



SERCO ASSURANCE

An Introduction and Tutorial Guide to Using the SOLOMON Event Tree Code

Brian Holmes

November 1999



SERCO ASSURANCE



SERCO ASSURANCE

An Introduction and Tutorial Guide to Using the SOLOMON Event Tree Code

Brian Holmes

November 1999

The information which this report contains is accurate to the best of the knowledge and belief of Serco, but neither Serco nor any person acting on behalf of Serco make any warranty or representation expressed or implied with respect to the use of the computer program described, nor assume any liabilities with respect to the use of, or with respect to any damages which may result from the use of information, method or data disclosed in this report.

	Name	Signature	Date
Author	Brian Holmes		
Reviewed by	Chris Maidment		
Approved by	Colin Cooper		

Executive Summary

The SOLOMON computer program has been developed by Serco Assurance experts to construct and evaluate large Event Trees. The code was originally developed for safety-critical applications, such as a Level 2 Probabilistic Safety Analysis of a Nuclear Power Plant, but its inherent flexibility makes it ideal for a wide range of applications. Because the number of branches in an event tree increases exponentially with the number of top events, or decision points, special features are required to handle all but the simplest of event trees. The special features included in SOLOMON that make it suitable for handling the largest of event trees include:

1. Branching probabilities are expressed algebraically, one per decision point instead of one per branch. This greatly reduces the amount of input data required as the number of branches increases;
2. Decision points, or nodes, can be grouped into 'supernodes' making the display of an event tree much simpler.

This Guide is provided with an evaluation version of SOLOMON to provide an introduction to the main features and use of the code. It includes a series of worked examples illustrating the construction of a simple model, as well as a brief description of a generic plant model provided as an example of a 'real world' application.

SOLOMON has been designed to run on an IBM compatible PC running under the Windows 9x (or later) operating system (a 16-bit version is available for systems running under Windows 3.x). It includes a Graphical User Interface to assist the user in constructing event trees and analysing the results of SOLOMON calculations. The code uses a combination of key words and C programming language to construct an Event Tree model.

Contents

1	Introduction	1
2	Key Features	1
2.1	PHYSICAL MODEL INTEGRATION	2
2.2	PRIOR PATH DEPENDENCY	2
2.3	MULTI-BRANCHING NODES	2
2.4	PRE-DEFINED END CATEGORIES	2
2.5	SUPERNODE TREES	2
2.6	UNCERTAINTY AND SENSITIVITY ANALYSIS	3
3	Worked Examples	3
3.1	EXAMPLE 1	3
3.2	EXAMPLE 2	10
3.3	EXAMPLE 3	14
3.4	EXAMPLE 4	19
3.5	EXAMPLE 5	20
3.6	EXAMPLE 6	24
4	Representative Plant Model	26
5	Supplied files	27
6	References	29

1 Introduction

The SOLOMON Event Tree code was originally developed for safety-critical applications, such as a Level 2 Probabilistic Safety Analysis (PSA) of a Nuclear Power Plant (NPP), but its inherent flexibility makes it ideal for a wide range of applications. Event trees are used throughout the PSA process, and many programs have been developed for their construction and evaluation. However, their use in a Level 2 PSA presents particular problems because of the size of the trees required.

The complexity and size of an event tree increases exponentially with the number of top events. For a Level 2 PSA of a NPP, there is a trend towards using event trees with many tens of top events to describe the progression of an accident sequence. For a typical case this may result in 2^{30} or more paths.

SOLOMON has been designed to enhance productivity in Probabilistic Safety Assessments and provide a cost-effective means of risk evaluation using the Event Tree formalism. The key features and benefits of the code are summarised in Section 2. A tutorial is provided in Section 3, beginning with a basic event tree and progressing in easy stages to include various features and illustrate key phases in using SOLOMON to develop an event tree. Section 4 describes a more realistic example of a generic nuclear plant model, which is the basis for the evaluation version of the code. Section 5 lists the files provided with this installation, which include the input and output for all examples described here, and describes how to access these files.

For a complete explanation of the syntax of the full SOLOMON command set, please see the SOLOMON User Guide (Roberts 1999).

2 Key Features

SOLOMON employs a system of nodes, or control points, to represent top events with a Graphical User Interface using a colour coded representation of the model to aid the user in navigating the event tree. Intuitive dialogs are available to guide the user in constructing a CET, allowing rapid development of complex event trees using a powerful combination of text editing and hierarchical dialogs.

SOLOMON offers a unique combination of features, including:

- system models integrated into the tree
- prior path dependency
- multi-branching nodes
- pre-defined end categories
- supernodes to combine the results of a number of nodes for display purposes
- uncertainty and sensitivity analysis using Latin Hypercube Sampling techniques

- fast execution times

2.1 SYSTEM MODEL INTEGRATION

Using a powerful combination of dedicated keywords, coupled with the functionality of the C programming language for the advanced user, SOLOMON enables system models to be constructed and integrated into the event tree. Simple models can be rapidly developed to represent any process that may impact on the probability of an event, such as physical phenomena, economic factors etc. As understanding of the problem under investigation grows, these may be developed further, introducing more mechanistic representations and providing the capability for sophisticated decision making within the tree. The implications of more detailed modelling on the final results are readily available for scrutiny, thus providing an auditable route for Quality Assurance purposes.

2.2 PRIOR PATH DEPENDENCY

Prior path dependency is another key feature of SOLOMON that allows the user to take account of decisions made earlier in the tree, such as systems availability, when calculating parameters for input to a model or the probability of a node. The selection criteria can include the initial state of the system being analysed, enabling a range of scenarios with similar features to be modelled within a single input dataset, thus reducing development and calculation turn-around times.

2.3 MULTI-BRANCHING NODES

Branching points represent decisions in the Event Tree. SOLOMON has the capability of representing binary or multi-branch nodes. Thus situations where one, two or more systems fail to operate, or where failures may be minor or catastrophic, can be handled at a single decision point.

2.4 PRE-DEFINED END CATEGORIES

A large Event tree, typical of a Level 2 PSA of a NPP, may result in 2^{30} or more paths. In such cases, organising the results into an ordered and easily understandable form could become very time consuming. SOLOMON addresses this problem by allowing the user to define a series of end categories, which group the path end-states into a logical combination of key events and phenomena.

2.5 SUPERNODE TREES

The supernode concept developed for SOLOMON presents the user with an intuitive method of combining nodes to collapse a large Event Tree down to a manageable size for convenient display and importing to other applications such as a word processor. Several simplified event trees can be constructed to focus on key aspects of the calculation, starting either at the beginning, or part way through an event tree.

2.6 UNCERTAINTY AND SENSITIVITY ANALYSIS

Often, a lack of detailed knowledge of the phenomena and processes governing the behaviour of a system can give rise to significant uncertainties in a PSA. SOLOMON automatically ranks phenomena in order of importance, allowing the user to apply uncertainty distributions to the critical parameters. The resulting uncertainty distribution for each end category can then be displayed in the Probability and Cumulative Density Function viewer, or exported to the users preferred spreadsheet application for more sophisticated data handling.

3 Worked Examples

The following sections provide a number of worked examples for the user to try. Although simple in concept, they build progressively to familiarise the new user with many of the main features of the SOLOMON code. Input and output datasets for these examples are provided with the SOLOMON installation.¹

3.1 EXAMPLE 1

In this example we will create a simple event tree with four 'nodes' (also referred to as 'events', 'questions' or 'top event questions' elsewhere in PSA). These nodes are called 'A', 'B', 'C' and 'D' for simplicity, although SOLOMON allows names of up to 32 characters, if required.

Each represents a question that can be TRUE or FALSE. The probability of each node being TRUE is as follows:

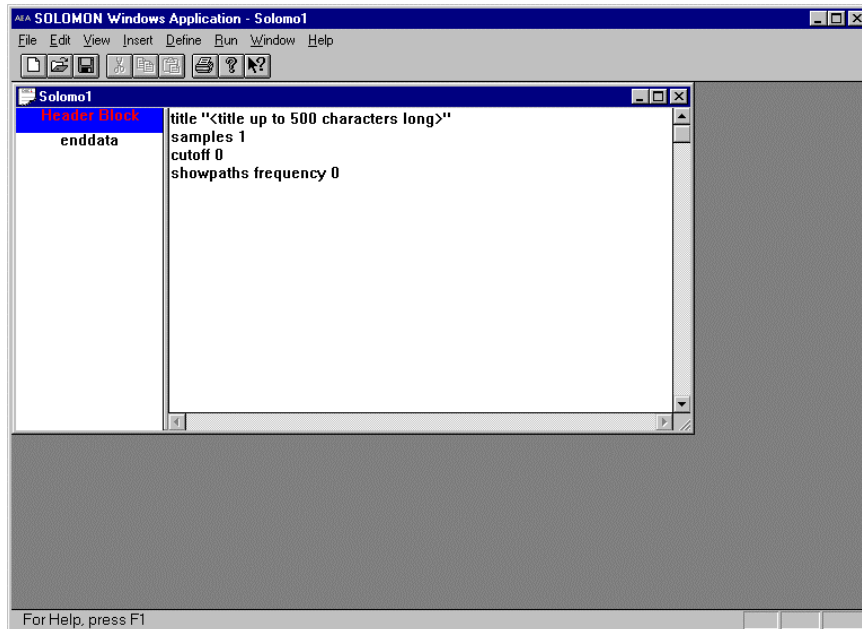
NODE	PROBABILITY OF BEING TRUE
A	0.1
B	0.2
C	0.3
D	0.4

Therefore the probability of all four nodes being simultaneously TRUE is $0.1 \times 0.2 \times 0.3 \times 0.4 = 2.4 \times 10^{-3}$. The probability of being simultaneously FALSE is $0.9 \times 0.8 \times 0.7 \times 0.6 = 3.024 \times 10^{-1}$.

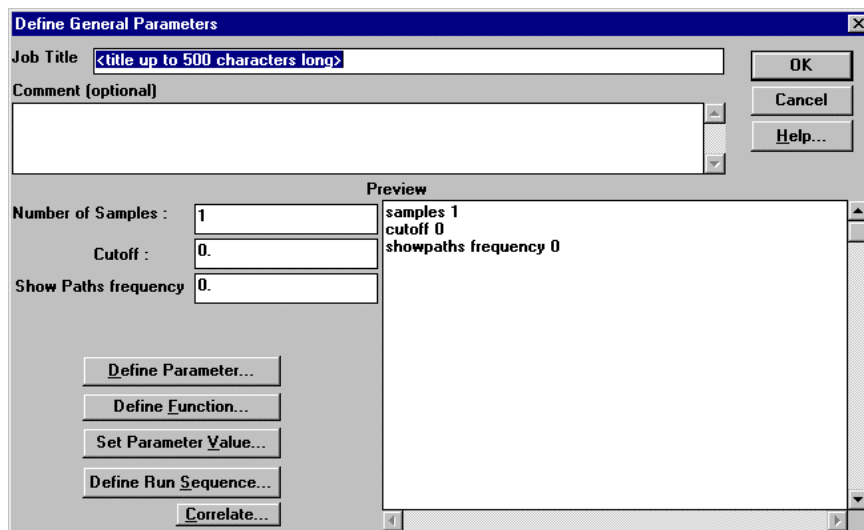
SOLOMON employs a Graphical User Interface (GUI) to guide the user in constructing an event tree. The input files are plain ASCII text, and so the advanced user may wish to skip the

¹ Note that the evaluation model has reduced functionality and when using this tutorial you will be able to create, edit and save input files. These may be verified by comparing with the supplied input files, but the calculation engine has been disabled so that only the pre-prepared output can be examined with the viewers.

GUI, and use a text editor. To begin constructing a new input file, first start SOLOMON (for details on how to start SOLOMON for your particular operating system, please refer to the booklet provided with your installation disc). Dismiss the 'About SOLOMON' window, select 'File' then 'New' from the menu. This will bring up the main window with the default name of 'solomo1', as shown below.

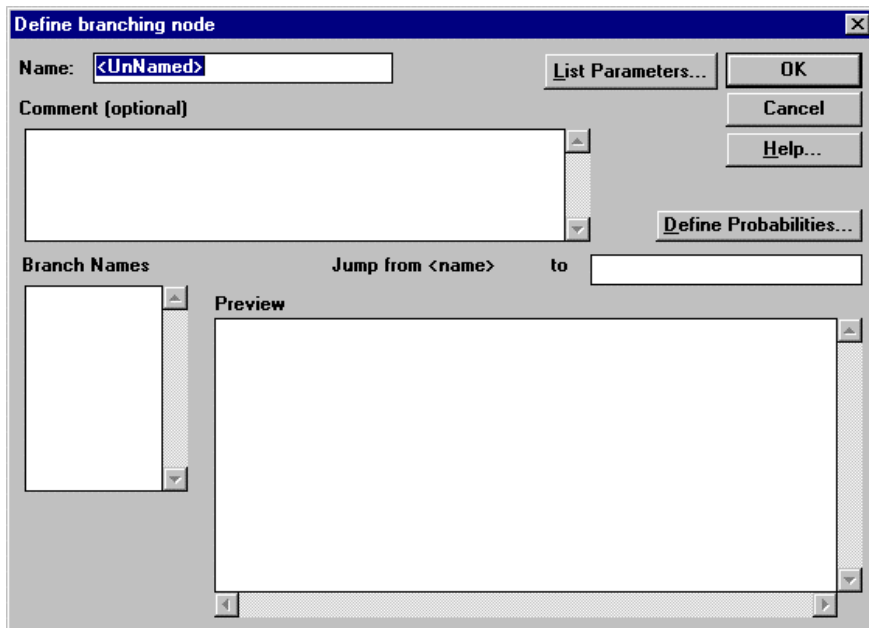


Double click on the blue 'Header Block' box in the left hand side of this window and you will see the dialog for input to the header block, as shown below.

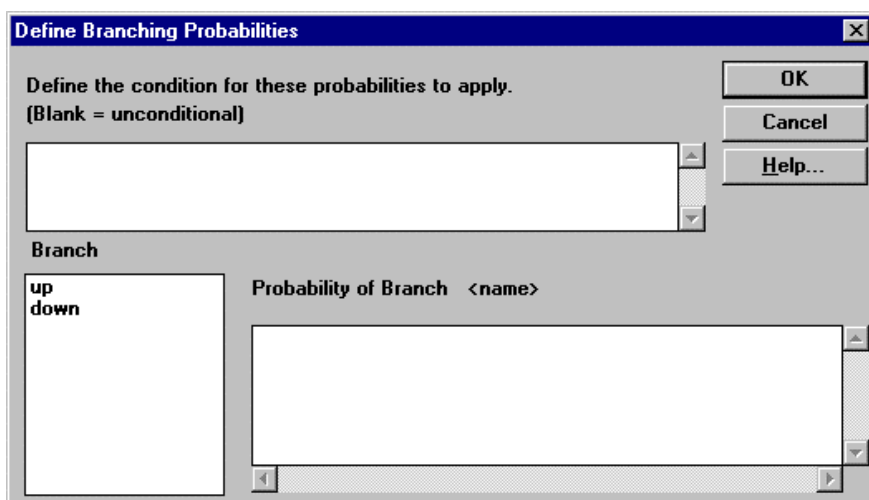


Enter a job title (such as 'Example 1') in the highlighted area and then click on 'OK'.

Now click on 'enddata' in the left hand window in preparation for inserting a node. *SOLOMON always inserts before the current position.* Go to the 'Insert' menu and select 'Branching Node...'. The resulting dialog will look like this:



SOLOMON allows node names up to 32 characters, which can be descriptive of the node, but for this simple case, we will call this node A by typing 'A' (without the quotes)² in the highlighted box. We will assume that this is a simple binary node, and assign branch names as 'up' and 'down' in the 'Branch Names' box. Click in the 'Branch Names' box and type 'up', press Enter and type 'down'. We now need to define the node probability by clicking on the 'Define Probabilities...' button. This will bring up the following dialog:



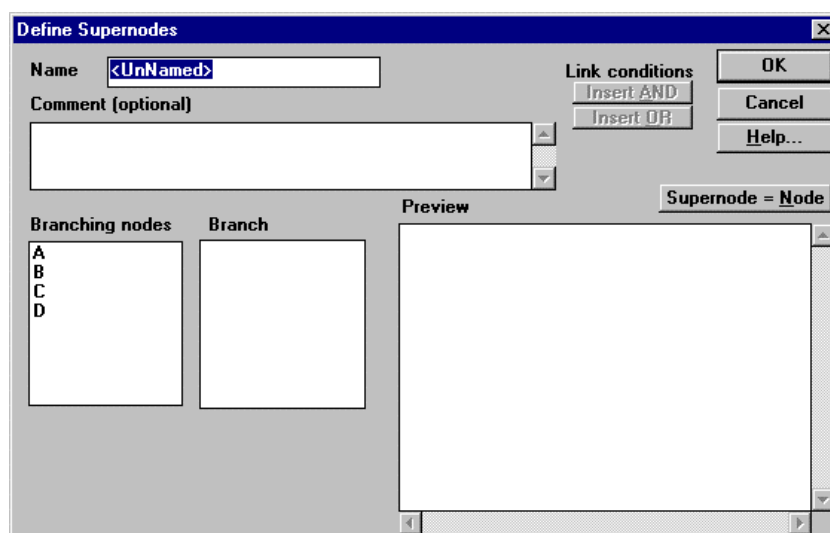
We will assume a simple branching probability so that no conditional parameters need be entered. Select the first (up) branch in this window with a click of the mouse button and then assign the probability for this branch by clicking in the right hand box and entering 0.1. You only need to assign probabilities for n-1 branches in an n-branch node, so for this binary branch you can now click on the 'OK' button. Then click on the 'OK' button in the node definition dialog, and this will enter the data for this branching node.

² Note that all keyword input (and menu items) are in quotes for identification purposes only. The quotes are not part of the input.

Repeat this process for three further nodes, call them 'B', 'C' and 'D', with up and down branches, as for node 'A', and success probabilities of 0.2, 0.3 and 0.4 respectively.

We now need to enter supernode definitions to allow the supernode tree viewer to display the event tree. Remember that you need to first click on 'enddata' to insert the supernode tree before 'enddata'. In the main window, from the 'Insert' menu, first select 'Supernode Tree...'. SOLOMON allows the user to define a number of supernode trees for each model to enable various parts of the tree to be displayed individually. Call this supernode tree 'Tree_1' by entering the name (without the quotes) in the highlighted box, and then click on 'OK'.

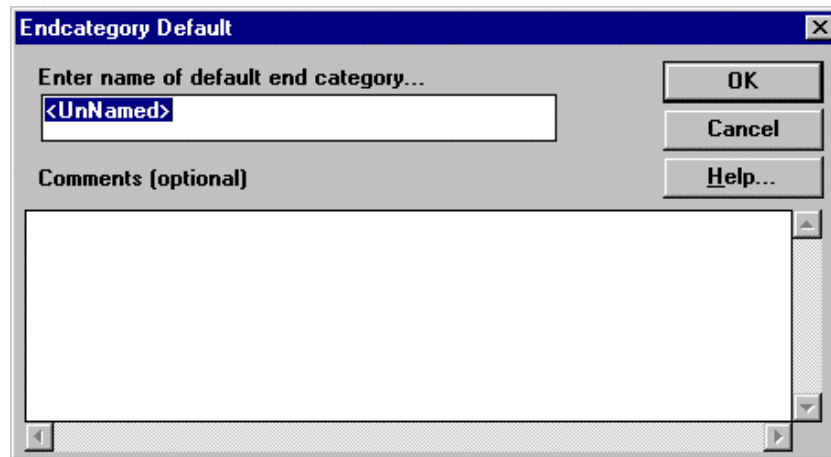
Now you can click on 'enddata' to insert the first supernode, by selecting the 'Insert' menu and selecting 'Supernode...'. This will activate the main supernode dialog box, as shown below:



Enter the name of the supernode which, for simplicity, we will call 'a'. This dialog also shows which nodes are available for selection. For this first case, we will simply select node 'A' by clicking in the 'Branching nodes' box. This then displays the available branches, any of which may be selected by the user. For this example, set the supernode equal to the node, by clicking on the button labelled 'Supernode = Node'. Now click on the 'OK' button in the main supernode entry dialog.

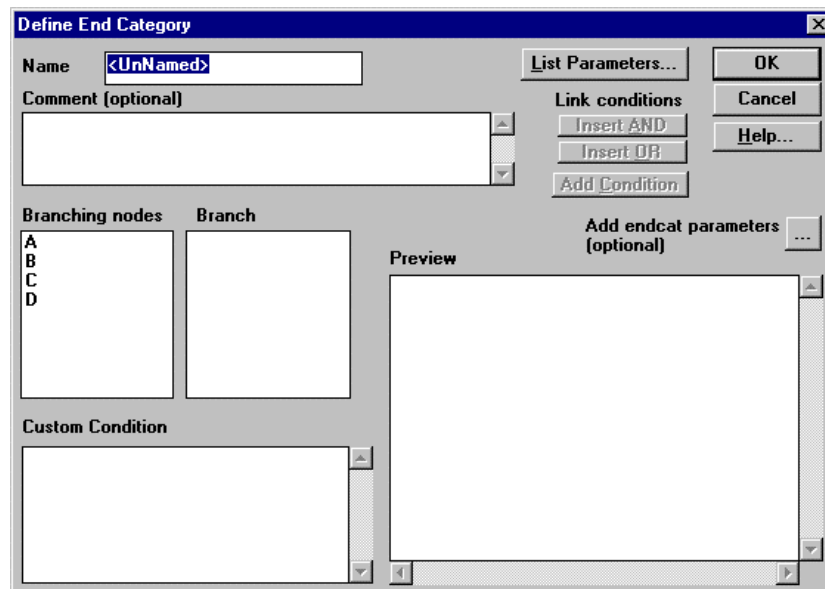
Repeat this process for three more supernodes named 'b', 'c' and 'd', each set equal to the respective supernode. Remember, you need to click on 'enddata' each time to insert the supernodes before 'enddata'.

We now need to define a series of 'endcats', which in a real problem would represent consequences. Remember that you need to first click on 'enddata' to insert the 'endcats' before 'enddata'. First enter a default category, which gathers all the paths through the event tree which have not been binned into pre-defined 'endcats'. From the 'Insert' menu, select 'Endcategory Default...', to bring up the following dialog:



Enter 'default' to replace the highlighted text and click on the 'OK' button.

Now click on 'enddata' once more and then, from the 'Insert' menu, select 'Endcat...' to bring up the following dialog:



As with the supernode tree input, this dialog displays the available nodes (and branches when a node is selected). Enter 'one' for the name of this 'endcat'. We will define this 'endcat' as follows. First select branching node 'A', then select the up branch, and click in the 'Preview' box. This enters the first part of the definition, which takes all paths with an 'up' branch for node 'A'. Now click on the 'Insert AND' button, and then select node 'B', the 'up' branch and click in the 'Preview' box. This has now defined the 'endcat' as a combination of 'up' branches for nodes 'A' and 'B'. Click on the 'OK' button to enter this information into the model.

Repeat this process with 'endcats' two, three and four, defining them as +A-B, -A+B and -A-B respectively, where + represents the 'up' branch and - represents the 'down' branch. Remember that, you need to first click on 'enddata' to insert the 'endcats' before 'enddata'.

This completes the entry of the first example. We will use this input as the basis for further examples, so you should save your work. Use option 'File' then 'Save' and save your work in

file 'Example1.in' in the 'temp' subdirectory in the main SOLOMON directory. The file should look something like this:

```

title "Example 1"
samples 1
cutoff 0
showpaths frequency 0
node A type branching
branch up
branch down

prob up 0.1
node B type branching
branch up
branch down

prob up 0.2
node C type branching
branch up
branch down

prob up 0.3
node D type branching
branch up
branch down

prob up 0.4
supernodetree Tree_1

supernode a

node A
supernode b

node B
supernode c

node C
supernode d

node D
endcategory default default
endcat one

cond ((A up)
      and (B up) )
endcat two

cond ((A up)
      and (B down) )
endcat three

cond ((A down)
      and (B up) )
endcat four

cond ((A down)
      and (B down) )
enddata

```

You can print this file in the usual manner, by selecting 'Print' (from the 'File' menu), and then check it against the above.

You are now ready to evaluate the event tree. Run SOLOMON from the 'Run' menu in the main window and select 'Solomon'. The code will then compile and link your input and run the code. This is done in separate stages; the generate step parses the input and checks for errors, the compile and link step then produces an executable which runs the calculation engines LHS, the event tree program and the show program.³ If there is an error in the input, a message will be displayed, directing the user to the appropriate part of the execution process. From the 'View' menu, you can examine the various run logs and use the 'Go to line...' option to go to the appropriate line in the code. The intermediate and final output files are described in detail in the SOLOMON User Guide (Roberts 1999).

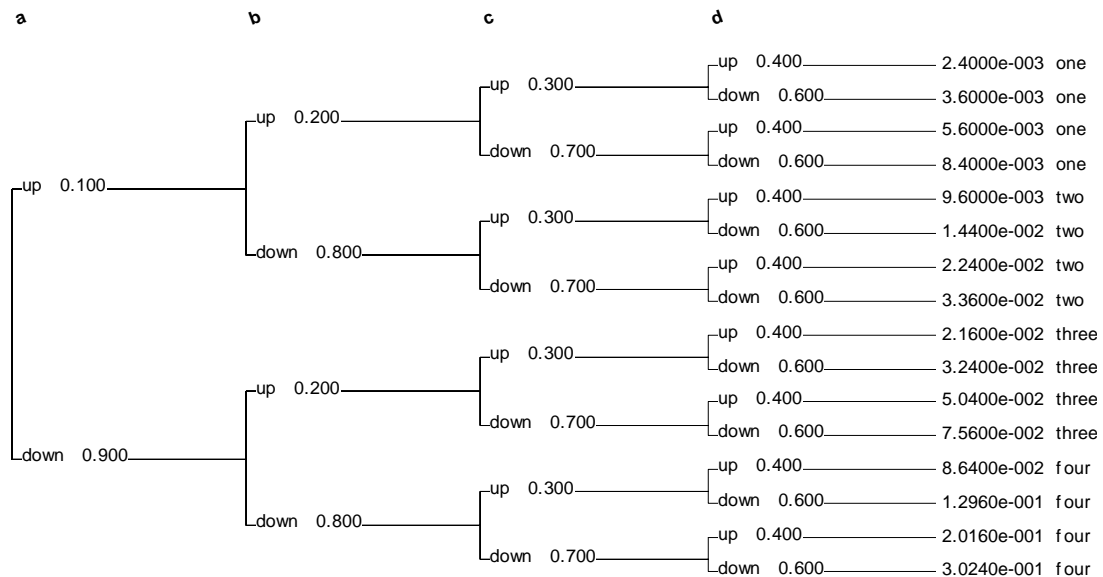
We will now examine the supernode tree. From the 'Run' menu, select 'Supernode Viewer'. This will bring up the main supernode tree viewer window, as below:



The supernode tree file can be opened from the 'File' menu in the usual way, (SOLOMON automatically opens in the directory in which the last run took place (navigation around a directory structure can be achieved in the standard windows manner). The viewer expects to open files with a '.smd' extension which, in this case, is called 'noname', since we did not enter a run name for this simple case. Double clicking on this file name then loads the file into the viewer. Each individual tree can be selected from the 'Tree' menu which, in our case, has just one entry, 'Tree_1'. Selecting this allows the tree to be viewed, and it should look something like the following:

³ Please remember that the calculation engine has been disabled in the evaluation version, and run-times will not be representative.

Title : Example 1
 Run : noname
 Tree : Tree_1



For more information about display options available in the viewer, please refer to the SOLOMON User Guide.

Select 'File' then 'Exit' in the supernode viewer to dismiss the viewer window.

Select 'File' then 'Exit' in the main SOLOMON window.

3.2 EXAMPLE 2

In this example we have included node probabilities that depend on the state of previous nodes. The definition of node C indicates that the probability of it being true is:

$$\begin{aligned}
 P(C = \text{TRUE}) &= 0.33 && \text{if } A=\text{FALSE} \text{ and } B=\text{TRUE} \\
 &= 0.5 && \text{if } A=\text{FALSE} \text{ and } B=\text{FALSE} \\
 &= 0.3 && \text{otherwise}
 \end{aligned}$$

where the convention is that the 'up' branch is TRUE and the 'down' branch is FALSE. Note that these 'conditional probabilities' do not have to total one.

Also, the probability of node 'D' being true depends on the state of node 'C':

$$\begin{aligned}
 P(D=\text{TRUE}) &= 0.6 && \text{if } C= \text{TRUE} \\
 &= 0.4 && \text{otherwise}
 \end{aligned}$$

Thus the probability of all the nodes being false is now $0.9 \times 0.8 \times 0.5 \times 0.6 = 0.216$

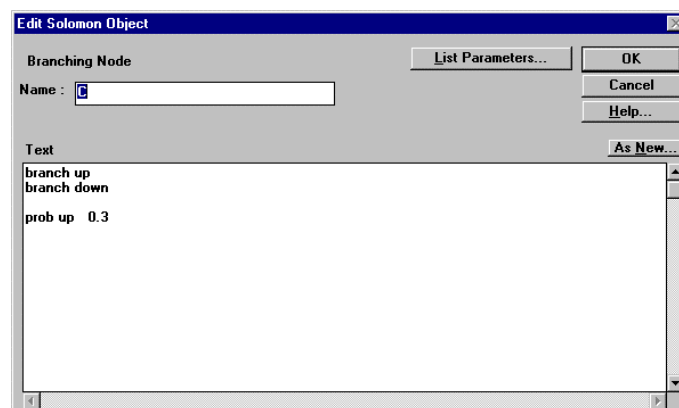
To begin constructing the next example file, first start SOLOMON and dismiss the 'About SOLOMON' window. Select 'File' then 'Open' from the menu. Open file 'Example1.in' in the 'temp' subdirectory in the main SOLOMON directory.

The simplest way to alter an input description is to select an item in the left hand window by clicking on it with the left mouse button, and then clicking in the right hand window to make simple changes using standard windows editing techniques (cut, copy and paste). This method is useful for simple changes, such as to the job title, or for the more advanced user, who is familiar with the SOLOMON input structure.

To change the 'Job Title', simply click on the 'Header Block' in the left hand partition of the main window. If you have only just opened the input file the Header Block will automatically be the currently selected item. The corresponding text entries will be displayed in the right side of the main window. Change the title to 'Example 2'.

For more complex changes, or for a new user, it is perhaps better to use the dialogs to assist the data input process. In this part of the example we will introduce conditional probabilities i.e. we will introduce changes in branching probability based on decisions made earlier in the tree.

Select Node 'C' for editing (by double clicking the light green block labelled 'C' in the left hand window). The following dialog will appear:



Click on the 'As New' button to invoke the standard dialog for this item. The 'Define Branching Node' dialog will appear. This allows full editing of the input.

Click the 'Define Probabilities' button and then enter the following conditional probability (in the top box of the 'Define Branching Probabilities' dialog). Remember that SOLOMON input is case sensitive, so upper case should be used for this example:

(-A) and (+B)

This will define the first conditional expression for node 'C'. Click on the 'up' branch and then enter 0.33 in the 'Probability of Branch' box. Now click on the 'OK' button.

Repeat the process (click 'Define Probabilities' onwards) with the following expression:

(A down) and (B down)

where the alternative naming convention for a node/branch name has been used, and the branching probability 0.5.

Repeat this process for node 'D', with the 'up' branch probability set to 0.6 if 'C' is TRUE.

This completes the changes for this example. We will use this input as the basis for further examples, so you should save your work in a different file. Use option 'File' then 'SaveAs...' and save your work in a file 'Example2.in' in the 'temp' subdirectory in the main SOLOMON directory. Note that the 'save' button in the toolbar, and the 'save' and 'save as' menu items will be greyed out unless you first click in the left hand window of the model. This positive reinforcement feature helps to ensure that the editing session has been completed before saving the file.

The file should look something like this:

```

title "Example 2"
samples 1
cutoff 0
showpaths frequency 0
node A type branching
branch up
branch down

prob up 0.1
node B type branching
branch up
branch down

prob up 0.2
node C type branching
branch up
branch down

prob up 0.3

cond ((-A) and (+B)) prob up 0.33
cond ((A down) and (B down)) prob up 0.5

node D type branching
branch up
branch down

prob up 0.4

cond (+C) prob up 0.6
supernodetree Tree_1

supernode a

cond (+A)
supernode b

cond (+B)
supernode c

```

```

cond (+C)
supernode d

cond (+D)
endcategory default default
endcat one

cond ((A up)
and (B up) )
endcat two

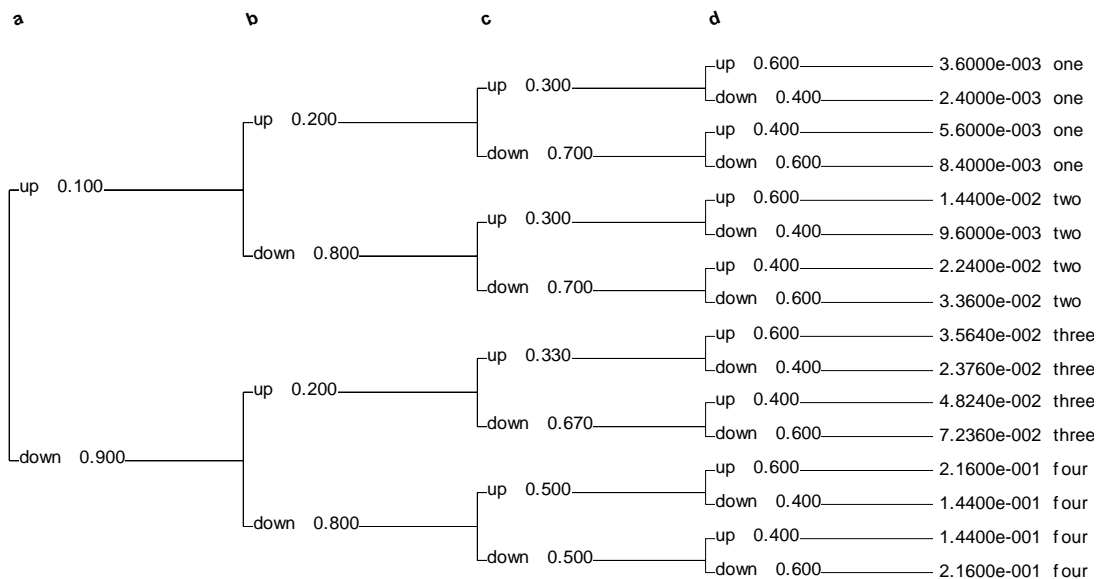
cond ((A up)
and (B down) )
endcat three

cond ((A down)
and (B up) )
endcat four

cond ((A down)
and (B down) )
enddata
    
```

Examine the supernode tree as before, which should now look like this:

Title : Example 2
 Run : noname
 Tree : Tree_1



Select 'File' then 'Exit' in the supernode viewer to dismiss the viewer window.

Select 'File' then 'Exit' in the main SOLOMON window.

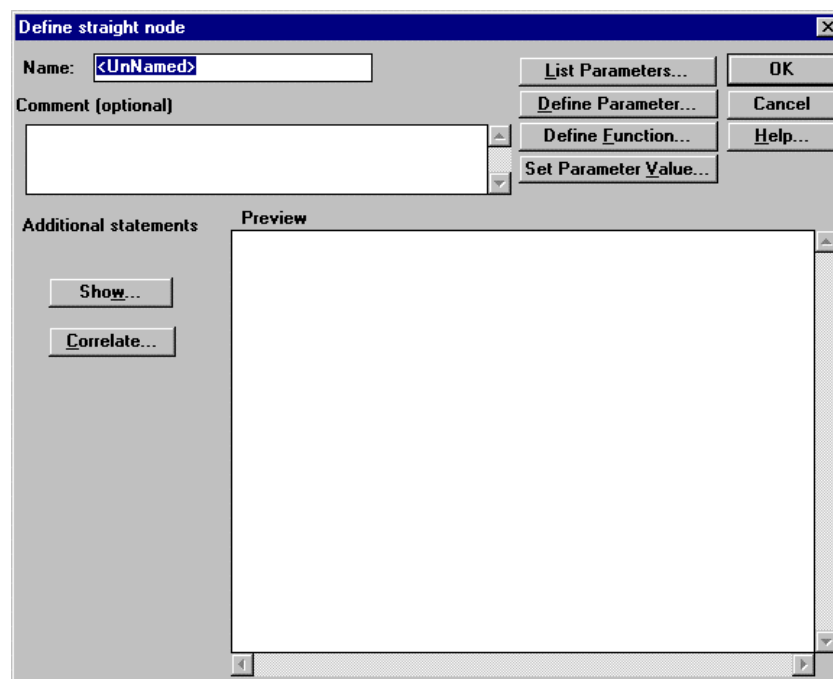
3.3 EXAMPLE 3

In this example we have included some calculations and some 'switches' which could be used, for instance, to represent the different states of a system model.

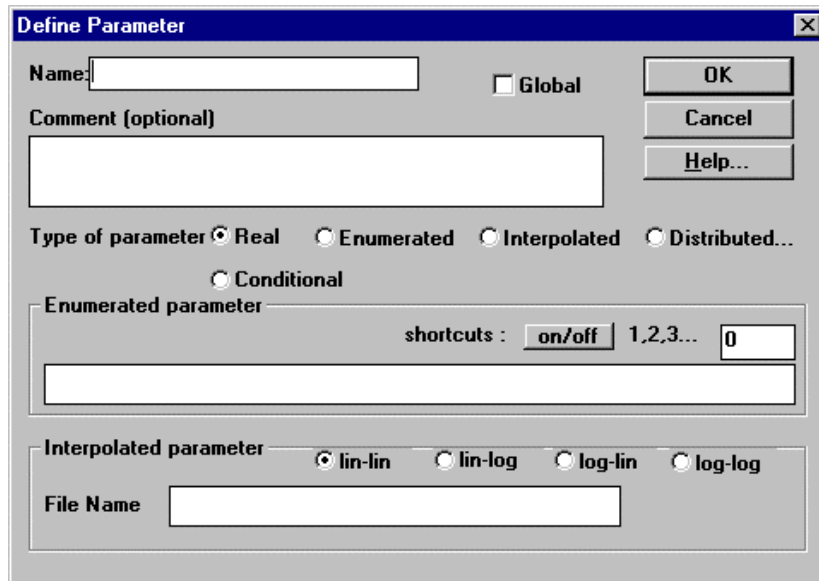
Node 'ZERO' is a 'straight' node - the event tree does not branch at this point but variables can be set and calculations performed within this node. This is one of the most powerful features of SOLOMON, which enables system models to be included within the event tree, which can then influence the progress of an accident sequence.

In node ZERO we will define a variable called 'sprays' which is an enumerated (or 'enum' type) variable. This means that it can take values from a list - in this case the list is 'on' and 'off'. Thus we can set 'sprays = off', or 'sprays = on'. This node could, for example, represent the action of the containment sprays in a nuclear power station, or the automatic fire sprinklers in an industrial plant or building.

We can begin by opening Example2.in, as before, and renaming the case by clicking on the header block and changing the title to 'Example 3'. We now need to insert a straight node, so called because it is used to calculate variable parameters to pass through to branching nodes further down the tree. To do this we click on node 'A' in the main SOLOMON window, and from the 'Insert' menu, we select a straight node, which will bring up the following dialog:

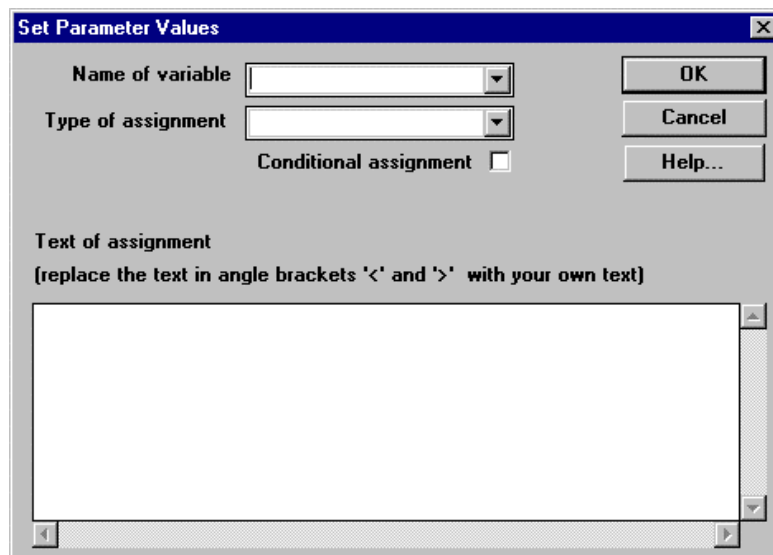


Enter 'ZERO' to name the node, and then click on the define parameter button to bring up the following dialog:



Enter the name 'sprays', click on 'global' to ensure that the results of this node will be available throughout the event tree. Then select Enumerated, click in the Enumerated Parameter box and enter 'on, off' to define the two states for the parameter 'sprays'. If you click on the 'on/off' shortcut button this will automatically enter the states 'sprays_on' and 'sprays_off', and the keywords 'on' and 'off' will need to be altered accordingly in the rest of this tutorial. Click on 'OK' to close the dialog.

Follow the same procedure to define P and Q as real parameters, *remembering to re-set the cursor to the bottom of the Preview window after each entry*. To set the sprays to 'off', click on the 'Set Parameter Value' button, to bring up the following dialog:



Then enter the name of the variable (sprays), select the 'name = value' assignment, and then replace the text <value> (including the brackets) with the word 'off'. Click on the 'OK' buttons to enter the information.

Nodes 'A', 'B' and 'C' remain the same as in Example 2, but a new node called 'CC' will be inserted. Node 'CC' is a straight node, and we use it to set the value of variables 'P' and 'Q' with the following logic:

if containment sprays are off, and node A is TRUE, then $P = 0.4$
 if containment sprays are on, and node A is TRUE, then $P = 0.5$
 otherwise, $P = 0$

if node B is TRUE then $Q = P \times 0.1$
 if node B is FALSE then $Q = P \times 0.2$

Select node 'D' as the insertion point by clicking on its representation in the left hand partition of the main window. As for node 'ZERO', we begin by selecting a 'straight node' from the 'Insert' menu, and name it 'CC'. Now select 'Set Parameter Value' and click on the drop-down menu for the 'Name of Variable' to select 'P' (this was not available in node zero because no parameters had been defined when we entered the information for this node). We first need to initialise 'P' to 0.0, which is accomplished by selecting 'name = value' and then setting the text <value> with 0.0. Click on 'OK', set the cursor to the next line in the preview window and then repeat the process, this time checking the 'Conditional assignment' box before selecting the 'name = value' option, and replacing the text <condition> with:

((sprays off) and (+A))

and the text <value> with:

0.4

Then repeat with the following conditional expression

((sprays on) and (+A)) 0.5

remembering to reset the position of the cursor to the bottom of the Preview window to ensure that the insertion point is correct.

Repeat this procedure for the parameter 'Q', using the 'name = calc' option, replacing the text <condition> with:

(+B)

and the text <expression> with:

(P * 0.1)

and then repeating with:

(-B)

and

(P * 0.2)

Once the 'OK' button has been clicked and the information entered, the input should now look like the following:

```
node CC type straight
set P to 0.0
set cond( (sprays off) and (+A) ) P to 0.4
set cond( (sprays on) and (+A) ) P to 0.5
set cond(+B) Q to calc ( P * 0.1 )
set cond(-B) Q to calc( P * 0.2 )
```

The value of 'Q' is then used as the probability of node 'D' being true, using the 'As New' button and replacing the probability of the up branch with the value of 'Q'.

The input file should now look like this:

```
title "Example 3"
samples 1
cutoff 0
showpaths frequency 0
node ZERO type straight

#
define enum sprays(on, off) global

#
define real P global

#
define real Q global

set sprays to off

node A type branching
branch up
branch down

prob up 0.1
node B type branching
branch up
branch down

prob up 0.2
node C type branching
branch up
branch down

prob up 0.3

cond ((-A) and (+B)) prob up 0.33
cond ((A down) and (B down)) prob up 0.5
node CC type straight

set P to 0.0
```

set cond((sprays off) and (+A)) P to 0.4

set cond((sprays on) and (+A)) P to 0.5

set cond(+B) Q to calc (P * 0.1)

set cond(-B) Q to calc(P * 0.2)

node D type branching

branch up

branch down

prob up Q

supernodetree Tree_1

supernode a

node A

supernode b

node B

supernode c

node C

supernode d

node D

endcategory default default

endcat one

cond ((A up)

and (B up))

endcat two

cond ((A up)

and (B down))

endcat three

cond ((A down)

and (B up))

endcat four

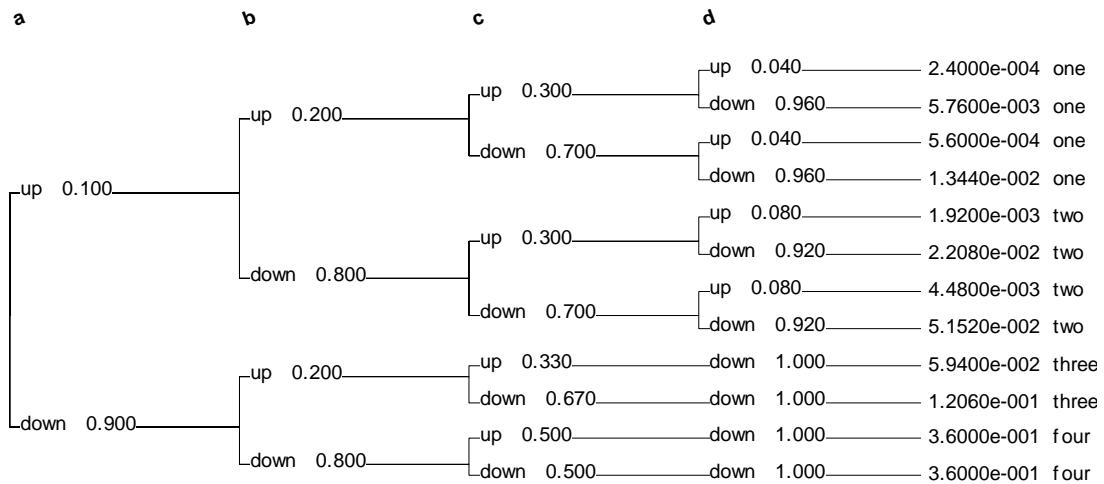
cond ((A down)

and (B down))

enddata

Running this case should produce the following event tree:

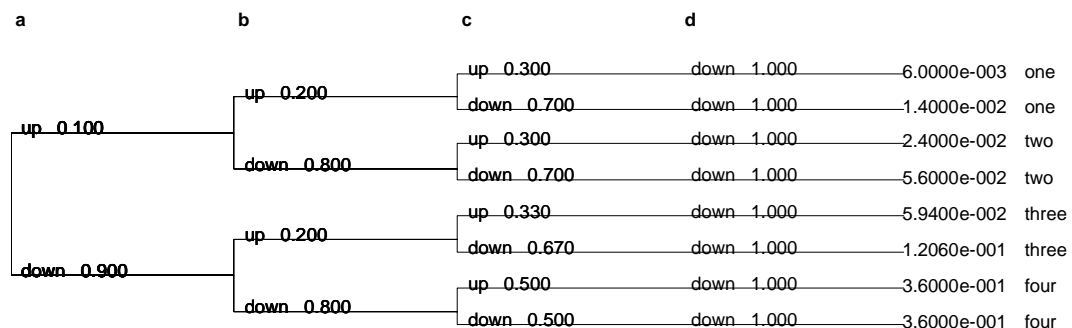
Title : Example 3
 Run : noname
 Tree : Tree_1



3.4 EXAMPLE 4

This example is the same as Example 3, except that the containment sprays are set to 'on'. This is most easily achieved by selecting node 'ZERO' and resetting the sprays on. Running this calculation will result in the following supernode tree:

Title : Example 4
 Run : noname
 Tree : Tree_1



3.5 EXAMPLE 5

In this example we include a randomly distributed variable in node 'ZERO', with the variable 'R' defined as being randomly distributed with a normal distribution function of mean 0.5 and standard deviation 0.1. This requires the user to specify the number of samples for input into the LHS program, which is required in sampling the probability space.

In node 'CC', this variable 'R' is used in a calculation to set the value of 'Q', which is used as the probability of node 'D'.

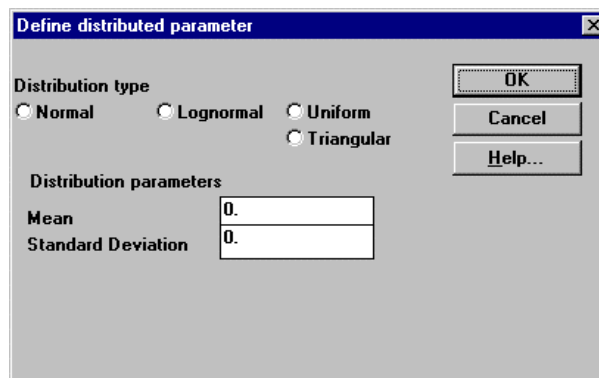
Although SOLOMON can draw this tree, it can no longer include the node probabilities, since it cannot print the randomly distributed variable (in fact, it prints the first sampled values of the node probabilities). However, the statistics for the end categories are printed in file *.out, where, in this case, * is replaced with noname.

The probability of end category +A+B+C+D (i.e. all nodes are true) will be randomly distributed with a mean value of 3×10^{-4} .

The probability of end category +A-B+C+D (i.e. all nodes are true except node B) will be randomly distributed with a mean value of 2.4×10^{-3} .

First set the title to Example 5 and the number of samples to 100, by selecting the header block and then manually editing the text.

To insert the random variable 'R', we double click on node 'ZERO' and 'As New'. Click on the 'Define Parameter' button and enter the name 'R'. Select distributed, from the list of parameter types, which brings up the following dialog:



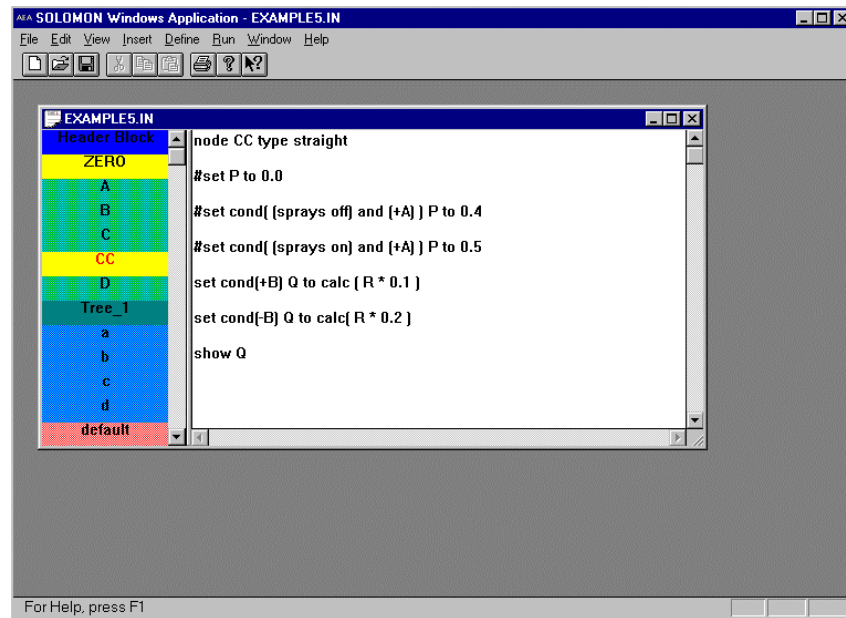
Select normal distribution, enter the mean and standard deviation values of 0.5 and 0.1, and then click on 'OK' to close each dialog.

We now need to set the value of 'Q' depending on the state of node 'B' using the randomly distributed variable 'R'. The simplest method is to edit the text in right hand partition of the main window, by selecting node 'CC' in the left hand side of the main window and then clicking in the right hand side and replacing 'P' with 'R' in the probability definitions for 'Q'.

The show parameter feature is invoked by inserting 'show Q' in the final line. This could have been inserted using the show parameter dialog, which enables the user to list which parameters

are available, but for this simple example, it is more convenient to simply type in the command manually.

The probability definitions for 'P' should be commented out by editing the right hand window and inserting the # character in the appropriate lines to render the input for this node as in the screenshot below.



The resulting input file should look like this:

```
title "Example 5"
samples 100
cutoff 0
showpaths frequency 0
node ZERO type straight

#
define enum sprays(on, off)

#
define real P global

#
define real Q global

set sprays to on

#
define distribution R normal 0.500000 0.100000

node A type branching
branch up
branch down

prob up 0.1
node B type branching
branch up
```

```

branch down

prob up 0.2
node C type branching
branch up
branch down

prob up 0.3

cond ((-A) and (+B)) prob up 0.33
cond ((A down) and (B down)) prob up 0.5
node CC type straight

#set P to 0.0

#set cond( (sprays off) and (+A) ) P to 0.4
#set cond( (sprays on) and (+A) ) P to 0.5

set cond(+B) Q to calc ( R * 0.1 )

set cond(-B) Q to calc( R * 0.2 )

show Q

node D type branching
branch up
branch down

prob up Q

supernodetree Tree_1

supernode a

node A
supernode b

node B
supernode c

node C
supernode d

node D
endcategory default default
endcat one

cond ((A up)
and (B up) )
endcat two

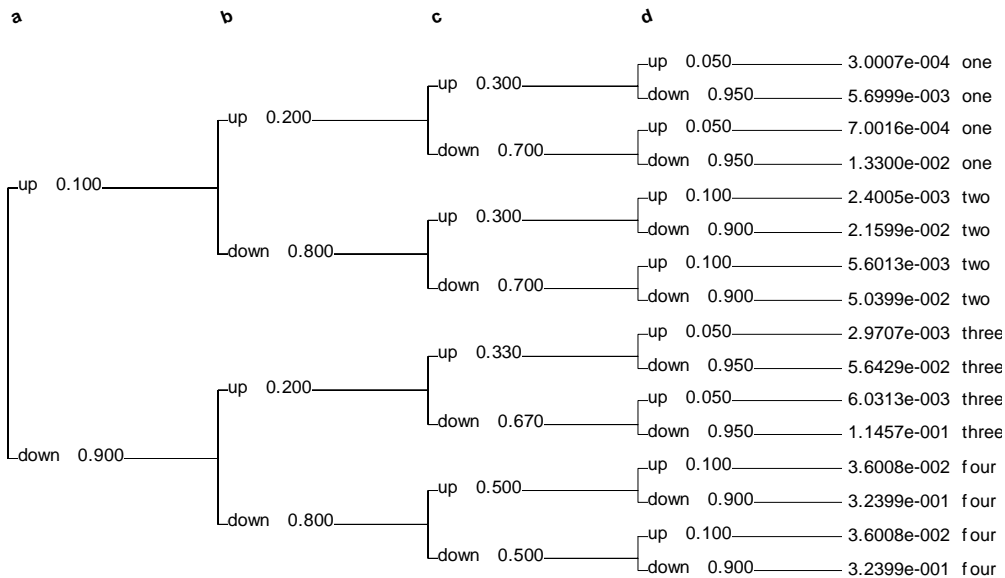
cond ((A up)
and (B down) )
endcat three

cond ((A down)
and (B up) )
endcat four

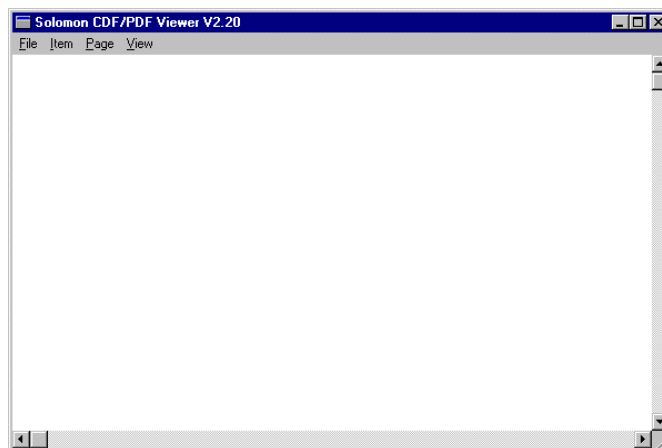
```

```
cond ((A down)
      and (B down) )
enddata
```

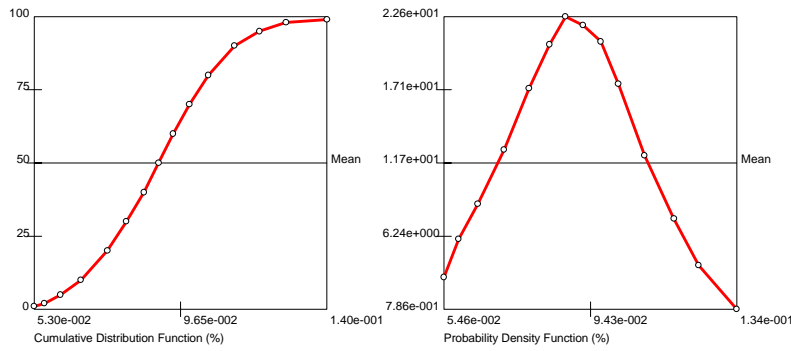
Title : Example 5
 Run : noname
 Tree : Tree_1



The supernode tree (for the first sample) looks like this:
 Now select the PDF/CDF viewer, from the Run menu. This will bring up the following dialog:



Select 'Open' from the 'File' menu, and select 'Variable Files' from the file type list. Double click on the file 'noname.sh1' and then select the 'Item' menu. This will present you with the value of 'Q at node CC', which is the single parameter requested as a show variable in the input. When you select the PDF/CDF output for this parameter, the viewer will display the following:



Variable: Q at node CC
 Project Title: Example
 Run: noname

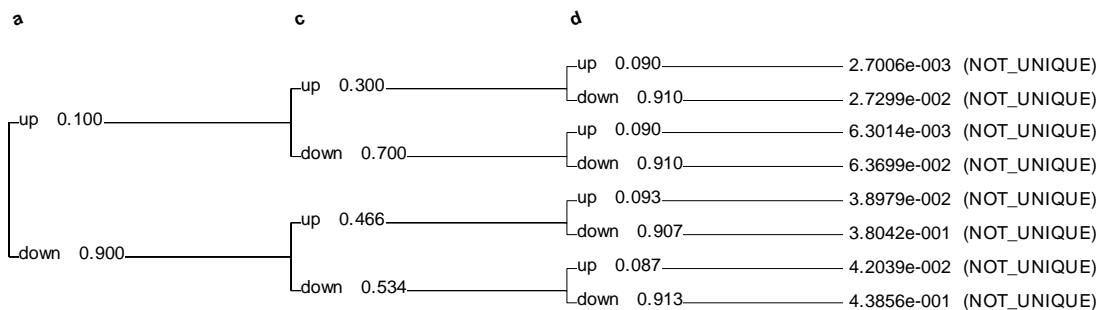
3.6 EXAMPLE 6

In this example, the event tree is identical to the one in Example 5, except that we demonstrate how supernodes can be used to condense the event tree. By redefining the supernodes as:

supernode a node A
 supernode c node C
 supernode d node D

i.e. we remove the information from node 'B'. This can be achieved by selecting supernode 'b' and using the delete option in the 'Define' drop down menu (alternatively, if the supernode may be needed in a later calculation, it can be deactivated by simply clicking on it with the right mouse button). The event tree now looks like this:

Title : Example 6
 Run : noname
 Tree : Tree_1



The probability values for node 'C' are consistent with Bayes theorem since, from the SOLOMON input file, we have defined

For A=TRUE:

$$P(C=TRUE) = 0.3$$

which agrees with the probability for node 'C' where 'A' is TRUE, shown in the above tree.

For A=FALSE:

$$P(B=TRUE) = 0.2$$

$$P(C=TRUE \mid B=TRUE) = 0.33$$

$$P(C=TRUE \mid B=FALSE) = 0.5$$

and so

$$\begin{aligned} P(C=TRUE) &= P(C=TRUE \mid B=TRUE) \cdot P(B=TRUE) + \\ &\quad P(C=TRUE \mid B=FALSE) \cdot P(B=FALSE) \\ &= 0.33 \times 0.2 + 0.5 \times 0.8 \\ &= 0.466 \end{aligned}$$

which agrees with the probability for node 'C' where 'A' is FALSE, shown in the above tree.

The input file is:

```

title "Example 6"
samples 1
cutoff 0
showpaths frequency 0
node A type branching
branch up
branch down

prob up 0.1
node B type branching
branch up
branch down

prob up 0.2
node C type branching
branch up
branch down

prob up 0.3

cond ((-A) and (+B)) prob up 0.33
cond ((A down) and (B down)) prob up 0.5

node D type branching
branch up
branch down

prob up 0.4

cond (+C) prob up 0.6
supernodetree Tree_1

supernode a

node A
supernode c

```

```

node C
supernode d

node D
endcategory default default
endcat one

cond ((A up)
      and (B up) )
endcat two

cond ((A up)
      and (B down) )
endcat three

cond ((A down)
      and (B up) )
endcat four

cond ((A down)
      and (B down) )
enddata

```

4 Representative Nuclear Plant Model

This example provides the user with an insight into the range of features typical of a more representative model, and is based on a typical Level 2 PSA application for a NPP. It is intended for demonstration purposes only, and is not based on any particular reactor design, although it contains many of the features typical of a Pressurised Water Reactor.

The model calculates the failure frequency for a simple containment with one compartment and includes a simple model to evaluate the probability of a hydrogen burn. Four Plant Damage States (PDSs) are analysed in series, covering large and small break loss of coolant accidents, station black-out and loss of feed-water. Engineered safeguards are represented in the definition of the Plant Damage State by the availability of containment sprays.

Each accident sequence is divided into four time-frames, each covering a period of the transient with broadly similar characteristics:

Time-frame 1: from accident initiation to the onset of core damage (defined by the start of core oxidation).

Time-frame 2: from the onset of core damage to core support plate failure.

Time-frame 3: from core support plate failure to failure of the lower head.

Time-frame 4: from failure of lower head to the end of accident sequence.

Base-line thermal-hydraulic data could be provided by calculations using a system code, such as MAAP or MELCOR. Typical data for containment temperature, pressure, vapour fraction and hydrogen mass are input into the SOLOMON model.

During the first time-frame, no hydrogen is released to the containment and the potential for containment failure is dependent on the rise in pressure due to steam/water release from the primary circuit. Recovery actions are modelled by a simple probability that the operators manage to regain control of the accident sequence by following the relevant accident management procedures.

In the second time-frame, the flammability of any hydrogen released from core oxidation is tested against the hydrogen burn model. This includes tests of global/local flammability, detonability, completeness of burn and probability of ignition. The containment pressure increase resulting from a hydrogen burn is added to the base-line pressure and a probability of failure of the containment is then calculated. Again, recovery actions are factored into the potential for containment survival.

Later time frames are similar to the second time-frame, apart from the need to track hydrogen inventory in order to account for hydrogen removed during burns occurring in previous time-frames, and the addition of carbon monoxide to the containment from core concrete interactions in time-frame 4.

To give an indication of the flexibility of the supernode concept, supernode trees are provided for the full event tree, time-frame 4 only, key nodes and failure/survival of the containment. Two simple end-categories are defined for failure of the containment before (early) and after (late) lower head failure, with default representing survival of the containment.

This model is necessarily simplified, but could be extended in a number of ways. SOLOMON input is sufficiently flexible that the user could include several compartments in the containment, or the capability to model partial failures, e.g. at containment penetrations, which may result in a leak rather than catastrophic failure, with completely different source term characteristics. More time-frames could be included, to model the time development of accident sequences in more detail, and additional accident sequences could be introduced. Additional physical models could be included such as, for example, core concrete interactions, Direct Containment Heating (DCH), steam explosions etc. The use of the functionality of the C programming language for constructing physical models means that their complexity is only limited by development time-scales and current understanding of the phenomena involved.

5 Supplied files

All files for these tutorials are supplied in subdirectory 'Examples' below the main SOLOMON directory. The complete set of files (input, intermediate and output) for each example is contained in a lower level subdirectory.

Currently we supply the following sets of files:

Subdirectory	Input file(s)
Examples\Example1	Example1.in
Examples\Example2	Example2.in
Examples\Example3	Example3.in
Examples\Example4	Example4.in
Examples\Example5	Example5.in
Examples\Example6	Example6.in
Examples\Model	Baseuncr.in, Aic.dt1 and Failure.prs

In addition, a full set of output files is provided for each example, which can be opened with the appropriate viewer. These may be used to check your results if you are using the full version of the product, or to explore the features of the viewers if you are using the evaluation version of the product. You will need to select the appropriate application to view a file, each of which filters file names with specific extensions, as below.

SOLOMON opens files with a '.in' extension

The PDF/CDF viewer opens files with a '.out' or '.sh1' extension

The Supernode viewer opens files with a '.snd' extension

For more details on these applications please refer to the SOLOMON User Guide.

6 References

Roberts G J, Holford G F and Maidment C (1999). SOLOMON Containment Event Tree Program - A User Guide. November 1999.