Advances in Probabilistic Risk Assessment (PRA): A Look into a Practitioners Toolbox

J. Mok, S. Kaasalainen, K. Donnelly
Risk Management Directorate
Nuclear Safety Solutions Ltd.
Toronto, Ontario, Canada, M5G 1X6

ABSTRACT

The ever-increasing emphasis on the use of Probabilistic Risk Assessment (PRA) in risk-informed decision making translates into increased expectations relating to PRA applications for the groups tasked with developing and maintaining the facility PRAs. In order to succeed in meeting the demand for PRA work, it is essential to develop methodologies and tools (or utilities) that improve the efficiency with which the PRAs are processed and manipulated to obtain a solution. Examples from the Nuclear Safety Solutions (NSS) PRA Practitioners tool box include utilities for cutting logical loops, optimizing fault trees (to decrease run-times), modularizing fault trees, and converting event trees into high level fault tree logic (an important element if the PRA study is to be used to support a risk monitor such as an Equipment Out-of-Service (EOOS) Monitor). The objective of this paper is be to briefly describe the main features of these utilities, and to illustrate the value they have in terms of improving the efficiency and effectiveness of PRA development and maintenance at NSS.

INTRODUCTION

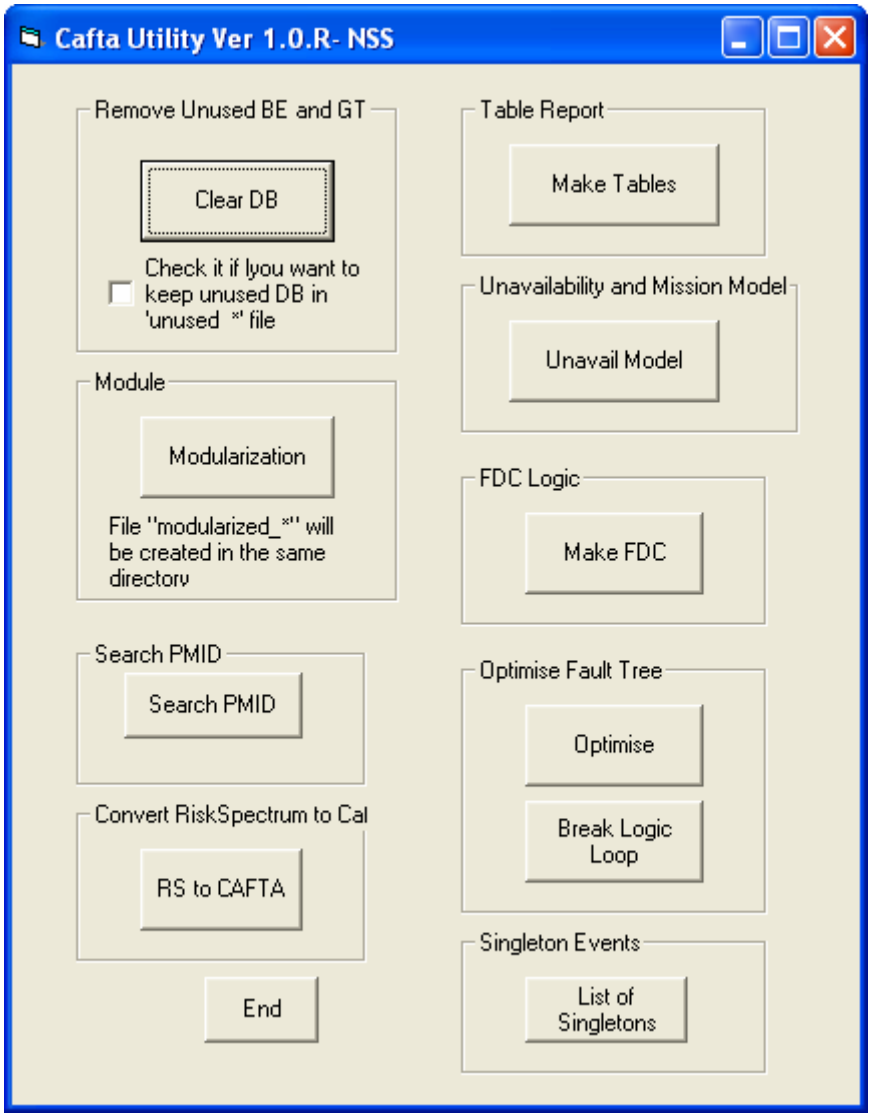
This paper reviews recent advancements in the use of utilities and tools to help optimize the development and maintenance of the Probabilistic Risk Assessments (PRAs) for the operating multi-unit CANDU power plants in Ontario.

A 'CAFTA Utilities' (Ver. 1.0.R) program was developed during the course of several Probabilistic Risk Assessment projects undertaken by NSS between 2002 and 2006. A total of 8 scripts constitute the 'CAFTA Utilities' suite, each of which is accessible through the *CAFTA_Util.exe* program. The Windows graphical user interface for this program is shown in Figure 1.

The CAFTA_Util program have been developed using the CAFTA API functions described in Reference 1 and Visual Basic. The API functions in CAFTA allow communication with various elements of the CAFTA software and databases and outside programs.

The functionality of each script, represented by each of the buttons in Figure 1, is the subject of the remainder of this paper.

Figure 1: NSS CAFTA Utilities Main User Interface



CLEAR DB

Several revisions of a fault tree are usually created before the final version is achieved. In the process many primary events and/or gates are created in the fault tree and then subsequently deleted. Deleting an event from a fault tree does not, however, clear this event from its associated database. As a result, the database corresponding to the final version of the fault tree may contain a large number of events/gates that are not utilized. It is thus desirable to have these unused events/gates cleared from the database. Removing all the unused events/gates from the database manually would be a tedious and very time consuming task; it is thus useful to have this task automated by means of the simple script outlined below.

- Algorithm and Program Flow

  - Open the CAFTA fault tree file.

  - Obtain the following information from the fault tree file: number of top events and their labels, number of gates and their labels, number of primary events and their labels, and the name of the associated database file. Open the database file.

  - One by one, read the labels of all the basic events from the basic event table in the database. Search for the corresponding labels in the fault tree file. If an event is not contained in the fault tree, delete it from the BE table in the database.

  - One by one, read the gate labels in the gate (GT) table in the database. Search for the corresponding labels in the fault tree file. If the gate is not contained in the fault tree, delete it from the GT table in the database.

  - Display the number of basic events and gates that were removed from the database.

MODULARIZATION

Modularization is a process of collapsing strings of OR gates involving primary events that are not replicated in a particular fault tree. In the past, the process of modularizing a fault tree was performed to reduce its size before the calculation of minimal cutsets. As such, modularization served the purpose of saving fault tree processing time and reducing the number of minimal cutsets to be reviewed.

The process of fault tree modularization can be time consuming and thus it is necessary to have this process automated in the form of an executable script.

- Algorithm and Program Flow

  - Open the CAFTA file to be modularized.
  - Search the CAFTA file for the name of the associated database.
  - Find the total number of top events.
  - For each top event:

    Modularize the fault tree (i.e., collapse the strings of OR gates (Reference 2)),

    Calculate probabilities for individual modules.

  - Update the database – i.e., insert the names of the modularized events.

  - Perform a fault tree reduction step where unused intermediate gates are removed.

  Note, throughout this process, the following events are excluded:

- events that are used more than once
- events characterized by failure modes MO and TO (i.e., scheduled or test outage events)
- all human interaction events
- developed (i.e., interfacing) events, and
- initiating events.

MAKE TABLE

In order to present the information contained in a CAFTA fault tree and the associated cutset file in a more compact and organized format customized to the client needs, it is convenient to partition the data into separate tables. Examples include the following:

- Interfacing Events – list of all developed events utilized within the fault tree.
- Module Listing – list of all the modules along with the constituting primary events (and their probabilities).
- Primary Event Data – list of all the primary events along with their associated data such as failure probability, test interval, test procedure or predefined number, etc.
- Simple Human Interactions – list of all simple human interaction events along with the relevant information such as failure mode group number, location, manipulation time etc.
- Complex Human Interactions – list of all complex human interaction events and the associated information such as task type, quality of indication and availability time.
- System Reliability Analysis – list of the top cutsets (number of which is specified by the user), for each of the top events within the fault tree, along with the corresponding event descriptions.
- Importance Indices – list of importance indices for each of the top events, where human interaction and initiating events are listed separately from all the other primary events.

Since manual generation of these tables would be extremely time consuming, a script is necessary to automatize this process. The tables generated using this tool are in Microsift Word format.

UNAVAILABILITY MODELS

Once a risk model fault tree is developed, it is sometimes also necessary to create a corresponding unavailability model—where the mission duration events are absent. The process of manually selecting and deleting all of the ‘mission’ events from a risk model fault tree can be time consuming and may also be prone to human error. Thus, it is useful to develop a tool to automate this process.

- Algorithm and Program Flow

  - Open the input file; read in the name of the risk model CAFTA fault tree and the list of all events to be deleted in addition to all the standard 'mission' model events.
  - Search the CAFTA file for the name of the associated database.
  - Read in the number of top events and the corresponding labels.
  - Create a file for writing output (i.e., this file is to have a prefix 'unavail_').
  - For each top event:

    - Search through the equations—which specify the calculation of failure probability—used by all the primary events and delete those events that utilize equation type '3' – i.e., that characterizing a mission event: 1-exp(lambda*t).

    - Search through all the primary event symbols and delete all initiating events (symbol='I'); set the probability of all developed events (symbol='D') to 0.0.

    - Delete all the events that might have been specified in the input file

    - Restructure the fault tree where necessary – for example, if an event under an AND gate was deleted, then the parent gate must also be deleted.

MAKE FDC

Given a set of event trees modelling the sequences of events following an initiating event and leading to various levels of fuel damage, an approach to solving the event sequences in terms of minimal cutsets is to develop an equivalent set of fault trees, each corresponding to a single level of fuel damage. Development of such a fault tree allows for the direct calculation of frequency of fuel damage corresponding to each category. This approach enables the development of a single top fault tree for use in applications, such as risk monitors (this enables a simple method to re-generate of cutsets within the risk monitor framework).

The 'Make FDC' script was developed to automate this task.

- Algorithm and Program Flow

An input file containing a list of end states and a list of all event trees is required.

  - For each event tree file listed in the input file:

    - Open the ET file by using CAFTA API function, ETOpen().
    - Get the total number of sequences in the ET file by using CAFTA API function, ETGetNumSequences().

  - Take the first sequence by using ETGetFirstSequence() and check if there is an end state for that sequence by using ETGetSequenceClass().
  - If there is no end state assigned move to the next sequence by using ETGetNextSequence().
  - If there is an end state existing for that sequence then store the end state name and the associated path sequence.
  - Repeat the above steps until all the sequences in the opened event tree have been processed.
  - Repeat the above steps for all the ETs in the input file.

- Sort the stored end states (with the corresponding sequences).
- For each end state, convert the stored logic into a CAFTA fault tree format by using CAFTA API functions (e.g., FTOpen(), FTGetTopIndex(), FTAddGateLogic() and so on).

The CAFTA API functions are described in Reference 1.

OPTIMIZATION

Fault tree based risk assessment for large facilities, such as nuclear generating stations, is performed in several stages. First, one identifies all the relevant systems within the facility along with all the possible interfaces between them. A fault tree model for each specific system is then developed separately – this task, being the most time consuming, is often spread among different analysts. Finally, all the fault trees are merged into a single fault tree, which is then used for evaluation of the minimal cutsets and their corresponding failure probabilities.

Generation of minimal cutsets for the (final) merged fault tree, often consisting of an excess of 30,000 basic events, can take several days or longer. More precisely, the length of time required for fault tree evaluation is dependent primarily on four factors: computing power (i.e., CPU speed and the available memory), truncation probability, complexity of the fault tree logic, and finally, efficiency of the cutset generator.

While modeling systems using fault tree logic, it is important to keep the structure of each fault tree coherent enough so that it can be subsequently reviewed, debugged or reused fairly readily by other analysts. A common technique for fault tree model development is referred to as the “rule of immediate cause”. Maintaining the fault tree structure derived from this approach, however, often means partitioning various events (by introducing more gates) and thus increasing the complexity of the overall fault tree. This in turn, makes the evaluation of the merged fault tree more complex and thus time consuming (i.e., not optimal for solving). The aim of the optimisation tool is to simplify the fault tree structure while preserving the intended logic (and hence all the cutsets of the original fault tree) and significantly decrease the computing time of minimal cutsets. The algorithm for simplifying the fault tree is described in detail in Reference 3. In

addition, new algorithms aimed specifically at reducing computing time are under development.

BREAK LOGIC LOOP

In PRA analysis, when the fault trees of systems are merged, logical loops are usually generated due to the mutual dependencies, especially among the support systems such as service water system, instrument air system and electric power system, etc. A fault tree with the logical loops cannot be solved due to the existence of an unlimited recursive loop. Typically, the logical loop is solved by breaking the logical loops at the points where the dependencies among the support system are relatively weak and developing new fault tree without the logic loops. However, this conventional method has some drawbacks such as:

1) It is time consuming and difficult to review the relations among the support systems and to develop new fault tree without logic loops.

2) Some minimal cutsets may be missed due to the arbitrary nature of breaking dependencies among the support systems.

To solve these problems the Break Logic Loop tool has been developed. Using a top-down approach, the logic of the fault tree is expanded. During expansion, the logical loops are identified based on given criteria and are deleted automatically. Using this approach, new logic for the fault trees is generated without the logical loops. The algorithm used is described in detail in Reference 4.

The conventional (manual approach) and developed (automated) methods are the same in that both methods delete some parts of fault tree logic in order to break the logical loops. However, the developed method provides the exact criteria, which is based on the Boolean equations, to identify parts of the fault trees that cause the logical loops. Such parts are deleted automatically during the expansion of fault tree logic. Therefore, this method enables savings in terms of man-power to review the dependencies among the support systems and to develop the new logic of fault trees without the logical loops. It also ensures complete results without missing minimal cutsets.

SINGLETON EVENTS

Singleton events resulting from fault tree analysis is a result that is of interest to system engineers and operators as it provides insights as to where improvements to system reliability could be achieved. A script has been developed to identify all the components which, when failing individually, lead to a system to be unavailable and make a list of such singleton events on a system basis.

- Algorithm and Program Flow

  - Make a list of all basic events in a system fault tree.

- Set a basic event to TRUE and check whether the top event is TRUE.
  - If the top event is TRUE, add this basic event to the list of singletons.
- Unset the basic event, to return it normal status.
- Repeat steps above for all basic events in the system fault tree.

SUMMARY

With ever increasing demands to develop, maintain and use nuclear power plant PRAs, it has been necessary in recent years to consider ways to improve the efficiency as well as accuracy with which PRA practioners perform various tasks. Several examples of utilities developed at NSS while under-taking various PRA projects have been discussed. These utilities have enabled NSS to meet ever increasing customer commitments and expectations. It is anticipated that the items discussed in this paper are only a starting point, and as reliance on PRA insights continues to increase in the Canadian nuclear industry, there will no doubt be a continued need to develop additional tools and discover new and innovative ways for dealing with new problems as they arise.

REFERENCES

1. Risk and Reliability WorkStation Applications Program Interface (API) function manual Ver 2.2 prepared By Science Applications International Corporation.

2. Han SH, Kim TW, Yoo KJ, Development of an integrated fault tree analysis computer code MODULE by modularization technique. Reliability Engineering System Safety 1988;21:145-54.

3. Niemela I. “On simplification of large fault tree”. Reliability Engineering System Safety 1994;44:135-8.

4. Yang JE, Han SH, Park JH, and Jin YH, “Analytic method to break logical loops automatically in PSA”, Reliability Engineering and System Safety, 56, page 101-105, 1997.