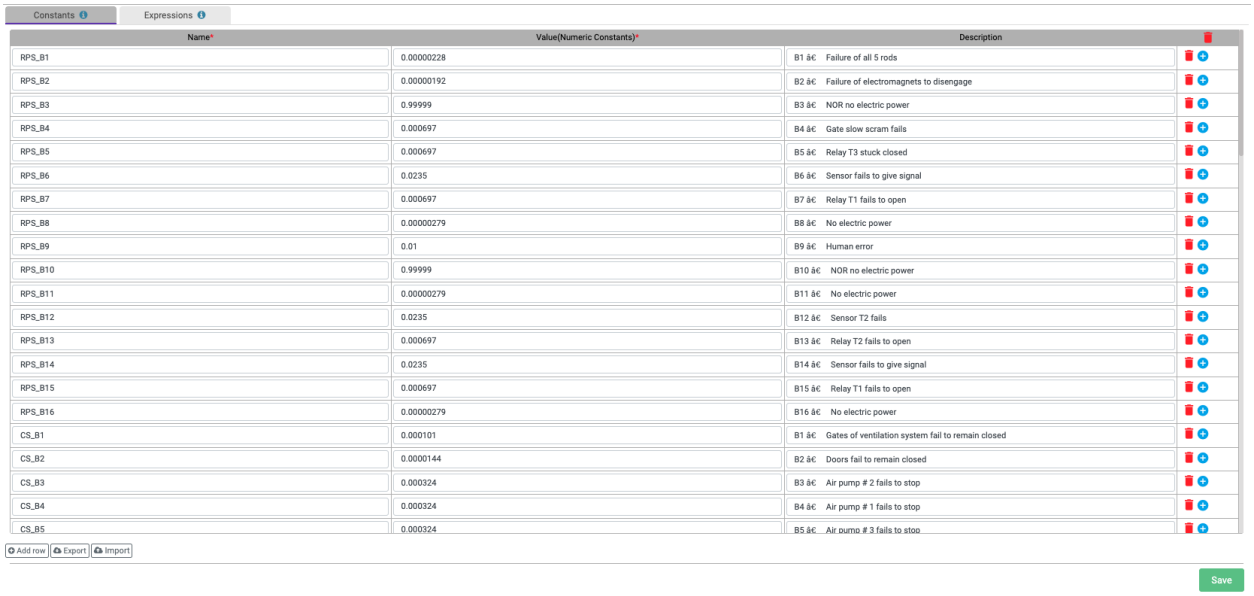
Click the “Import parameter set” to import the parameter set in .ps format (a format in which parameter sets are imported/exported in our tool).

Export

Click the  icon to export a parameter set in .ps format.

Update

Click a parameter set to update its details



Constant

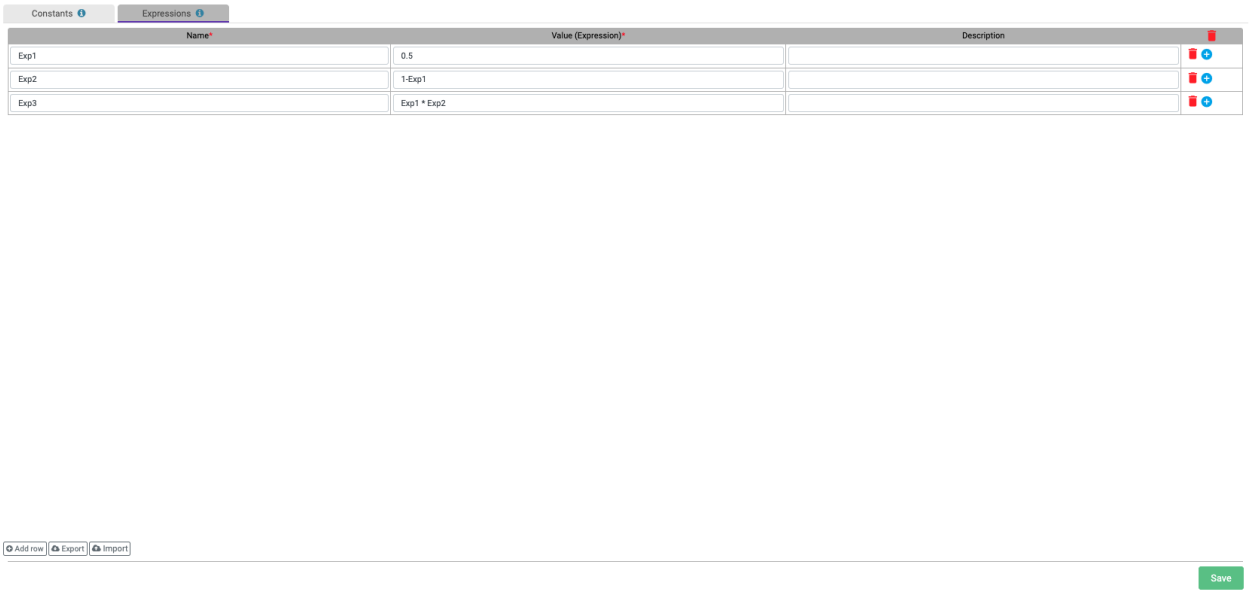
Constants can only be numeric e.g. 4, 2.3, 4e-6 etc. Their value can be changed at the time of analysis. For example, graphs can be plotted for matric results against ranges of values of constants.

- Click “Add row” to enter a new row in the table.
- Click “Export” to export constants in a csv format.
- Click “Import” to import constants from a csv file.
- Click the  icon in the last column of any row to add a new row above it.
- Click the “Save” button to save the data. This action will save data in all the tabs.

Real-value Expressions

These are non-negative, real-value expressions, which can use constants (defined above) e.g. x + 2 where x is a constant. The grammar of the expression is given here.

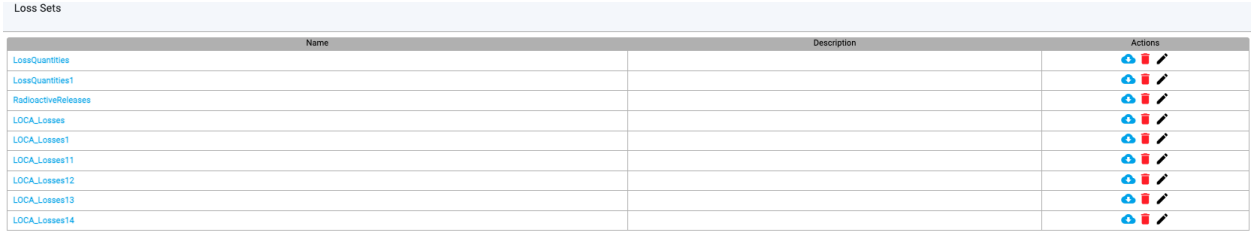
- Click “Add row” to enter a new row in the table.
- Click “Export” to export expressions in a csv format.
- Click “Import” to import expressions from a csv file.
- Click the  icon in the last column of any row to add a new row above it.
- Click the “Save” button to save the data. This action will save data in all the tabs.



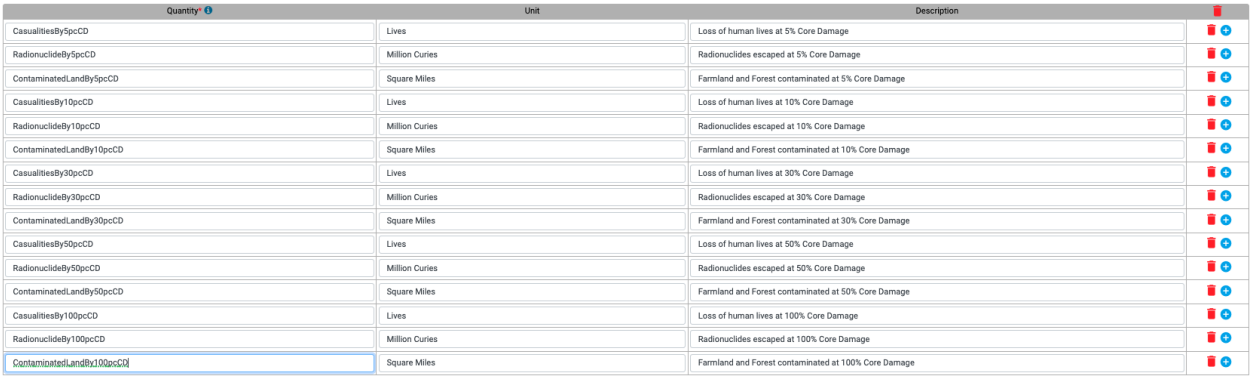
Loss Sets

A loss set contains quantities along with their units and descriptions. These quantities can be assigned values at nodes/states of event trees. During analysis, their expected values are calculated by the analysis algorithms.

Click the “Loss Sets” under “Bowtie Analysis” in the left panel to view all existing parameter sets.



- Click the “Add loss set” button to create a new loss set.
- Click the “Import loss set” to import the loss set in .rs format (a format in which loss sets are imported/exported in our tool).
- Click the icon to export a loss set in .rs format.
- Click a loss set to view its details



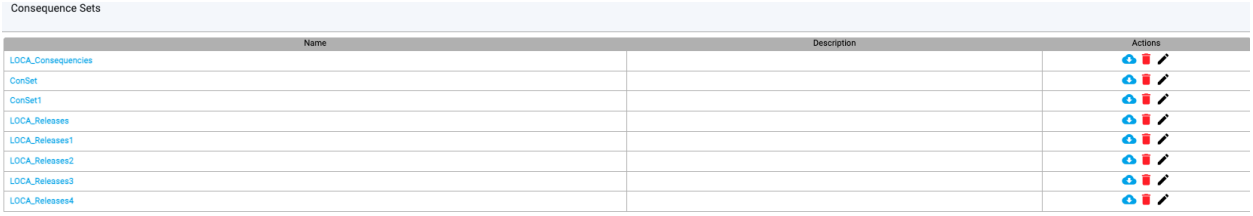
- Click the “Add row” button to add a new loss quantity in the above table.
- Click the icon in the last column of any row to add a new row above it.
- Click the “Save” button to save the data.
- Click “Export” to export loss in a csv format.

- Click “Import” to import consequences in a csv format.

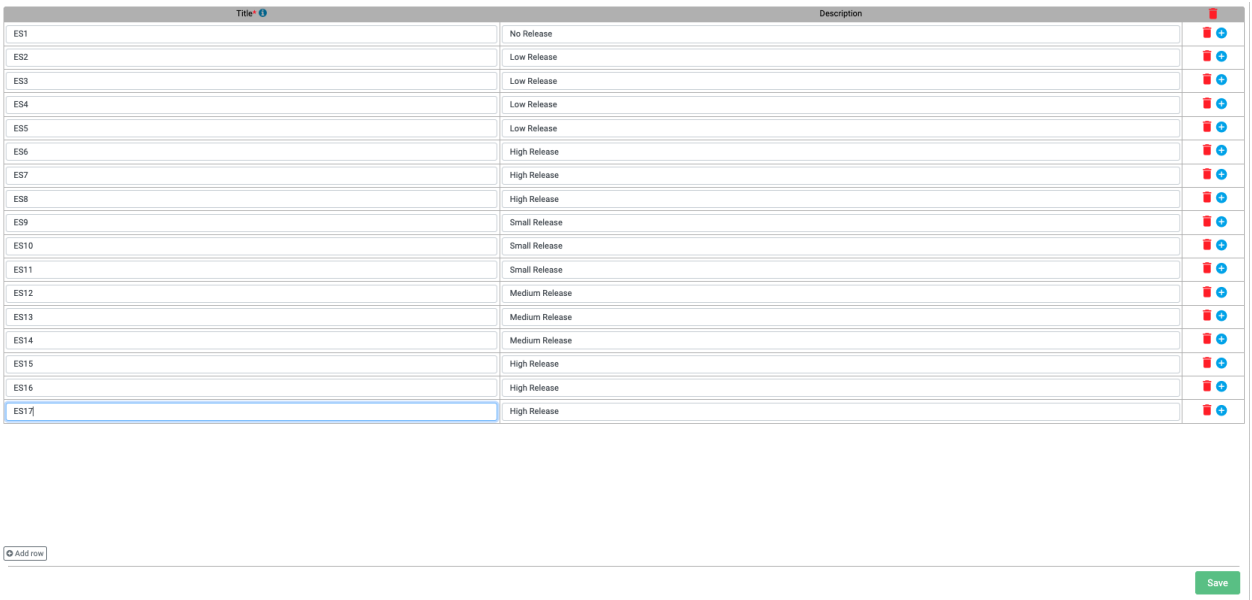
Consequence Sets

A consequence set contains outcomes along with their descriptions. These outcomes can be assigned to nodes/states in event trees. During analysis, their expected frequencies/probabilities are calculated by the analysis algorithms.

Click the “Consequence Sets” under “Bowtie Analysis” in the left panel to view all existing parameter sets.



- Click the “Add consequence set” button to create a new consequence set.
- Click the “Import consequence set” to import a consequence set in .cs format (a format in which loss sets are imported/exported in our tool).
- Click the icon to export a consequence set in .cs format.
- Click a consequence set to view its details



Title*	Description
ES1	No Release
ES2	Low Release
ES3	Low Release
ES4	Low Release
ES5	Low Release
ES6	High Release
ES7	High Release
ES8	High Release
ES9	Small Release
ES10	Small Release
ES11	Small Release
ES12	Medium Release
ES13	Medium Release
ES14	Medium Release
ES15	High Release
ES16	High Release
ES17	High Release

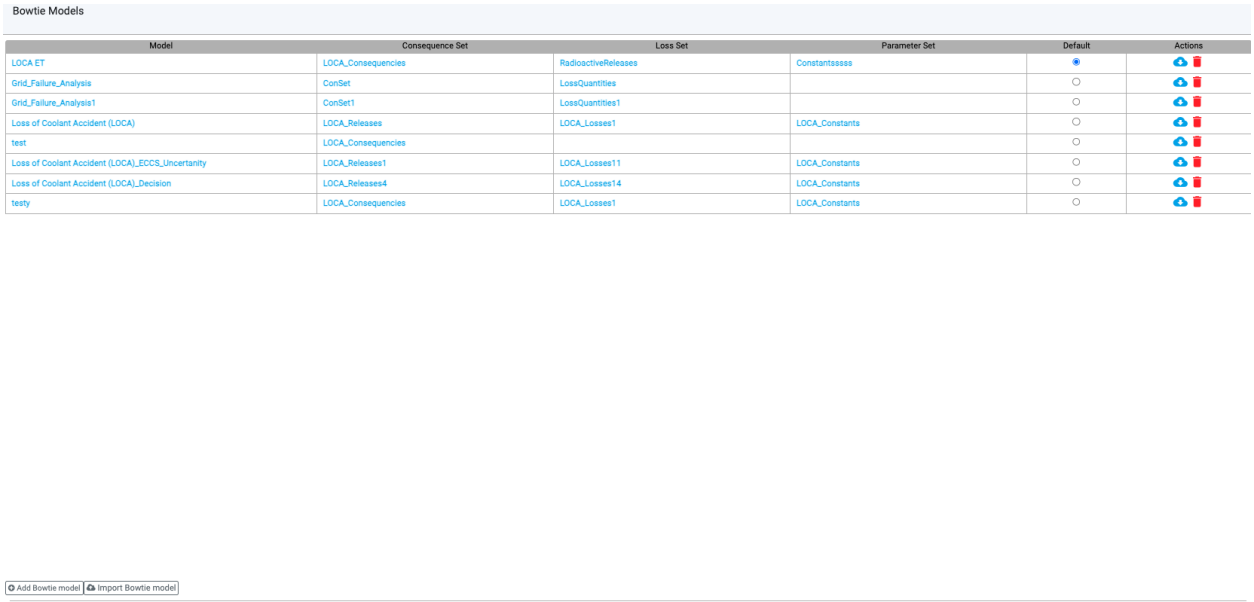
- Click the “Add row” button to add a new consequence in the above table.
- Click the icon in the last column of any row to add a new row above it.
- Click the “Save” button to save the data.
- Click “Export” to export consequences in a csv format.

Bowtie Models

It contains multiple Bowtie models represented as event trees in a graphical way.

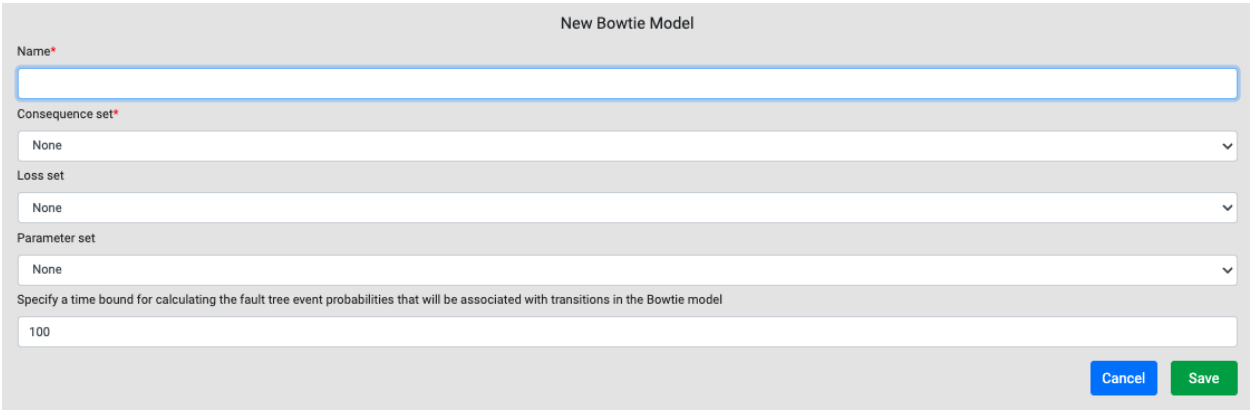
View

Click on the “Bowtie Models” under “Bowtie Analysis” in the left panel to display all Bowtie models in a table. A model that is worked upon the most can be selected as a default model by selecting the corresponding radio button.



Creation

Click the “Add Bowtie model” to create a new model.



Enter all mandatory fields and click the “Save” button.

Export

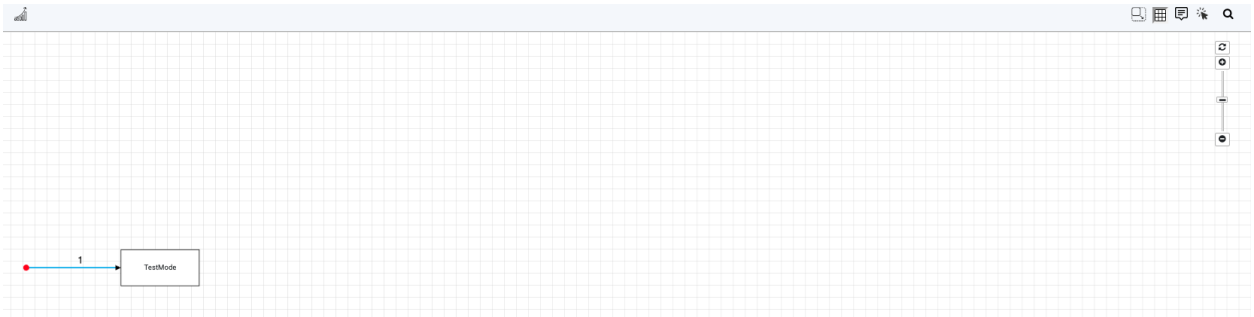
Click the icon in the corresponding row to download the bowtie model in .et format (a format used in the SAFEST tool to import/export bowtie models).

Import

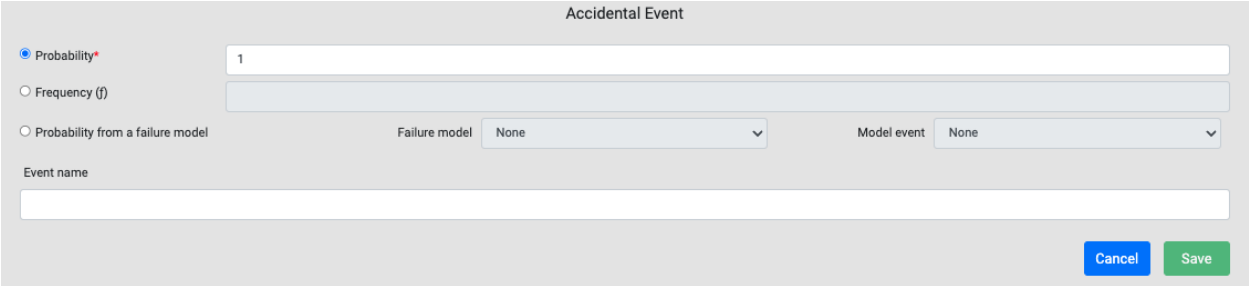
Click the “Import Bowtie model” button to import a Bowtie model (saved in .et format) from your drive.

Update

Click on the model in the table to open it in the canvas.



- Click on the first arrow to update the information of “Accidental Event” (initiating event).

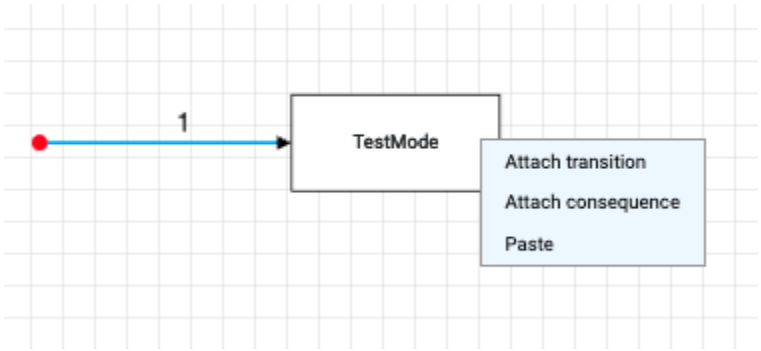


For the Accidental Event, select the respective radio button to enter its:

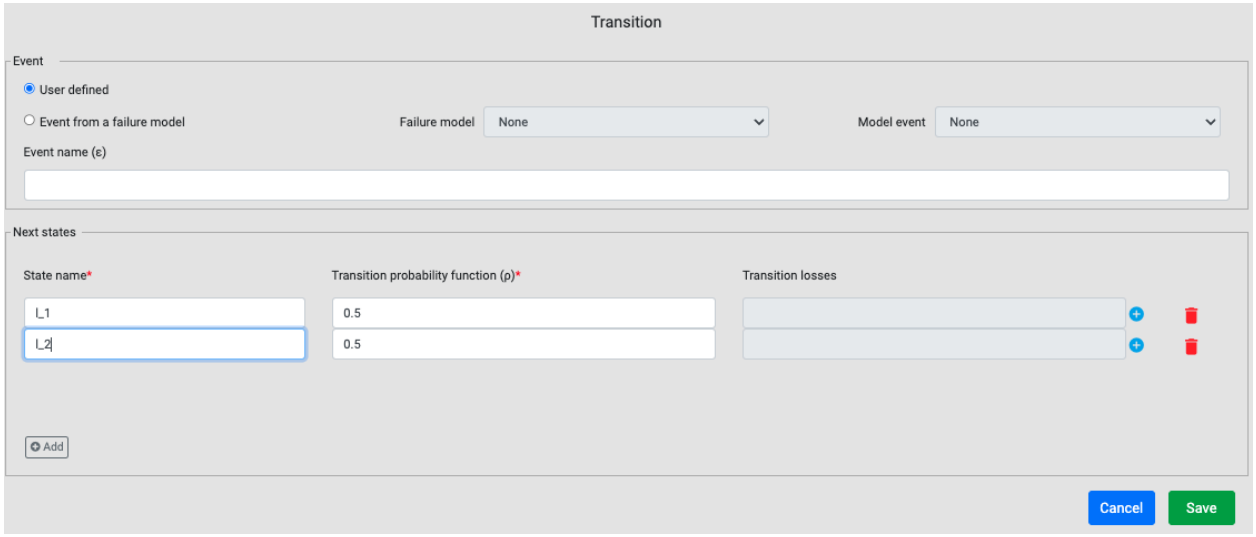
- Probability,
- Probability from a failure model – The “Failure model” dropdown will be filled with all failure models that exist under the “Failure models” under the “Fault Tree Analysis” in the left panel. Now select a failure model from the “Failure model” dropdown, it will populate the “Model event” dropdown with all the failure events (basic as well as compound) defined for the selected failure model. Now select an event from the “Model event” dropdown. It will populate the “Event name” field with the name of the model concatenated with the name of the event. One can update it to make the name simple.
- Frequency

Optionally enter the name of the event in the “Event name” field.

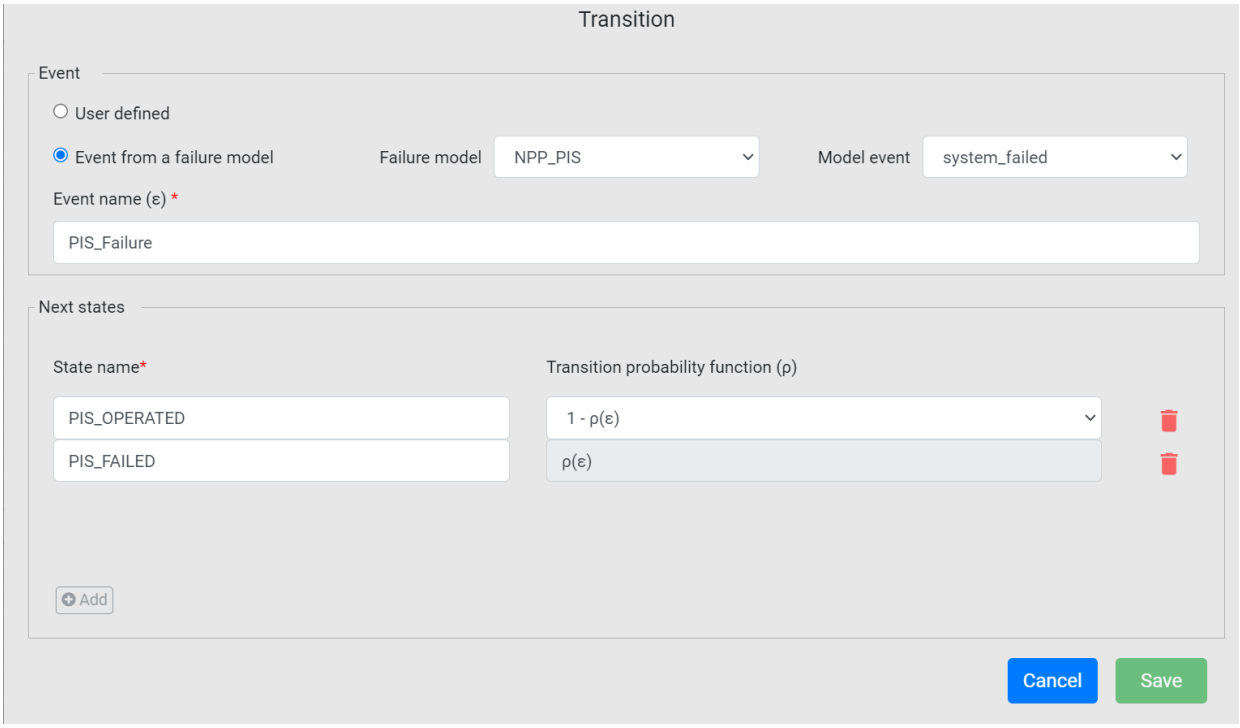
- Right-click on a node in order to add a transition, a consequence, or past a transition (which is already copied).



- Click the “Attach transition” to add a new transition from the selected node. You can add multiple transitions from a given node. Note that a transition can only be attached with a node having no consequence attached to it.



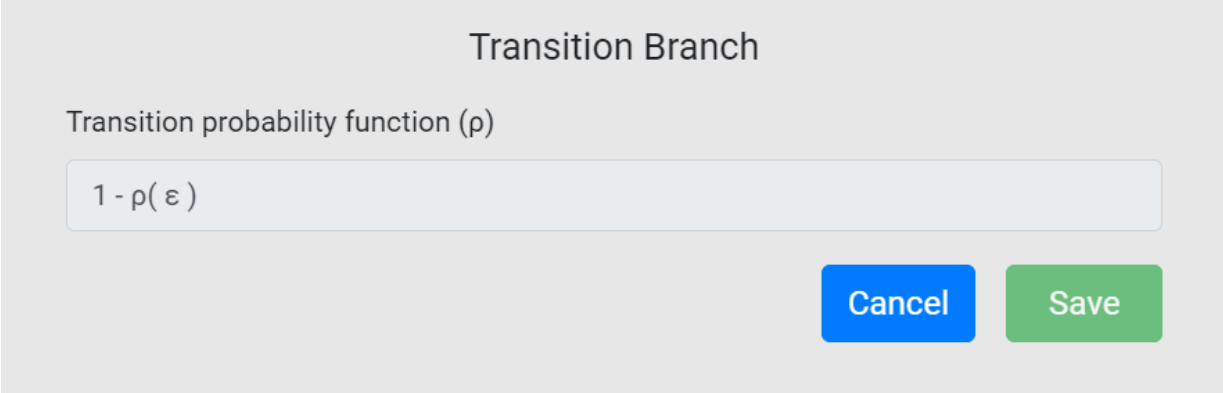
  - The event that triggers the transition can be
    - User-defined, or
    - Event from a failure model (DFT).
  - In case the event is from a failure model (fault tree), select the “Event from a failure model” radion button.
    - It will populate the “Failure model” with all failure models that exist under the “Failure models” under the “Fault Tree Analysis” in the left panel.
    - Now select a failure model from the “Failure model” dropdown, it will populate the “Model event” dropdown with all the failure events (basic as well as compound) defined for the selected failure model.
    - Select an event from the “Model event” dropdown. It will populate the “Event name” field with the name of the model concatenated with the name of the event. One can update it to make the name simple. It will also add two rows in the “Next States” table - one is annotated with the probability of the occurrence of the event and the other one with its complement.
    - Note that only two branches are possible when the triggering event is taken from a failure model.



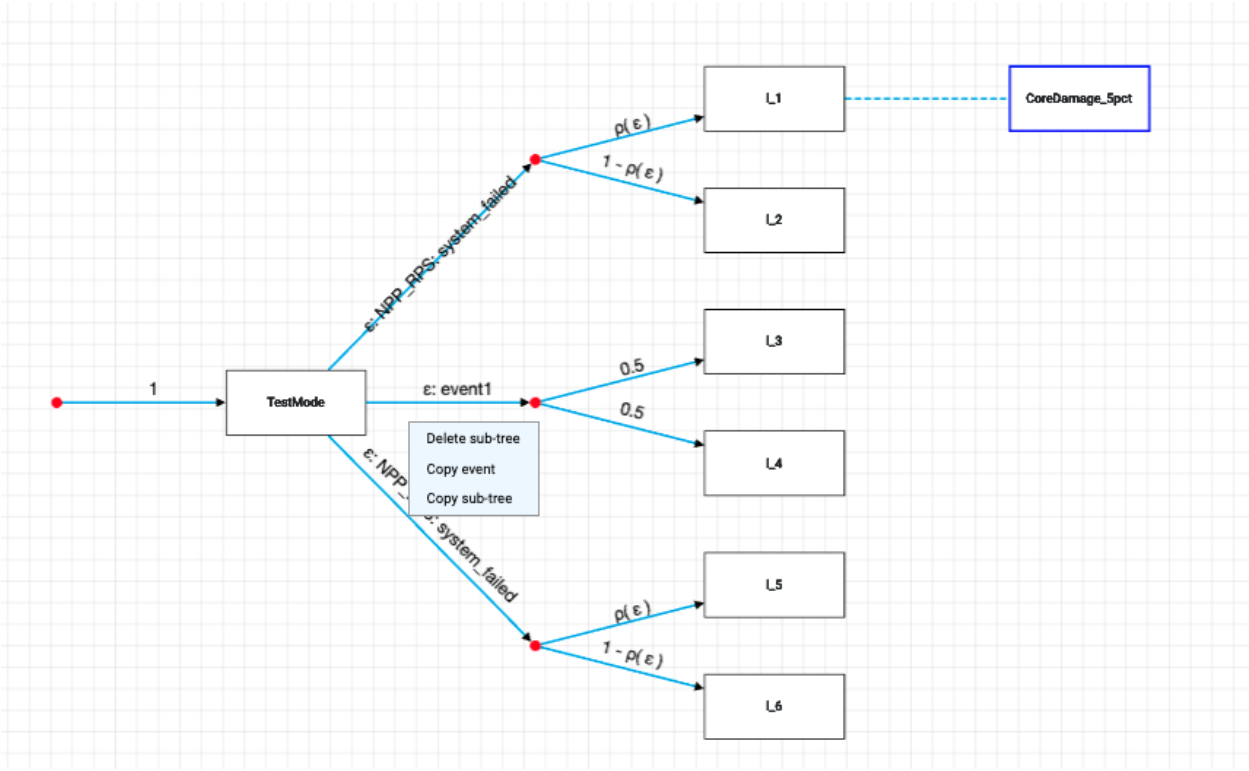
- One can update the names of the next states.

- In case the event is user-defined, select the “User-defined” radio button. Fill up the name of the event.
  - Multiple branches of transition can be added in this case. Click the “Add” button to add a new branch in the “Next States” table.
  - Add transition probability function for each branch of transition. It can be a constant value, a constant parameter, or an expression parameter. Note that the constant parameter or the expression parameter should be defined in a parameter set and attached to the model before analysis.
  - Note that the sum of probabilities of all branches should be equal to one. Otherwise, there will be an error at the time of analysis.

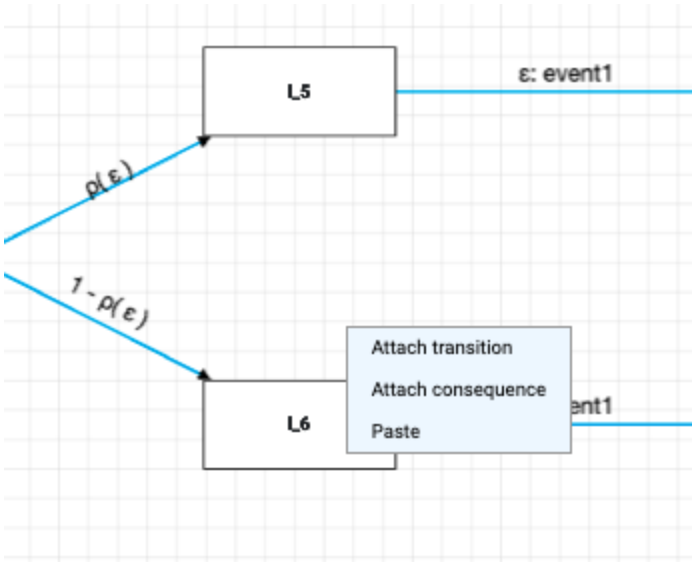
- Click a transition to update it. The popup, which comes up while adding a new transition, will appear.
- Click on a transition branch to update it. The following popup comes up. One can update the transition probability only if it has been specified by the User i.e. it is not computed from an attached fault tree.



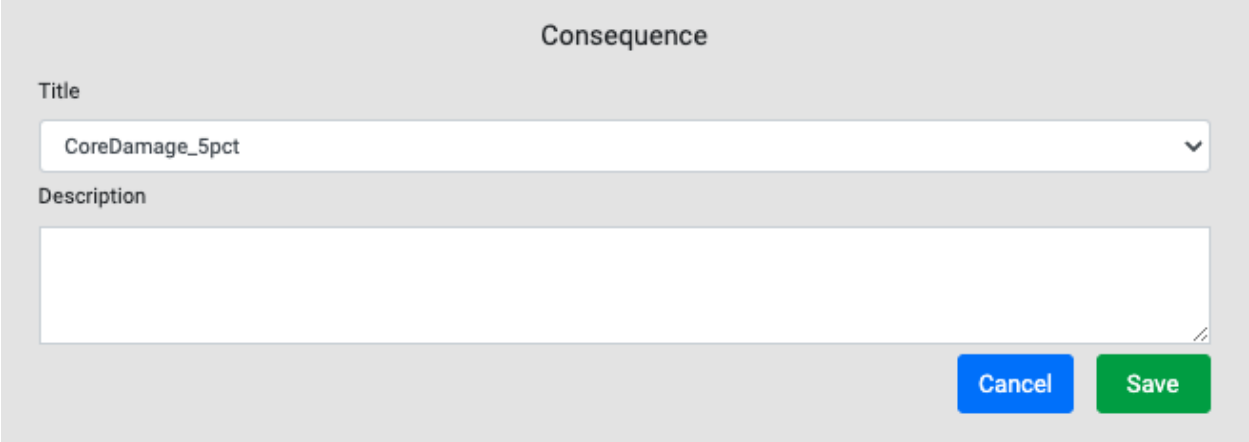
- Right-click on a transition in order to copy/delete a sub-tree originating from the transition, or copy the transition (along with its branches).



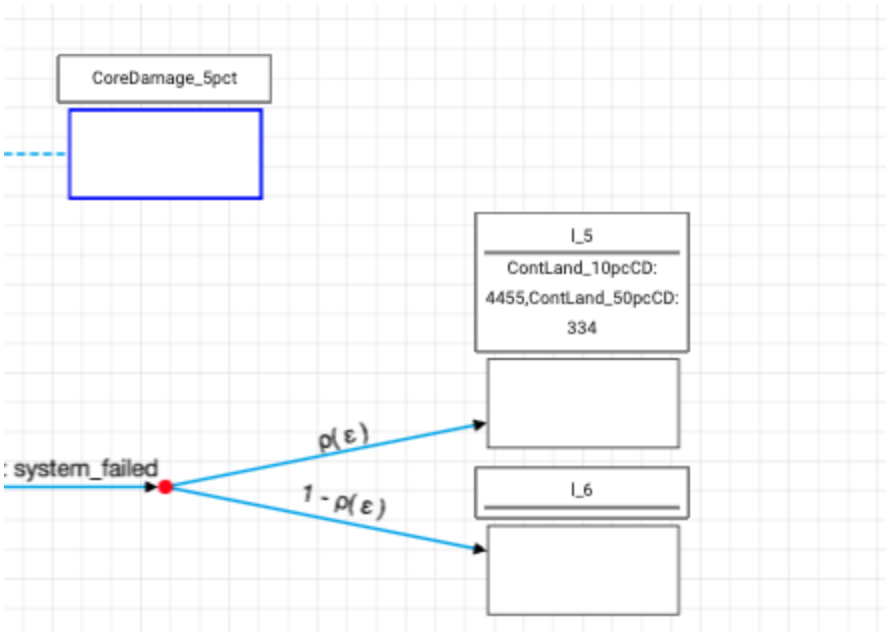
- Copy a transition/sub-tree and then right-click on a node in order to paste it. Note that transition branches cannot be copy-pasted individually.



- Click the “Attach consequence” in order to attach a consequence with a node. All consequences in the “Consequence Set” attached to the model will be available in the “Title” dropdown. Select one of the consequences and click the “Save” button. Note that a consequence can be attached to a node if and only if there is no other transition attached to the node.



- Click on the icon  to enable the navigator at the bottom of the screen.
- Click on the icon  to show the grid on the screen.
- Click on the icon  to show the summary information about each element on the screen.



- Click on the icon  to display summary information about an element on hovering.
- Click on the icon  to search for any element on the screen.

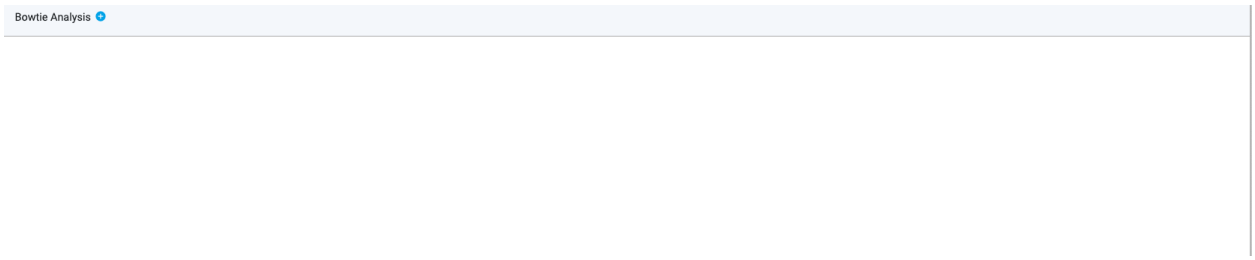
Computing

Bowtie models can be analyzed for consequence frequencies/probabilities and expected values of (loss) quantities.

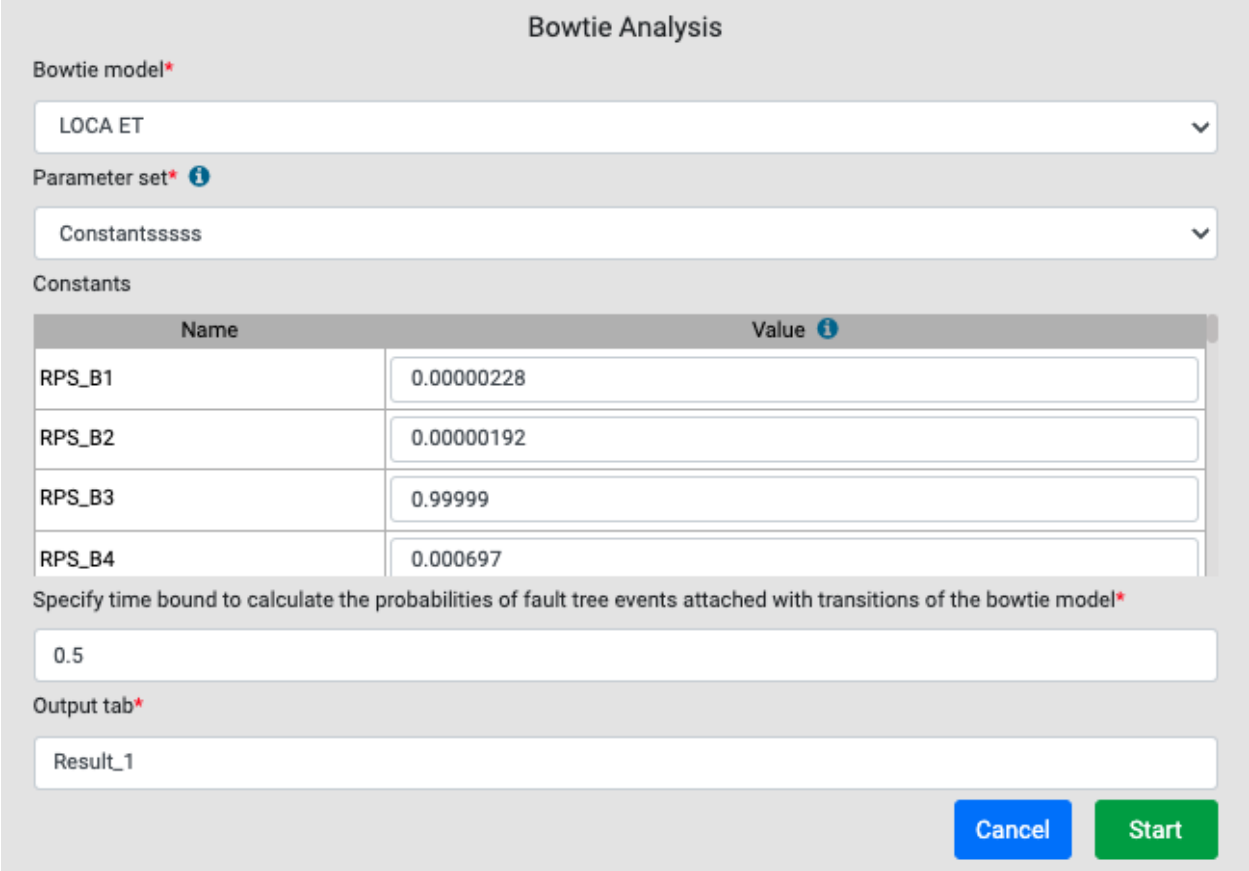
- Analysis – the exact results of consequence frequencies/probabilities and expected values of (loss) quantities can be computed.
- Graphs – the exact results of consequence frequencies/probabilities and expected values of (loss) quantities can be graphed against time duration.

(Exact) Analysis

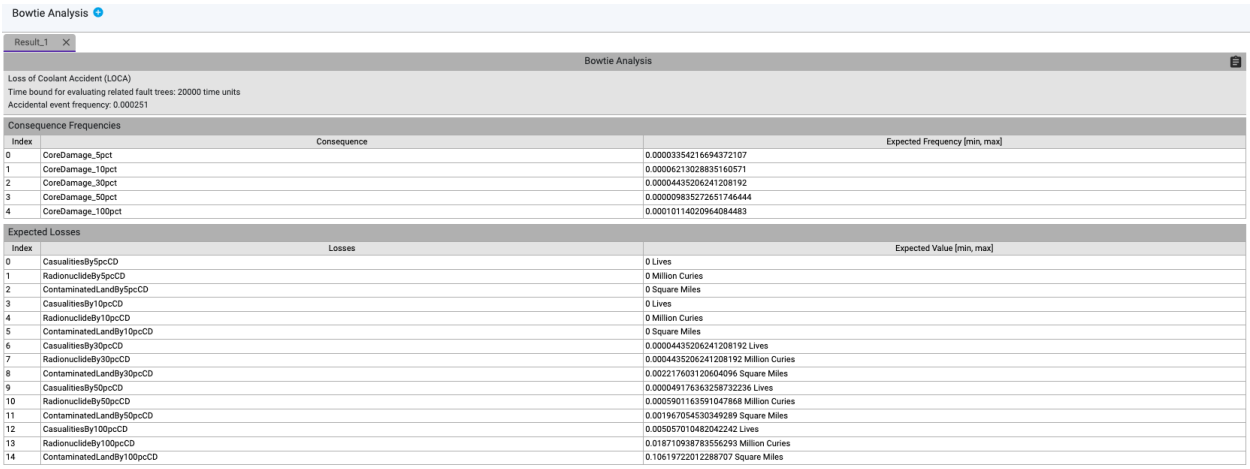
Click on the “Analysis” link under “Computing” in the left panel. The following window will appear.



Click the ⊕ icon and the following popup will show up:



- Select the bowtie model that you want to analyze from the “Bowtie mode” dropdown – it shows all bowtie models that exist under the “Bowtie Models” in the left panel.
- Change the parameter set if needed. The selected parameter set should contain all constants/expressions that have been used in the bowtie model as parameters.
- The constants defined in the selected parameter set (above) are shown in the table, which can be updated if needed.
- “Specify time bound to calculate the probabilities of fault tree events attached with transitions of the bowtie model” field contains the default value associated with the selected bowtie model (above). It can be updated here if needed. The failure probability of all failure models attached to the selected bowtie model will be analyzed for this time-bound.
- Enter the name of the tab where the results of the analysis will be displayed.
- Click the “Start” button to start the analysis.



Bowtie Analysis

Result_1

Bowtie Analysis

Loss of Coolant Accident (LOCA)
Time bound for evaluating related fault trees: 20000 time units
Accidental event frequency: 0.000251

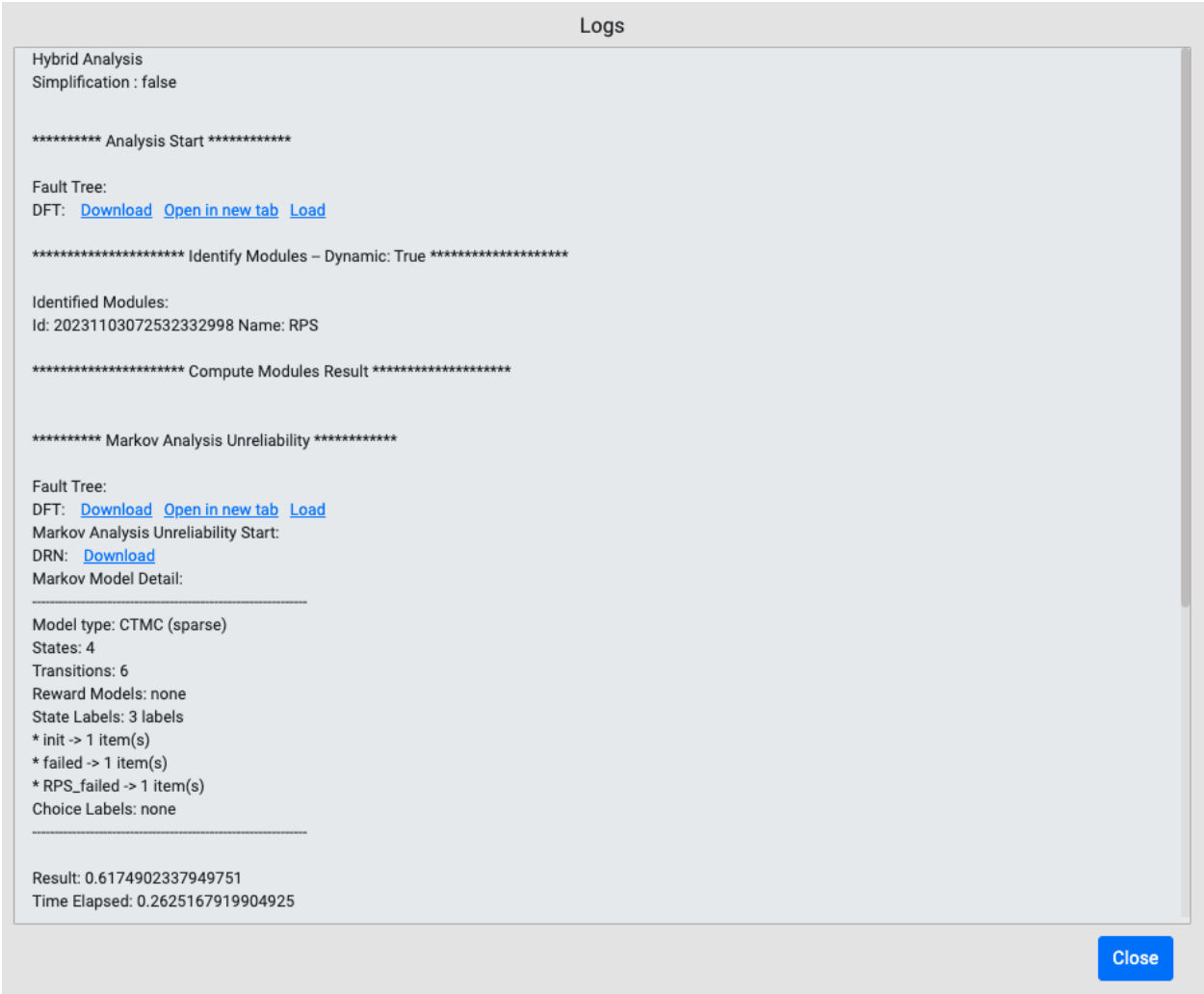
Consequence Frequencies

Index	Consequence	Expected Frequency [min, max]
0	CoreDamage_5pct	0.00003354216694372107
1	CoreDamage_10pct	0.00006213028835160571
2	CoreDamage_30pct	0.00004435206241208192
3	CoreDamage_50pct	0.00000983527651746444
4	CoreDamage_100pct	0.00010114020964084483

Expected Losses

Index	Losses	Expected Value [min, max]
0	CasualitiesBy5pcCD	0 Lives
1	RadionuclideBy5pcCD	0 Million Curies
2	ContaminatedLandBy5pcCD	0 Square Miles
3	CasualitiesBy10pcCD	0 Lives
4	RadionuclideBy10pcCD	0 Million Curies
5	ContaminatedLandBy10pcCD	0 Square Miles
6	CasualitiesBy30pcCD	0.00004435206241208192 Lives
7	RadionuclideBy30pcCD	0.00044352062412081920 Million Curies
8	ContaminatedLandBy30pcCD	0.00221760312060409 6 Square Miles
9	CasualitiesBy50pcCD	0.00004917636325873223 6 Lives
10	RadionuclideBy50pcCD	0.00059011635910478 68 Million Curies
11	ContaminatedLandBy50pcCD	0.00196705453034928 9 Square Miles
12	CasualitiesBy100pcCD	0.00505701048204224 2 Lives
13	RadionuclideBy100pcCD	0.01871093878355629 3 Million Curies
14	ContaminatedLandBy100pcCD	0.10619722012288707 Square Miles

- Click the  icon to view the log of the analysis.



Logs

Hybrid Analysis
Simplification : false

********** Analysis Start ************

Fault Tree:
DFT: Download Open in new tab Load

********** Identify Modules – Dynamic: True **********

Identified Modules:
Id: 20231103072532332998 Name: RPS

********** Compute Modules Result **********

********** Markov Analysis Unreliability ************

Fault Tree:
DFT: Download Open in new tab Load
Markov Analysis Unreliability Start:
DRN: Download
Markov Model Detail:

Model type: CTMC (sparse)
States: 4
Transitions: 6
Reward Models: none
State Labels: 3 labels
* init -> 1 item(s)
* failed -> 1 item(s)
* RPS_failed -> 1 item(s)
Choice Labels: none

Result: 0.6174902337949751
Time Elapsed: 0.2625167919904925

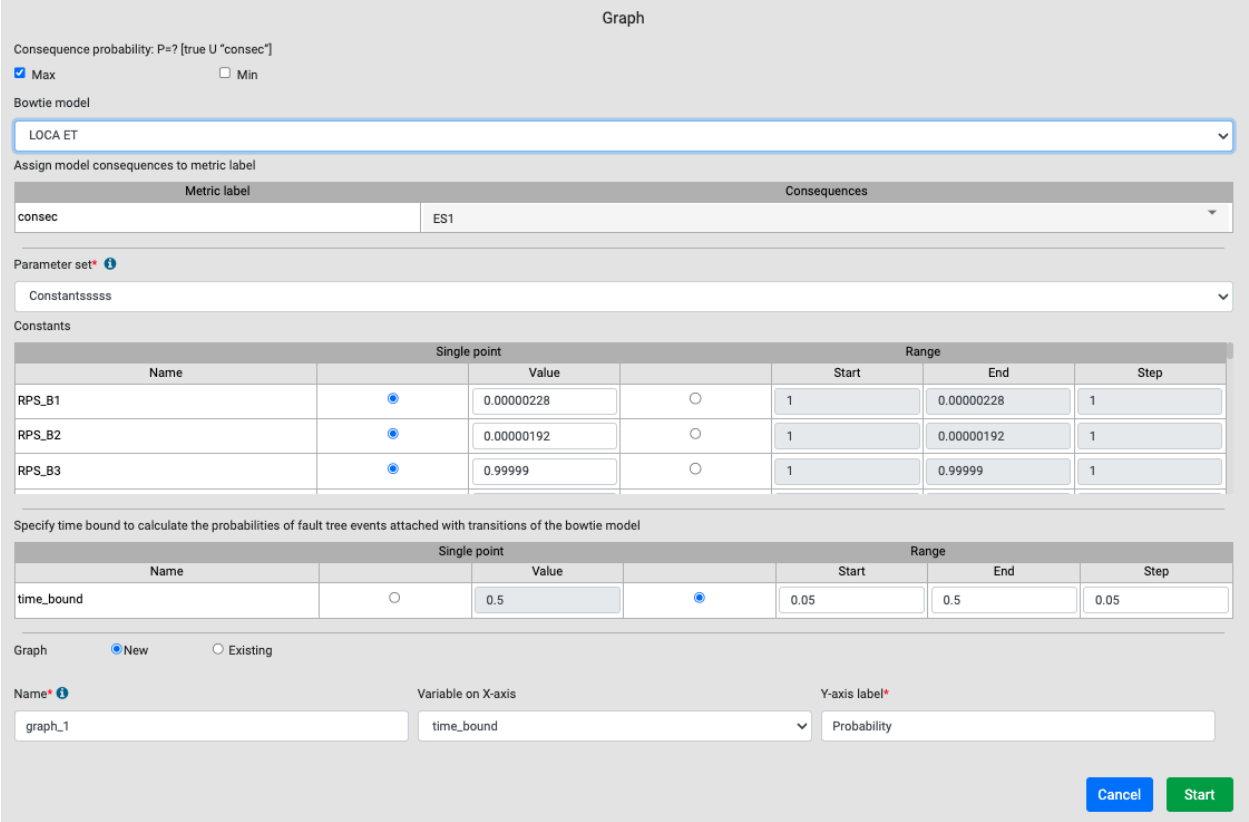
Close

- Click the “Download” link to download the generated artifact (DFT/DRN).
- Click the “Open in new tab” link to open the generated artifact in a new browser tab.

- Click the “Load” link to load the generated DFT as a failure model in the current project.
- Click the icon to download the results in the selected rows in a csv format.

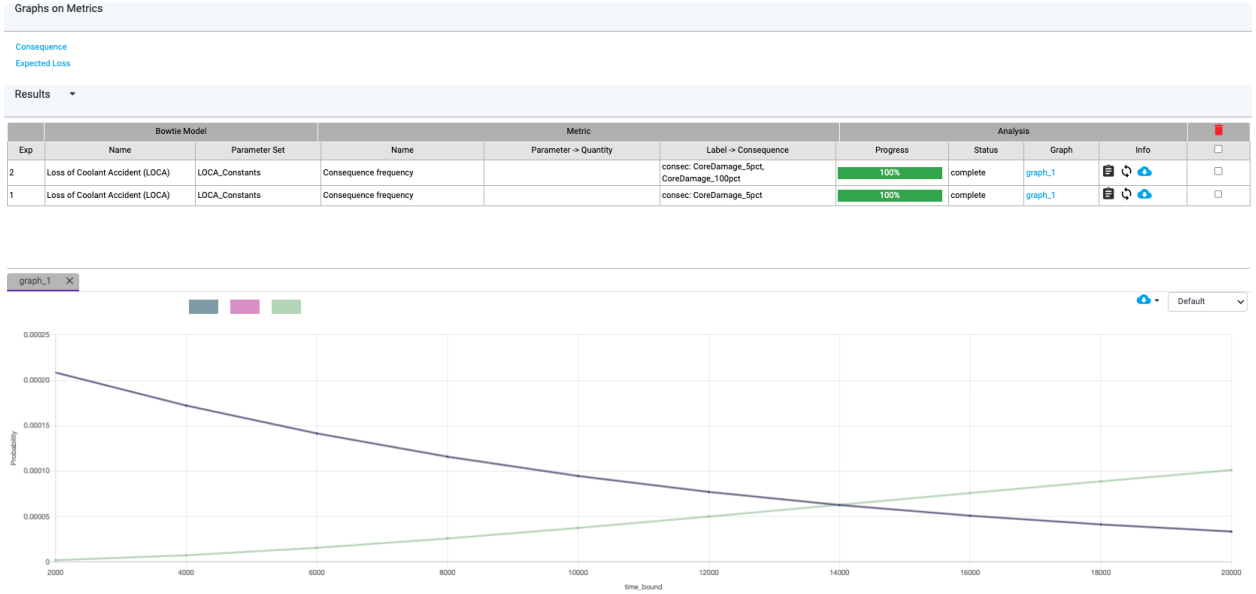
Graphs

We provide an interface to plot consequence probabilities/frequencies or expected values of loss quantities against different parameters of interest. Click on the “Graphs” link under “Computing” in the left panel. Click the “Consequence/Expected Loss”, and the following window will appear:



- Select whether the min, max, or both values of consequences probabilities/frequencies (expected loss) are to be computed.
- “Bowtie model” dropdown: a model that is selected as a default model in the “Bowtie Models” page is automatically selected.
- “Assign mode consequences (quantities) to metric labels (parameter)”: Select (multiple) consequence(s) (quantities) from the dropdown for which we want to measure probabilities/frequencies (expected values).
- A parameter set which is attached with the selected bowtie model (above) is automatically selected. It can be changed at this point to generate another variant of the model.

- One can specify a range of values of time-bound as well as of constants defined in the selected parameter set.
- Select whether to draw a new graph or it is to be plotted on an existing graph that has the same variable on X-axis.
- The variable on the X-axis of the graph can be either time-bound or from the constraints of the selected parameter set.
- Click the “Start” button to display the graph:



- Click the ■ icon to stop the running analysis.
- Click the ⟳ icon to rerun the analysis and draw a graph.

5. Annotation of SysML Models with Safety Information

In order to annotate SysML model elements with safety information, we have created a few packages, which are to be used inside the SysML models against which fault trees are to be generated. These packages are:

- DGBMetadata: It contains a package DFTElements with the following sub-packages and elements:

- DFTGates package: It defines all gates that are used to construct fault trees.

```
package DFTGates {

    metadata def AND;
    metadata def OR;
    metadata def VOT {
        /* if any k-out-of-n components fail (input events),
        the system will fail (output event) n number of input events */
        attribute k : Number;
    }
    metadata def SPARE;
    metadata def PAND;
    metadata def POR;
    metadata def FDEP {
        /* The failure of the trigger_element renders the children of FDEP failed
        as per the value of probability attribute. The default value of probability
        is 1. */
        occurrence trigger_element;
        attribute probability: Real;
    }
    metadata def FSEQ;
    metadata def MUTEX;
}
```

- DFTBEs package: It defines all basic elements that may be used in fault trees.

```
package DFTBEs{
    abstract metadata def BE{
        /* The value of dormancy attribute is only relevant if the BE is a
        child of a spare spare gate. The default value of dormancy is 1. */
        attribute dormancy:Real;
    }
    abstract metadata def BE_CONSTANT_DISTRIBUTION specializes BE {
        attribute prob:Real;
    }
    abstract metadata def BE_EXPONENTIAL_DISTRIBUTION specializes BE {
        attribute rate:Real;
    }
    abstract metadata def BE_ERLANG_DISTRIBUTION specializes BE {
        attribute rate:Real;
        attribute phases:Real;
    }
    abstract metadata def BE_NORMAL_DISTRIBUTION specializes BE {
        attribute mean:Real;
        attribute stddev:Real;
    }
    abstract metadata def BE_WEIBULL_DISTRIBUTION specializes BE {
        attribute rate:Real;
        attribute shape:Real;
    }
}
```

    - TOP_LEVEL metadata: It is used to annotate an element of a fault tree as a top-level element. More than one element can be annotated as top-level elements. This helps generate multiple fault trees (for different scenarios) collectively that may share Gates and BEs.
- FailureModes: It defines all failure modes that may be used to annotate elements of SysML models with safety information. At the moment we allow failure modes to be modeled with the following failure distributions:
    - Exponential distribution
    - Erlang distribution
    - Weibull distribution
    - Log-normal distribution, and
    - Constant distribution

  Moreover, within this package, we allow to define model constants as (DFTParameters) enumerations. These constants can be used to define failure rates, probabilities, shapes, etc. of failure modes.

```
package FailureModes {

    import DGBMetadata::DFTElements::*;
    /* DFTParameters enumiration defines constants used to annotate
    failure rates/probabilites/shares/etc. of BEs in fault trees. */
    enum def DFTParameters :> Real {
        FIT1 = 0.000000001;
        FIT2 = 0.000000002;
        FIT3 = 0.000000003;
        FIT4 = 0.000000004;
        prob = 0.2;
        param1 = 10.2;
        param2 = 1.1;

    }

    metadata def FIT1 specializes BE_EXPONENTIAL_DISTRIBUTION{
            attribute redefines rate = DFTParameters::FIT1;
    }
    metadata def FIT2 specializes BE_EXPONENTIAL_DISTRIBUTION{
            attribute redefines rate = DFTParameters::FIT2;
    }
    metadata def FM1 specializes BE_CONSTANT_DISTRIBUTION{
            attribute redefines prob = DFTParameters::prob;
    }
    metadata def FIT3 specializes BE_ERLANG_DISTRIBUTION{
            attribute redefines rate = DFTParameters::FIT3;
            attribute redefines phases = 2;
    }
    metadata def FM_4 specializes BE_NORMAL_DISTRIBUTION{
            attribute redefines mean = DFTParameters::param1;
            attribute redefines stddev = DFTParameters::param2;
    }
    metadata def FIT4 specializes BE_WEIBULL_DISTRIBUTION{
            attribute redefines rate = DFTParameters::FIT4;
            attribute redefines shape = 3;
     }
}
```

Laptop Example.

The following example explains how elements within the SysML model can be annotated to generate fault trees out of them.

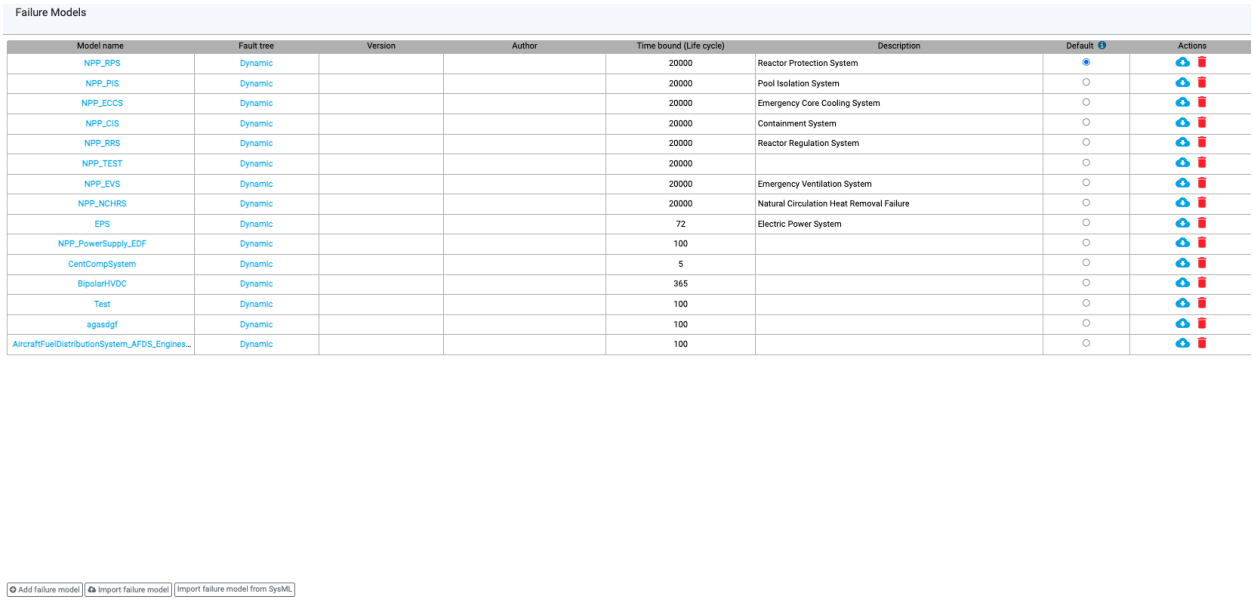
```
package LaptopPackage {
    import FailureModes::*;
    part Laptop {
        part CPU1 {
            metadata Failure:FIT2;
        }
        part CPU2 {
            metadata Failure:FIT1;
        }
        part cooling {
            metadata Failure:FIT1;
        }
        part plug {
            metadata Failure:FIT2;
        }
        part battery {
            metadata Failure:FIT2;
        }
        part switch {
            metadata HWF:FIT1;
        }
        metadata power:SPARE about plug::Failure, battery::Failure;
        metadata processor:AND about CPU1::Failure, CPU2::Failure;
        metadata laptop:OR about power, processor;
        metadata Dep:FDEP about CPU1::Failure, CPU2::Failure {
            trigger_element = cooling::Failure;
        }
        metadata TLE1:TOP_LEVEL about laptop;
        metadata TLE2:TOP_LEVEL about power;
    }

}
```

After the compilation of the SysML model annotated with safety information using our packages in Jupyter Notebook, run the following command to export the package in JSON format. **Currently, we support the latest version – v0.33.0 – of SysML 2.0.**

%export <package_name>

After downloading, it can be uploaded inside the SAFEST tool at the “Failure Models” page under “Fault Tree Analysis” in the left panel:



Failure Models

Model name	Fault tree	Version	Author	Time bound (Life cycle)	Description	Default	Actions
NPP_RPS	Dynamic			20000	Reactor Protection System	◉	
NPP_PIS	Dynamic			20000	Pool Isolation System	○	
NPP_ECCS	Dynamic			20000	Emergency Core Cooling System	○	
NPP_CIS	Dynamic			20000	Containment System	○	
NPP_RRS	Dynamic			20000	Reactor Regulation System	○	
NPP_TEST	Dynamic			20000		○	
NPP_EVS	Dynamic			20000	Emergency Ventilation System	○	
NPP_NCHRS	Dynamic			20000	Natural Circulation Heat Removal Failure	○	
EPS	Dynamic			72	Electric Power System	○	
NPP_PowerSupply_EDF	Dynamic			100		○	
CentCompSystem	Dynamic			5		○	
BipolarHVDC	Dynamic			365		○	
Test	Dynamic			100		○	
agasdgf	Dynamic			100		○	
AircraftFuelDistributionSystem_AFDS_Engines...	Dynamic			100		○	

Add failure model | Import failure model | Import failure model from SysML

Click the “Import failure model from SysML” button to extract failure models as well as parameter sets from the SysML model. The failure models are added to the list of existing failure models on the “Failure Models” page. Whereas all constants (DFTParameters enumerations inside the SysML FailureModes package) are added to the existing parameter sets on the “Parameter Sets” page.

6. Grammars

Regular Expressions of Identifiers and Numeric Constants

- Identifier (id):
  - It is used to give a unique name to e.g. constants, expressions, etc.
  - It is a string of characters starting with a capital letter (A-Z), a small letter(a-z), or an underscore (_) followed by a capital letter(s), a small letter(s), underscore or digit(s) (0-9).
  - Note that identifiers cannot be from the list of keywords of a grammar.
- Numeric constant (nc):
  - Simple,decimals and exponential i.e 123, 123.123, 123e+1, 123e-1, 123e1, 123.123e+1, 123.123e-1, 123.123e1, 123.123E1, 0.12

Context Free Grammar (CFG) of Real Expressions

- RE → E | + nc | - nc
- E → E OP E | nc | id | (RE) | pow(RE,RE) | log(RE,RE)

- OP → + | - | * | /

Where

- "id" is an identifier of an expression,
- "nc" is a numeric constant string, and
- pow, log are keywords of the grammar.

Context Free Grammar (CFG) of Boolean Expressions

- E → E OP E | id | (E) | !id | !(E) | system_failed | failed
- OP → |
- OP → &

Where

- "id" is an identifier of an expression, and
- failed, system_failed are keywords of the grammar.

Context Free Grammar (CFG) of Continuous Stochastic Logic (CSL)

- PROP → P OP2 Type [PathFormula] | T OP2 Type [RewardFormula] | LongRun OP2 Type [StateFormula]
- Type → =? | OP3 E
- LongRun → LRA | S
- PathFormula → OP4 BoundedExpression StateFormula | StateFormula OP5 BoundedExpression StateFormula
- BoundedExpression → ^ { Bound } | { Bound } | Bound | NULL
- Bound → [E,E] | OP3 TIME
- TIME → (E) | nc
- RewardFormula → I = E | C <= E | F StateFormula | LongRun
- StateFormula -> StateFormula OP6 StateFormula | P OP2 OP3 E [PathFormula] | LongRun OP2 OP3 E [StateFormula] | id | system_failed | failed | (StateFormula) | true | !StateFormula
- OP → => | & | | | = | != | <= | >= | > | < | + | - | * | / | %
- OP1 → + | -
- OP2 → min | max | NULL
- OP3 → <= | >= | > | <
- OP4 → G | F
- OP5 → U | W | R
- OP6 → | | &
- E → E OP E | id | nc | (E) | !(E) | (OP1 nc)
- NULL → empty string

Where

- "id" is an identifier of an expression,
- "nc" is a numeric constant string, and
- true, false, Pmin, Pmax, Smin, Smax, Tmin, Tmax, LRAmin, LRAmax, P, R, T, S, LRA, min, max, G, U, F, W, C, I, failed, system_failed are keywords of the grammar.