Transactions of the Korean Nuclear Society Autumn Meeting
Gyeongju, Korea, October 27-28, 2016

NuFTA 2.0: New Templates and an Automatic Generator of Fault Tree for NuSCR

Junik Son, Yonghyun Kim, Kukbin Jeong, Dong-Ah Lee *, Junbeom Yoo
Division of Computer Science and Engineering, Konkuk University
1 Hwayang-dong, Gwangjin-gu, Seoul, 143-701, Republic of Korea
**Corresponding author: ldalove@konkuk.ac.kr*

1. Introduction

A nuclear power plant control system is one of safety critical systems whose accidents may result in critical damage to human lives or loss of properties. Fault Tree Analysis (FTA) [1] is one of hazard analyses. It analyzes trees which contain causes of a fault in a system. FTA is widely used for high-hazard industries: aerospace, nuclear power, chemical industry, etc. Software Fault Tree Analysis (SFTA) [1] is also a hazard analysis method that applies FTA in software.
This paper introduces new templates for generating fault tree from NuSCR [2] and an automatic generator of fault trees using the templates [3]. NuSCR is a formal specification language used for specifying software requirements of KNICS (Korea Nuclear Instrumentation and Control Systems) RPS (Reactor Protection System) in Korea. A previous version of NuFTA [4] was too difficult to analyze its results because of the size of fault trees. To complement disadvantages of the previous version, the NuFTA 2.0 with the new templates generates the smaller size of fault trees which is concise enough to analyze the trees.
This paper is organized as follows. In Section 2, to help readers better understand NuFTA, we explain NuSCR and FTA precisely. Section 3 introduces new NuFTA templates and NuFTA 2.0. Section 4 shows a case study that about fault tree generated by NuFTA 2.0 using real NuSCR. Finally, Section 5 concludes the paper.

2. Background

2.1 NuSCR

NuSCR [2] is a formal software requirements specification method of KNICS RPS in Korea for the digital nuclear power plants control system. NuSCR has three variable models: function variables, history variables and timed history variables. Function variables specify the mathematical functional behavior of a system. History variables specify the state-based behavior of a system. And timed history variables specify the time-related behavior of a system. Each variable is a defined Structured Decision Table (SDT), Finite State Machine (FSM), Timed Transition System (TTS). Fig. 1 depicts a simplified example of Function Overview Diagram (FOD) in NuSCR. FOD describes data flow between the variables from inputs to outputs.

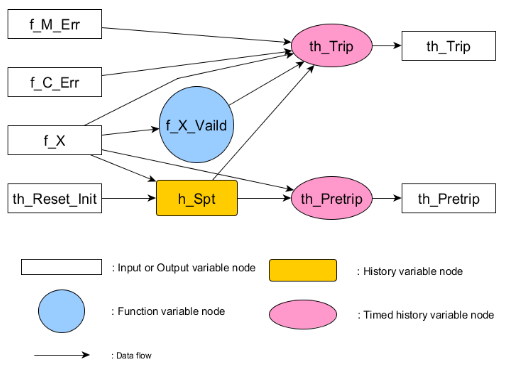

Fig. 1 A simplified example of FOD in NuSCR

2.2 Software Fault Tree Analysis

Software Fault Tree Analysis (SFTA) [1] is a hazard analysis method for software of safety critical systems. SFTA uses a defined-fault and analyzes fault trees contain causes of the fault. Fig.2 depicts an example of software fault tree. As Fig.2, terminal causes of the fault at the top are called leaf nodes. In SFTA, in order to automatically generate fault trees, templates are used.

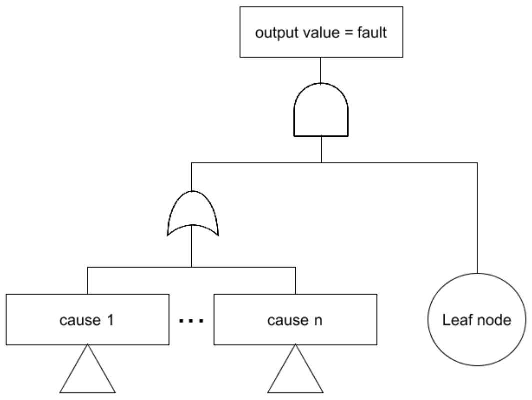

Fig. 2 An example of software fault tree

3. Automatic Fault Tree Generator

3.1 Templates of Fault Tree

3.1.1 The template of SDTs

SDT is Condition/Action tables, which represents the actions (assignment statements) performed if their guiding conditions (condition statements) are satisfied [2]. Fig. 3 depicts an example of SDT. For example, if f_X is a value between k_X_Min and k_X_Max, the output value of f_X_Valid is 0 in Fig. 3.

Conditions		
k_X_Min <= f_X <= k_X_Max	T	F
Actions		
f_X_Valid := 0	O	
f_X_Valid := 1		O

Fig. 3 An example of SDT

Fig.4 depicts the previous version of the fault tree template for SDT [3]. It draws a tree using only condition statements. Action statements, however, is able to be causes when an output of other variable node is appeared in an output of a current SDT. Therefore, we add an action statement in the new version of the template. Fig. 5 depicts the new version of the template for SDT. The action statement is only described when an output of the action is related to a value of an input and condition statements are not related to the value of the input. Condition statements are modified when condition statements are related the value of the input.

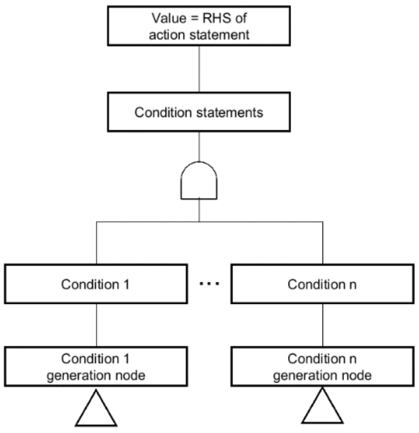
Value = RHS of action statement
Condition statements
Condition 1
Condition n
Condition 1 generation node
Condition n generation node

Fig. 4 The previous version of the template for SDT

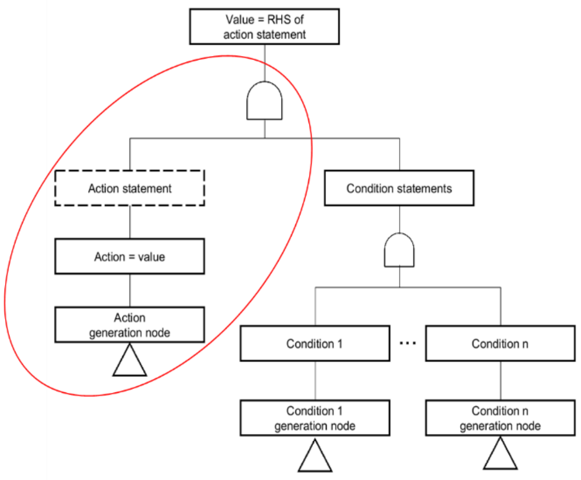

Fig. 5 The new version of template for SDT

3.1.2 The template of FSMs

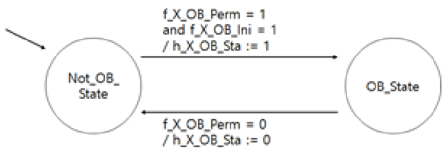
f_X_OB_Perm = 1 and f_X_OB_Ini = 1 / h_X_OB_Sta := 1
Not_OB_State
OB_State
f_X_OB_Perm = 0 / h_X_OB_Sta := 0

Fig. 6 An example of FSM

As Fig.6, History variable node is defined as FSMs. FSM consists of finite number of states, transitions between states, and labels on each transition. Labels are the "Conditions/Actions" statements which are same as that of SDTs [2]. Fig. 7 depicts the previous version of the fault tree template for FSM [3].

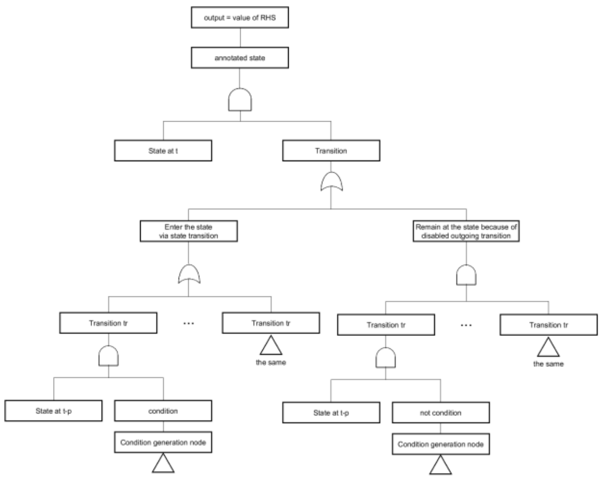

Fig. 7 The previous version of the template for FSM

The previous version generates duplicate conditions in a tree. It makes the size of fault tree enormous. So we modified the template to reduce duplicate conditions. Because action statement is able to be causes, we add action statement in the new version of the template. In the case of "Enter the states via state transition", condition statements are modified when an assignment of state is related to value of input. In the case of "Remain at the state because of disabled outgoing transition", the output value of the action statement is at value which was a previous cycle output value of current FSM. This two cases are added in a new version of the template. Fig. 8 depicts a new version of the template for FSM

3.1.3 The template of TTSs.

The timed history variable node is defined as TTSs. TTS is one of FSM extended with the timing constrains [a, b] in transition conditions. [a, b] means the time duration between time a and b [2]. A template of TTS is same FSM except part of timing constrains [3]. So, the template is changed as same as FSM,

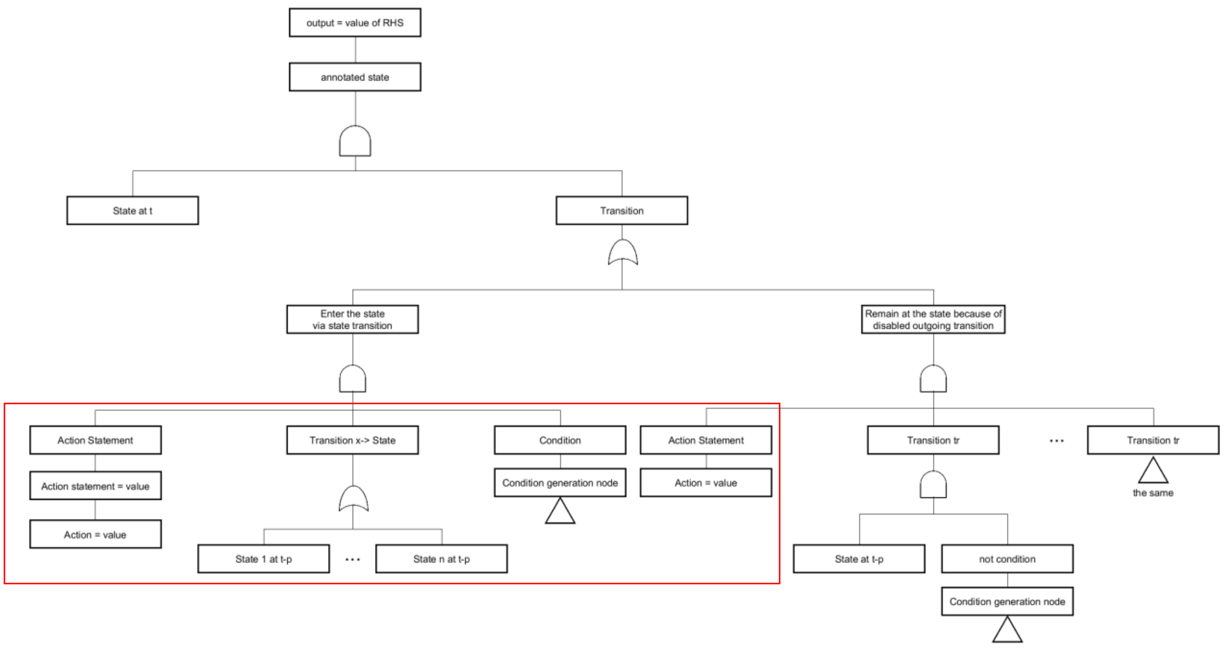

Fig. 8 The new version of template for FSM

3.2 NuFTA 2.0

We developed the NuFTA 2.0 in Eclipse Rich Client Platform based on Java. The NuFTA 2.0 reads XML files written in NuSCR and automatically draws the fault tree according to an expert-defined fault. The Fault tree analysis process in NuFTA 2.0 is as follows:

① The NuFTA 2.0 reads the information from the XML file and stores the object for each node.
② The NuFTA 2.0 identifies one output node which was defined as a fault by an expert.
③ The NuFTA 2.0 draws Fault Tree according to the template. The NuFTA 2.0 checks the generated last node and determines whether to extend the fault tree. The NFTA 2.0 defines values of input variable nodes as basic events and the value of the previous cycle as undeveloped events.
④ Until all leaf nodes are basic events or undeveloped events, continue on step ③.
⑤ As it is Shown in the Fig. 10, The NuFTA 2.0 creates a logical formula which based on the generated Fault Tree.
⑥ Logical formula follows the following structure.

```
P is n period time.
<Logical formula> := <State expr> ∧ <Conditions>
<State expr>:=<node name> == (<original state name> at t
              ∧(<annotated state expr>))
<annotated state expr> := <annotated state name> <time>
                          | ∨<annotated state expr>
<time> := at t-p | for [<Nature number> , <Nature number>]
<node name>:= <name>_state
<annotated state name> := <original state name>_<number>
<number> := [0..9] | <number>
<Conditions> := <State expr> | <Input node condition>
<Input node condition> := <Input value>
                          | <operation> <Input node condition>
<Input value> := <Constant expr> | <Range expr>
                 | <Boolean expr>
<operation> := ∧ | ∨
<Constant expr> := <Input name> = <Integer>
<Range expr> := <Integer> <= <Input name> <= <Integer>
<Boolean expr> := <Input name> = (true | false)
```

4. Case Study

We demonstrate usability and effectiveness by applying The NuFTA 2.O to RPS BP specification written in NuSCR. Fig. 9 is a part of FOD for the Pressurizer low pressure trip logic.

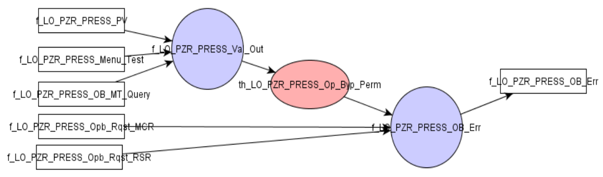
f_LO_PZR_PRESS_PV
f_LO_PZR_PRESS_Menu_Test
f_LO_PZR_PRESS_OB_MT_Query
f_LO_PZR_PRESS_Opb_Rqst_MCR
f_LO_PZR_PRESS_Opb_Rqst_RSR
f_LO_PZR_PRESS_Val_Out
th_LO_PZR_PRESS_Op_Byp_Perm
f_LO_PZR_PRESS_OB_Err
f_LO_PZR_PRESS_OB_Err

Fig. 9 A part of FOD
for the Pressurizer low pressure trip logic

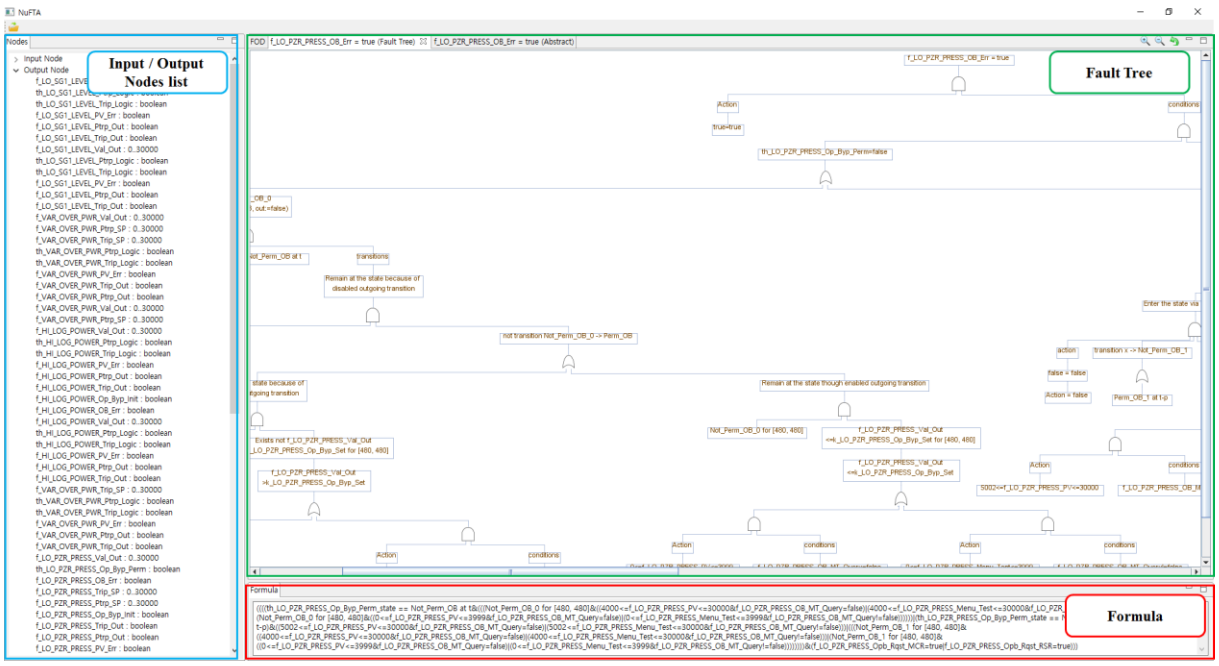

Fig. 10 A result of generating fault tree and formula in NuFTA 2.0

This FOD's output value, f_LO_PZR_PRESS_OB_Err, is a decision block of bypass operation. Fig.10 depicts a fault tree and a logical formula that is created when the output value set true. In the Fig. 10, left part is a list of input / output nodes. In the Fig. 10's right upper part, the fault tree was created to fit the template of each linked variable nodes. In the Fig. 10's right lower part, the NuFTA 2.0 displays a logical formula.

When we check result of the logical formula, we extracted one of the input combination that causes fault as shown below.

```
((((th_LO_PZR_PRESS_Op_Byp_Perm_state ==
Not_Perm_OB at t&
(((Not_Perm_OB_0 for [480, 480]&
((4000<=f_LO_PZR_PRESS_PV<=30000&
f_LO_PZR_PRESS_OB_MT_Query=false)|
(4000<=f_LO_PZR_PRESS_Menu_Test<=30000&
f_LO_PZR_PRESS_OB_MT_Query!=false))&
(f_LO_PZR_PRESS_Opb_Rqst_MCR=true|
f_LO_PZR_PRESS_Opb_Rqst_RSR=true))
```

And for measuring performance, we run NuFTA 1.0 and 2.0 at the same environments and conditions. Table 1 shows the comparison of the fault tree drawing time and the length of the formula between NuFTA 1.0 and 2.0.

Table 1 The comparison of the fault tree drawing time and the length of the formula between NuFTA 1.0 and 2.0

	Drawing time (ms)	Length of formula (Byte)
NuFTA 1.0	1759915	916140
NuFTA 2.0	738	1012

5. Conclusion and Future Work

This paper introduces the NuFTA 2.0 which automatically generates fault tree and logical formula from NuSCR to assistance FTA. The result of case study suggests improvement of drawing time and length of formula. One of our next goals is going to study how to analyze time from 0 to 'a' which is minimal value of time constrains [a, b] in TTS. Another is finding minimal cut-set, the smallest list of causes that is essential to cause the fault, in order to analyze formula.

ACKNOWLEDGEMENT

This paper was supported by the Ministry of Science, ICT & Future Planning

REFERENCES

[1] N. G. Leveson and P. R. Harvey, "Software fault tree analysis," *Journal of Systems and Software,* vol. 3, pp. 173-181, 1983.

[2] Junbeom Yoo, Taihyo Kim, Sungdeok Cha, Jangsu Lee and Han Seong Son, "A Formal Software Requirments Specification Method for Digital Nuclear Plants Protection Systems," *Journal of Systems and Software,* vol. 74, no. 1, pp. 73-83, 2005.

[3] Taeho Kim, "Property-based Theorm Proving and Template-based Fault Tree Analysis of NuSCR Requirements Specification," in *Ph.D Diessertation*, KAIST, 2006.

[4] Sanghyun Yun, Dong-Ah Lee and Junbeom Yoo, "NuFTA: A Case Tool for Automatic Software Fault Tree Analysis," in *Transactions of the Korean Nuclear Society Spring Meeting*, Pyeongchang, Korea, May 27-28, 2010.