2011 Fifth International Conference on Secure Software Integration and Reliability Improvement - Companion

A Domain-Specific Safety Analysis for Digital Nuclear Plant Protection Systems

Sanghyun Yoon
College of Info. and Comm.
Konkuk University
Seoul, South Korea
pctkdgus@konkuk.ac.kr

Jaeyeon Jo
College of Info. and Comm.
Konkuk University
Seoul, South Korea
tm77@konkuk.ac.kr

Junbeom Yoo
College of Info. and Comm.
Konkuk University
Seoul, South Korea
jbyoo@konkuk.ac.kr

***Abstract*—Rigorous safety demonstration through safety analysis is strongly mandated for safety-critical systems. Nuclear plant protection systems often use techniques such as FTA, FMEA and HAZOP. Safety experts perform them manually, and quality of the analysis totally depends on the ability and experience of the experts. If we restrict the application domain of safety analysis into specific critical failures, we could automate a large part of the analysis and also improve its quality too. This paper proposes a domain-specific safety analysis technique, *NuFTA*, for nuclear plant protection systems. NuFTA mechanically constructs a software fault tree of nuclear reactor protection systems specified with NuSCR requirement formal specification language. The root failures of the fault tree constructed through NuFTA are restricted into '*shutdown*' events of nuclear reactors, which is the most important event in the domain. Within the domain specific restrictions, NuFTA can construct software fault trees mechanically and aid safety experts' analyses efficiently.**

I. Introduction

Safety-critical systems (e.g. nuclear power plants and airplanes) require rigorous quality demonstration prior to operational approval issued by regulation agencies. Safety analysis [7] tries to assure systems safety through performing various safety analysis techniques together such as FTA (Fault Tree Analysis), FMEA (Failure Mode and Effect Analysis) and HAZOP (Hazard and Operability). Safety experts apply the techniques into target systems manually. Therefore, quality and correctness of the analysis results are totally depending on the knowledge and experiences of safety experts.

For software in safety-critical systems, many software fault tree analysis techniques [19], [15], [5], [10], [9], [13] have been researched, focusing on mechanizing the construction of fault trees from software specification or program code. They all have a common intrinsic limitation; containing information only on their specifications or source code and not showing anything beyond that captured in the sources. Safety experts are still needed even after constructing fault trees mechanically. Therefore, the software fault tree analysis techniques above cannot substitute the analysis itself but can only be used as an aid for constructing fault trees.

Although (mechanical) software fault tree analysis techniques have the limitation, we can use them more efficiently, if we restrict the target of safety analysis into some important failures, from a domain-specific point of view. In the RPS (Reactor Protection System) in nuclear power plants, the most important event which should be evaluated through safety analysis is '*shutdown*' signals. If an RPS fires the signal, the nuclear reactor should stop immediately. Whenever a nuclear reactor is under dangerous conditions, RPS should fire the signal on time. However, the signal should not be fired when it should not, since shutdown costs about $260,000 a day. To avoid both catastrophe and shutdown lost, safety analysis for RPS should analyze all cases which might lead to the fire of the signal.

This paper proposes a domain-specific safety analysis technique, *NuFTA* for nuclear reactor protection systems. NuFTA generates software fault trees mechanically, whose root nodes (top failures) are *shutdown* signals. It constructs a software fault tree from a formal requirements specification of RPS, written in the NuSCR [22] formal modeling language. A CASE tool *NuSRS* supports specifying NuSCR formal specification, and they had been used to develop KNICS(Korean Nuclear Instrumentation and Control System consortium) RPS [6] for APR-1400 nuclear reactor in Korea [21]. NuFTA starts its analysis from a node in the NuSRS tool. NuFTA then finds all cases which lead to a condition in which the node has a predefined value indicating a shutdown (*e.g. th_X_Trip == 0*) or normal output (*e.g. th_X_Trip == 1*) in the form of fault tree. It analyzes all NuSCR constructs backwards, and also generates a logic expression representing all cases in summary.

Even if the NuFTA is not a software fault tree analysis technique for general purposes, it showed its advantage on analyzing specific failures in the domain of nuclear reactor protection systems. Our former work /citeFTtemplate had proposed conceptual algorithm and templates for mechanical generation of fault tree, but they are insufficient to develop a CASE tool. We refined the templates, constructed algorithm and developed a CASE tool, NuFTA.

The remainder of the paper is organized as follows: Section 2 briefly overviews techniques for mechanical software fault tree analysis. It also introduces the NuSCR formal requirements specification language briefly, and an example of a mechanically generated software fault tree. Section 3 explains templates and algorithms developed to construct a software fault tree from an NuSCR specification mechanically. Section 4 conducts a case study using a requirements specification [4] of the KNICS RPS in Korea. We evaluated performance of

978-0-7695-4454-0/11 $26.00 © 2011 IEEE
DOI 10.1109/SSIRI-C.2011.21

NuFTA through estimating computation time of all relevant nodes in requirements specification. We conclude the paper in Section 5.

II. BACKGROUND

A. Software Fault Tree Analysis

Software fault tree analysis (FTA) analyzes an undesired state of a software system using Boolean logic such as AND ($\wedge$) and OR ($\vee$). Intrinsic features of software have directed many researches into mechanical generation of software fault trees from program source code or specifications, but they contain information only on their sources and cannot show anything beyond that captured in the sources. Mechanically generated fault trees have provided a good starting point of FTA to safety analysis, and widely used in FTA of safety critical systems such as nuclear power plants and satellites.

[8], [9], [11] constructed software fault trees from Ada83/Ada95 programs. They defined fault tree templates for all program constructs in Ada83/Ada95, and constructed a fault tree for a root-node (a top failure) through combining a series of templates and Boolean logic gates. Templates help analysts focus on all possible causes in the constructs. [13], [16] proposed a fault tree analysis technique on Function Block Diagrams (FBDs) [1],one of the most widely used PLC (Programmable Logic Controller) programming languages. It was applied to the main shutdown logic of the KNICS RPS [6] in Korean nuclear power plants.

In case of software specifications, [15] proposed HiP-HOPS (Hierarchically Performed Hazard Origin and Propagation Studies), which generates fault trees from an architectural diagram similar with data flow diagram. It analyzes on only interactions between software components, since it considers a software component as a black-box. [14] extended it to generate fault trees from Matlab-Simulink models, but cannot analyze internal behavior of components neither. [19] proposed a technique generating software fault tree from architecture description language RIDL (Reliability Imbedded Design Language), and developed an analysis tool Galileo [18]. RIDL can describe redundant modules and components from aspect of reliability, but cannot go into details on module and component.

[5] proposed a synthesis method of software fault tree from software requirements specification written in NuSCR [22] formal specification language. The software fault tree reflects requirements on both structure and behavior, and can be used for analyzing safety in the view of structure and behavior both. It provided templates for all constructs in NuSCR, but the templates and synthesis algorithm were not mature to develop a supporting CASE tool. Software fault tree depicted in Fig. 3 was generated from the NuSCR specification presented in Fig. 1, according to the proposal [5]. [12], [17], [20] describe software fault tree generation and analysis techniques developed for State charts [2] and RSML (Requirements State Machine Language) [10] formal requirements specifications.

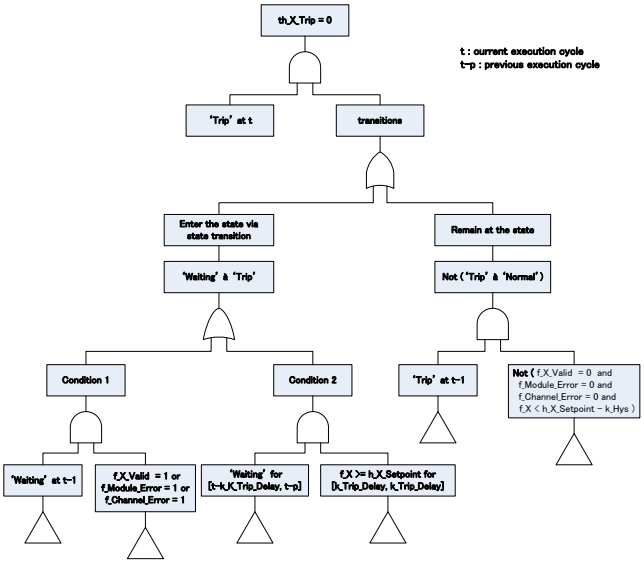
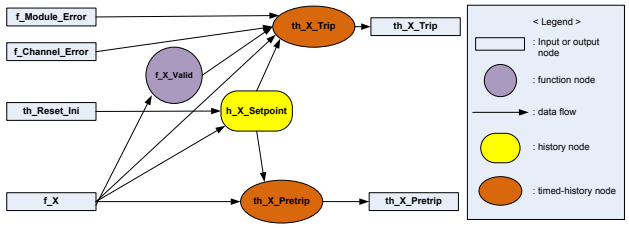


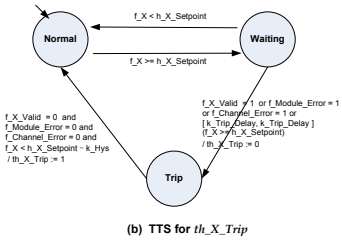
Fig. 1. A software fault tree generated from the NuSCR specification in Fig. 2 (excerpted)

B. NuSCR Formal Specification Language

Fig. 2 describes the NuSCR [22] specification for "manual reset variable set-point rising trip" logic part in the KNICS RPS BP (Bistable Process) [3]. We used a prototype version of requirements specification and simplified it to aid understanding. We could use an official version [4], but we are unable to disclose all details and they are also similar with each other, so we decided to use the prototype version. Although the details are beyond the scope of the paper, this section introduces important features of the NuSCR example which are pertinent to our discussion.



(a) FOD for *g_Manual_Reset_Variable_Set_Point_Rising_Trip*



(b) TTS for *th_X_Trip*

Conditions			
k_X_MIN <= f_X	T	F	-
f_X <= k_X_MAX	T	-	F
Actions			
f_X_Valid := 0	X		
f_X_Valid := 1		X	X

(c) SDT for *f_X_Valid*

Fig. 2. A simplified NuSCR specification for *g_Manual_Reset_Variable_Set_Point_Rising_Trip*

The NuSCR has 4 basic constructs, FOD (Function Overview Diagram), FSM (Finite State Machine), TTS (Timed Transition System) and SDT (Structural Decision Table). Fig. 2(a) describes an FOD for *g_Manual_Reset_Variable_Set_Point_Rising_Trip* logic. Prefix '*g*' means that it is a group of nodes in a hierarchy of FODs. The FOD is composed of four internal nodes and they are all defined individually. The prefixes '*f*', '*h*', and '*th*' denote function variable node, history variable node and timed history variable node respectively. Arrows denote data-flow dependency relation. NuSCR is based on a sequential data-flow.

Fig. 2(b) is a TTS definition for timed history variable node *th_X_Trip* in the FOD. TTS is an FSM extended with time duration constraint [a, b] in transition conditions. It is interpreted as follows: "If condition '*f_X == h_X_Setpoint*' is satisfied in state '*Normal*', then it transits to '*Waiting*' state. In this state, if the condition lasts for *k_Trip_Delay* or one of three input errors (*f_X_Valid, f_Module_Error, f_Channel_Error*) occur, then it fires the trip signal (*th_X_Trip := 0*). In the state '*Trip*' if the trip condition is canceled, then it comes back to the state '*Normal*' and the output becomes '1'. The TTS expression [*k_Trip_Delay*,*k_Trip_Delay*] means that the condition has to remain true for *k_Trip_Delay* time units. We skipped the explanation on the history variable node *h_X_Setpoint*, for it is a subset of the timed history variable node.

Fig. 2(c) defines an SDT for function variable node *f_X_Valid*. It is interpreted as follows: "If the value of *f_X* is between *k_X_MIN* and *k_X_MAX*, the output value of *f_X_Valid* is 0, which means that it is a normal case. Otherwise, *f_X_Valid* is 1". The NuSCR recommends multiple correlated condition statements per row, e.g. '$k\_X\_MIN \leq f\_X \leq k\_X\_MAX$'. It helps the NuSCR resolve table-size explosion problems [22], and also can increase readability of SDTs. Fig. 3 is a screen-dump of NuSCR supporting tool *NuSRS*, showing an FOD of NuSCR specification *g_VAR_OVER_PWR* [4] for KNICS RPS BP.

C. An Example of Mechanical Software Fault Tree Analysis

Our former work [5] proposed templates for mechanical generation of fault trees and a prototype of fault tree construction algorithm. According to the guide, we constructed a software fault tree (depicted in Fig. 1) from the NuSCR specification (described in Fig. 2) in hand. The *g_Manual_Reset_Variable_Set_Point_Rising_Trip* logic in Fig. 2(a) fires a shutdown signal (*th_X_Trip := 0*) in Fig. 2(b) when the nuclear reactor is in danger from aspect of the input variable *f_X*. It is one of 18 shutdown logics of nuclear reactors, in which safety analysis is essential to safety demonstration.

Fig. 3 is a fault tree when the shutdown signal is fired. Therefore, the root-failure node of the fault tree is the case that the shutdown is fired (*th_X_Trip := 0*), and we constructed the fault tree through combining a series of templates. The fault tree analysis shows all possible cases that the shutdown signal is fired. It analyzes the cases in two ways, 'entering the state' and 'remaining at the state'. We assumed that the software fault tree has only 2 system execution cycles. An assisting tool *NuFTA* which we developed and proposed in this paper, currently also supports 2 system execution cycles.

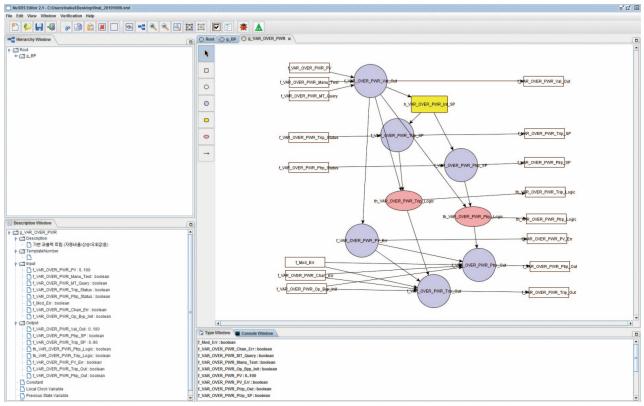


Fig. 3. A screen-dump of NuSCR specification of KNICS RPS BP in NuSRS tool-set (ver. 2.1)

III. NuFTA

NuFTA is a CASE tool for mechanically constructing software fault trees from NuSCR formal requirements specification. They all have aimed at plant protection systems in Korean nuclear power plants, as we had introduced in [21]. NuFTA does not aim at a general-purpose tool but a domain-specific one, which analyzing important events only in the domain. In our case, it does on 'shutdown' signal only. By restricting its target into the signal, NuFTA can construct a software fault tree in feasible time efficiently. We refined the templates and construction algorithm which our former work [5] had proposed, and developed the CASE tool NuFTA. The NuFTA is now embedded in NuSRS, a supporting tool for NuSCR formal specification and verification. Fig. 4 below represents a screen-dump of NuFTA, describing a software fault tree which constructed from the node in a dashed rectangle in Fig. 2. This section first overviews the NuFTA's software fault tree construction, and explains the templates and algorithms in detail.

A. Overview

The following overviews the construction of software fault tree using NuFTA:

1) An analyst selects a node generating shutdown signal in NuSRS.
2) NuFTA analyzes backwardly causes of the signal throughout all connected nodes in FODs.
3) Using fault tree template for all nodes of NuSCR, NuFTA constructs a software fault tree for the node as Fig. 4 shows.
4) NuFTA produces a logic expression representing the software fault tree generated, if it is feasible.

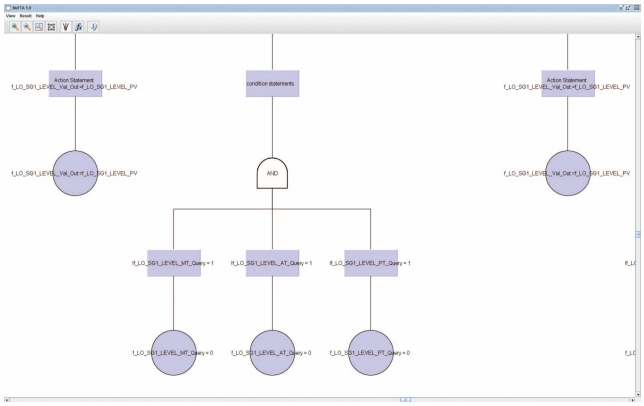


Fig. 4. A screen-dump of NuFTA (ver. 1.0)

The first thing which a safety analyst performs is to select an important node generating shutdown signal in a NuSCR specification. He (She) selects a node and assigns to it a value which means a shutdown condition. For example, for the selected node in Fig. 3, output variable *f_VAR_OVER_PWR_Trip_Out* should have zero (0) to signal a shutdown. Then the NuFTA starts analyzing backwardly all combinations of conditions causing the result. It backtracks the FOD to input nodes, since NuSCR is a sequential data-flow based language. Whenever it meets nodes such as function, history and timed history variable nodes, it uses appropriate fault tree templates. For history and timed history variable nodes, NuFTA transforms them into expanded formats in order to apply the templates into them. A predefined execution step used to expanding them. Details are introduced in Section 3.3. The NuFTA finally produces a logic expression, minimal cut-set representing all combinations of the conditions.

B. Fault Tree Templates

Constructs of the NuSCR formal specification language have different definitions respectively. According to the definitions, our former work [5] had proposed three types of templates. We refined them more precisely for developing a CASE tool, NuFTA. Function variable node is defined as an SDT which consists of condition statements and action statements. The outputs are described in action statements and the causes are described in condition statements. Fig. 5(a) shows a template for SDTs whose value of output is *value*. The template describes the relation between action statements and an output of an SDT. The leftmost branch describes that the value of output and a value of RHS (Right Hand Side) of an action statement of are the same. The middle branch describes that the value of output is included or shared in the range of RHS of an action statement. And the rightmost branch describes that common range of the value of output and RHS of action statements cannot be founded or other exceptional cases. This branch should also be included in a fault tree, because a fault tree should include every cause to generate failures. One of the three branches are selected by relation of value of an output and action statements, so they are described in dotted box. For analyze action statements, the template classify RHS of action statements as Fig. 5(b) shows. The leftmost node describes that RHS of action statement specified with values of other NuSCR node's outputs. In this case, fault trees for other nodes which have the output values should be connected. Middle node describes that a function variable nodes RHS of action statement specified with the function variable node's own an output value of previous cycle. In a fault tree, analysis for the value of previous cycle should be added. The rightmost node describes that RHS of action statement specified with only constants.

Condition statements that corresponding each output statement are specified in lower nodes. If the condition statements are composed with a few conditions connected with logical connector, the template uses AND gate and OR gate in order to analyze conditions respectively.

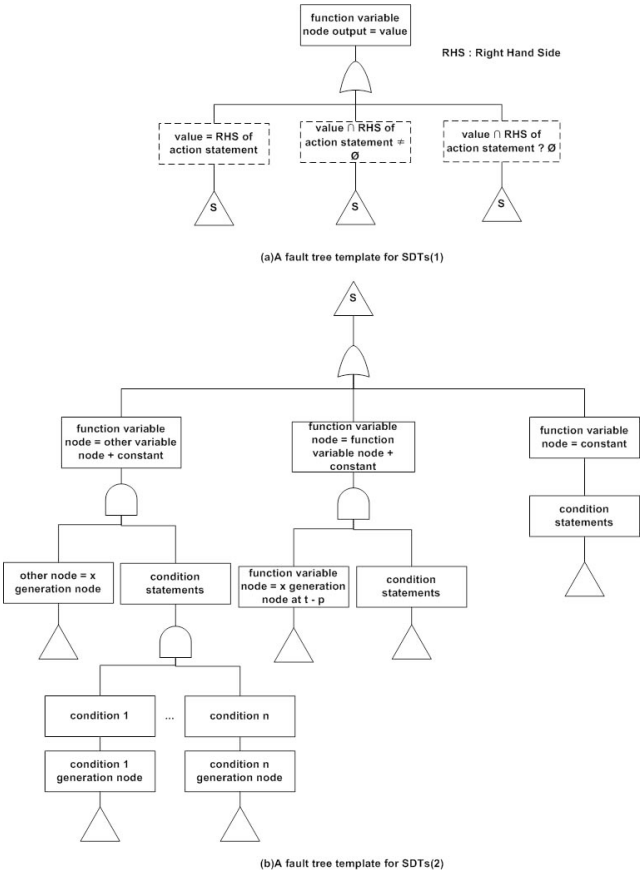


Fig. 5. A fault tree template for SDTs

History variable node is defined as an FSM which consists of states and transitions. In order to analyze conditions of an output, NuFTA need to search states which generate the output. But it is difficult to find possible outputs of states because states on FSM dont have form for specifying outputs. Therefore, NuFTA unfolds the original FSM to other format,

an annotated FSM for algorithmic search. In the annotated FSM, states have own output values and names of original states. Transitions are reordered by relations of annotated states. A template for history variable node uses annotated FSM as Fig. 6. The first branch describes that the output value is the same as the RHS of the annotated assignment statement. The second branch describes that the output value is shared with RHS of the annotated assignment statement. The third branch describes that the relationship between values is not decided. The last one describes that the annotated state which has original state is same as the output state (history variable node = state).

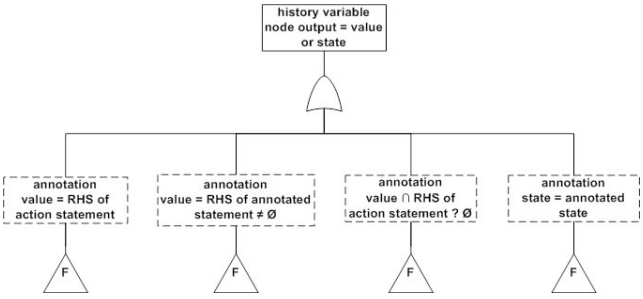


Fig. 6. A fault tree template for FSMs(1)

Fig.7 shows a fault tree extends by using a template in Fig.6. An annotated FSM can have a few annotated states that generating same output value. The annotated states are connected with OR gate. The template classifies causes of output value with two cases. One is enter the state via state transition and the other is remain at the state. The former means that one condition of current states ingoing transitions is satisfied then FSM enters the current state via ingoing transition. The latter is classified two cases as Fig. 8. When an outgoing transition of a state is disabled, an FSM remains at the state. Therefore, the template describes that causes of the first case are logical inverse of transitions condition. In the second case, outgoing transition of current state is enabled but the transitions target and source state are same. In other words, the transition of current state is self-cycling. The template describes that second case is occurred when the condition of the state is satisfied.

Timed history variable node is defined as TTS. TTS is very similar with FSM, excepts TTS can have time constraint condition on transition. Therefore, a fault tree template for TTSs is similar with FSMs'. In order to generate fault tree from a TTS, a template for TTSs uses same approach as a template for FSMs shown as Fig. 9. In Fig. 10 and Fig. 11, the template for TTSs describes how analyzes conditions of transitions of current state in TTS. The template for TTSs defines a transition which has timed constraint as timed transition and the other transition as untimed transition. Analysis of untimed transitions is described completely same as FSM's. For analyzing condition of timed transition, the template focuses on how long the condition is satisfied and

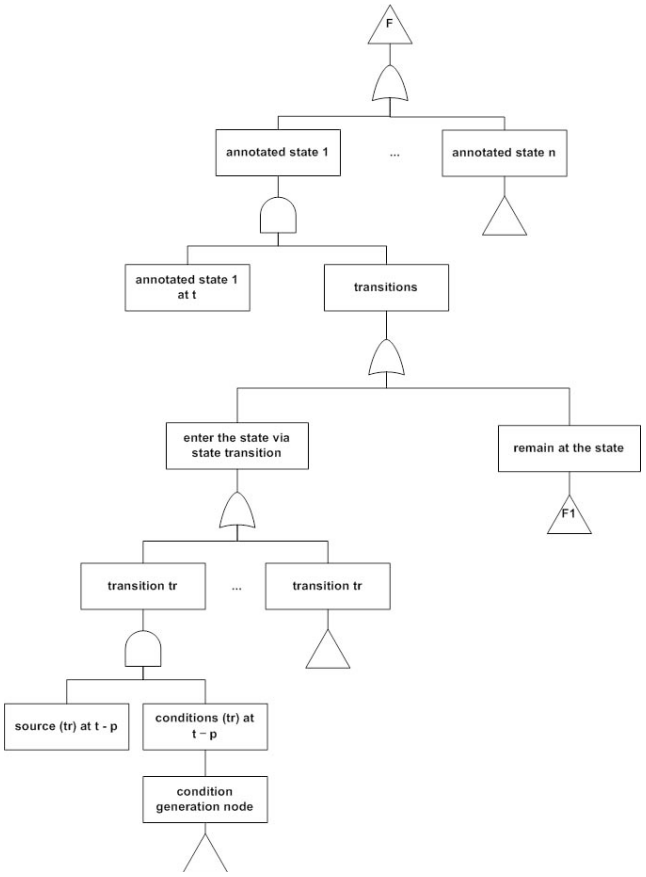


Fig. 7. A fault tree template for FSMs(2)

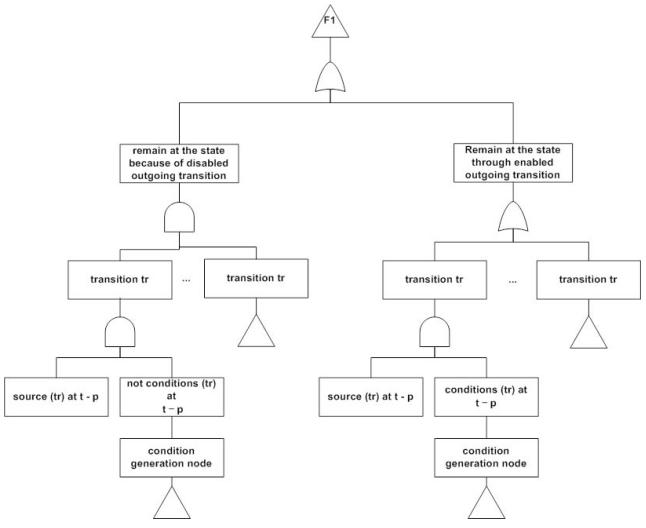


Fig. 8. A fault tree template for FSMs(3)

stayed at previous state. When a state of TTS is on the source state of current state's ingoing transition *k_delay* unit times before and condition of the transition is satisfied for *k_delay*

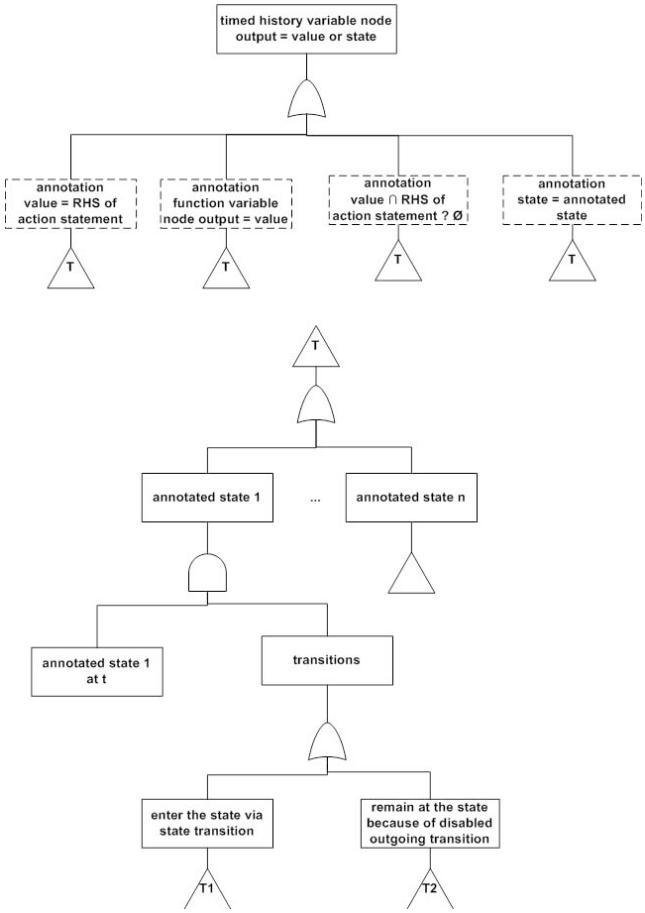


Fig. 9. A fault tree template for TTSs(1)

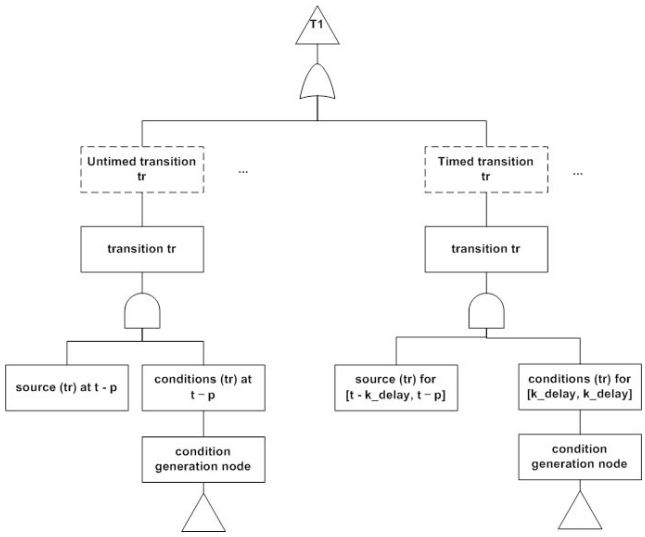


Fig. 10. A fault tree template for TTSs(2)

unit times then the timed transition is enabled.

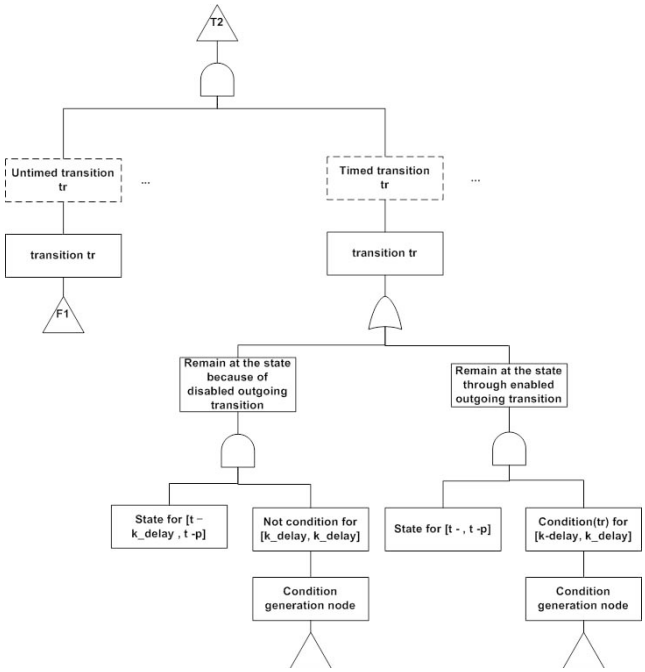


Fig. 11. A fault tree template for TTSs(3)

C. Construction Algorithm

NuFTA analyze an NuSCR specification using the fault tree templates for each NuSCR variable node. Construction of a software fault tree using NuFTA starts when an analyst enters a failure as an input to NuFTA. NuFTA finds a variable node in an NuSCR specification that generates the failure. The remainder sequence of analysis is divided by type of variable node. If the variable node is defined with SDT, NuFTA finds an output value at action statements and conditions of the output value at condition statements. If the variable node is defined with FSM or TTS, NuFTA unfolds the variable node with annotated FSM or annotated TTS. Because states in FSM and TTS have no own output value in them, algorithmic construction needs another format which has states' notified output value. Then NuFTA finds an output value at annotated states and that's conditions at annotated transitions. When NuFTA find NuSCR variable node in conditions of an output, NuFTA attaches a new fault tree for the variable node to the cause node. If all leaf nodes of fault tree are input nodes or states of NuSCR specification, NuFTA finishes the analysis on the failure. The specific algorithm are as follows.

Function create Fault Tree is a function that constructs fault tree data. According type of NuSCR variable node, the function calls *buildSDTTree* and *buildFSMTree*. They construct fault trees using each template and return the trees to *Function create Fault Tree*. A node of fault tree that contains cause of failure called *cause node*. Causes of a failure can be value of NuSCR input node or variable node. If the *cause node* has variable node, *Function create Fault Tree* recursively call itself and attach a fault tree for the variable node.

```
Function create Fault Tree
    U is a set of candidate fault node
    s is a root node of fault tree
    U:={s}

    while(U != {})
        u is element of U
        k is type of NuSCR variable node of u
        if(k is type of SDT)
            connect(u, buildSDTTree(u))
        else if(k is FSM)
            a := annotate_FSM(FSM of u)
            connect(u, buildFSMTree(a))
            connect(cause node, u)

        T is set of variable nodes in causes of failure
        U :=(U-{u})∪T
```

```
Function annotate_FSM
    Input:FSM
    Output:annotated FSM

    ss is source state
    vs is value of source state
    ts is target state
    vt is value of target state

    T is set of transition
    closed is set of transition
    Q is stack of (state, output value of state)

    is :=initial state
    n :=output value of initial state
    S :=(is, n) ∪ S

    push(Q, (is,n))

    while(Q !={})
        (ss,ns) := pop(Q)
        for each transition t from ss
            ts := target state of t
            if(t has assignment)
                vt := assigned value after processing t
            else
                vt := n

            if(v > maxvalue)
                exit for
            if((ts,vt) ∉ closed)
                push(Q,(ts,vt))
                T := ((ss,ns),(ts,vt)) cup T
                closed := (ts, vt) cup closed
    annotated FSM is combined by T and states of FSM
```

Constructing annotated FSM is necessary for analyzing FSM with algorithmic way. Our former work [5] suggested annotated FSM, but method for constructing annotated FSM is not suggested. We constructed annotated FSM using DFS (Depth-First Search) algorithm. Constructing annotated FSM stars with get output value of initial state. Then, NuFTA finds target states of outgoing transitions of initial state then insert the state to a stack. NuFTA pop one element of the stack and assign output value to the element and recursively finds target state as did at initial state. When stack is empty, NuFTA stops constructing annotated FSM. Transitions have can assignment on output value when the conditions of transition are satisfied. So the output value of transitions target state is assigned by the assignment. If a transition has no assignment, output value of the transitions target state assigned by output value of transitions source state.

IV. EXPERIMENTAL RESULT

We constructed experiments for measuring cost time according to complexity of all FODs in an NuSCR specification about shutdown signal (*trip_out = true*) and pre-shutdown signal (*pretrip_out = true*). FODs composed a few NuSCR variable nodes and specify shutdown logic in an RPS specification. The variable nodes in one FOD use a same variable that represents measurement value of RPS. The variable is called process variable. Complexity of shutdown signal defined with a process variables range and causes that generate failure value on an NuSCR specification. States of annotated automata (FSM or TTS) have own output value and name of original state. Therefore, when NuFTA unfolds FSM and TTS to annotated automata, number of annotated states can be increased proportionally to range of process variable. Before measure cost time of all FODs, we constructed first experiment for measuring time of analysis according to range of process variable. Target is a simple FOD, *g_LO_SG1_LEVEL*. This FOD is composed two SDTs and one TTS which has three states and four transitions. Range of process variable in *g_LO_SG1_LEVEL* is '0∼100', but we adjust that for experiments. Table 1 shows result of first experiment.

TABLE I
TIME COST ACCORDING TO PROCESS RANGE OF VARIALBE OF *g_LO_SG1_LEVEL*

Range of a process variable	*trip out(ms)*	*pretrip out(ms)*
0∼10	96	93
0∼100	138	109
0∼1000	351	170
0∼10000	3377	453

As we assumed, result shows that the cost time is increases according to range of process variable. We constructed second experiment that measures cost time of analysis for all FODs. If the cost time is over 5 minutes, we stopped the analysis. Table 2 shows the result.

TABLE II
ANALYSIS TIME OF EACH FOD

Name of FOD	Range of a process variable	Analysis time of *trip out(ms)*	Analysis time of *pretrip out(ms)*
g_VAR_OVER_PWR	0∼100	-	-
g_LO_SG1_LEVEL	0∼100	138	109
g_HI_LOG_POWER	0∼100	92	142
g_LO_PZR_PRESS	0∼100	205	197
g_SG1_LO_FLOW	0∼100	111	108
g_HI_LOCAL_POWER	0∼2	8	4

Except one FOD, *g_VAR_OVER_PWR*, NuFTA shows good performance of analysis for NuSCR specification. The FOD is much more complex than other FODs. It composed with 6

SDTs, 2 TTSs, and one FSM which has 4 states and 16 transitions. With experiments for small range of a process variable on *g_VAR_OVER_PWR*, we concluded that constructed fault tree for *g_VAR_OVER_PWR* correctly. But because of lack of optimization of source code and data structure, NuFTA need large needs long cost time and large memory for analyzing the FOD.

V. CONCLUSION AND FUTURE WORK

We proposed a domain-specific safety analysis technique *NuFTA* for the domain of plant protection systems in Korean nuclear power plants. We restricted the application domain of safety analysis into specific types of critical failures, 'shutdown' signal in nuclear power plants. It made possible to automate a large part of the analysis and also improve its quality too. The NuFTA mechanically constructs a software fault tree of nuclear reactor protection systems specified formally with NuSCR. Within the domain specific restrictions, NuFTA can construct software fault trees mechanically and aid safety experts' analysis efficiently. Experiment result showed that NuFTA can construct software fault trees for important signals but needs optimizations of source code and data structure for more efficient mechanical construction. Furthermore, we plan to expand system cycles of the software fault trees according to user's request.

ACKNOWLEDGMENT

This research was supported by the MKE(The Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency (NIPA-2011-(C1090-1131-0008), NIPA-2011-C1090-1131-0003), under the Development of Performance Improvement Technology for Engineering Tool of Safety PLC(Programmable Logic Controller) program supervised by the KETEP(Korea Institute of Energy Technology Evaluation And Planning) (KETEP-2010-T1001-01038), Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2010-0002566) and IT R&D Program of MKE/KEIT [10035708, "The Development of CPS(Cyber-Physical Systems) Core Technologies for High Confidential Autonomic Control Software"]

REFERENCES

[1] International. Electrotechnical Commission (IEC). International standard for programmable controllers: Programming languages, 1993. part 3.

[2] D. Harel. On visual formalism. *Communication of the ACM*, 31(5):514-530, 1986.

[3] KAERI(Korea Atomic Energy Rearch Institute). SRS for Reactor Protection System. *KNICS-RPS-SRS101 Rev.00*, 2003.

[4] KAERI(Korea Atomic Energy Rearch Institute). Fromal SRS for Reactor Protection System. *KNICS-RPS-SVR131-01 Rev.00*, 2005.

[5] T. Kim, J. Yoo, and S. Cha. A Synthesis Method of Software FaultTree from NuSCR Formal Specification using Templates. *Journal of the Korean Institute of Information Scientists and Engineers - Software and Application (in Korean)*, 32(12):1178-1191, 2005.

[6] KNICS. http://www.knics.re.kr/english/eindex.html.

[7] N. G. Leveson. SAFEWARE, System safety and Computers. *Addison Wesley*, 1995.

[8] N. G. Leveson, S. Cha, and T. Shimeall. Safety verification of ada programs using software fault trees. *IEEE Software*, 8(4):48-59, 1991.

[9] N. G. Leveson and P. Harvey. Analyzing software safety. *IEEE Software*, 9(5):569579, 1983.

[10] N. G. Leveson, M. P. E. Heimdahl, H. Hildreth, and J. D. Reese. Requirements specification for process-control systems. *IEEE Transactions on Software Engineering*, 20(9):684-707,1994.

[11] S.-Y. Min, Y. kyu Jang, S. Cha, Y.-R. Kwon and D. Bae. Safety verification of Ada95 programs using software fault trees. *In Computer Safety, Reliability and Security (SAFECOMP) - LNCS 1698/1999*, pages 226-238, 1999.

[12] R. Mojdehbakhsh, S. Subramanian, R. Vishnuvajjala, W. Tsai, and L. Elliott. A process for software requirements safety analysis.*In International Symposium on Software Reliability Engineering*, pages 45-54, 1994.

[13] Y. Oh, J. Yoo, S. Cha, and H. S. Son. Software Safety Analysis of Function Block Diagrams using Fault Trees. *Reliability Engineering and System Safety*, 88(3):215-228, 2005.

[14] Y. Papadopoulos and M. Maruhn. Model-based synthesis of fault trees from matlab-simulink models. *In the 2001 International Conference on Dependable Systems and Networks (DSN01)*, pages 7782, 2001.

[15] Y. Papadopoulos, J. McDermid, R. Sasse, and G. Heiner. Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. *Reliability Engineering and System Safety*, 71(3):229247, 2001.

[16] G.-Y. Park, K. Y. Koh, E. Jee, and P. H. Seong. Fault tree analysis of KNICS RPS software. *Nuclear Engineering and Technology*, 40(5):397-408, 2008.

[17] V. Ratan, K. Partridge, J. Reese, and N. Leveson. Safety analysis tools for requirements specifications. *In the 7thCOMPASS Workshop*, pages 149160, 1996.

[18] K. Sullivan, J. Dugan, and C. Coppit. The Galileo fault tree analysis tool. *In the 29th Annual International Symposium on Fault-Tolerant Computing*, pages 232235, 1999.

[19] K. Vemuri, J. Dugan, and K. Sullivan. Automatic synthesis of fault trees for computer-based systems. *IEEE Transactions on Reliability*, 48(4):394-402, 1999.

[20] J. Yoo, S. Cha, and H. S. Son. Automatic Generation of Goal-Treefrom Statecharts Requirements Specification. *America Nuclear Society Transactions*, 88:37-38, 2003.

[21] J. Yoo, E. Jee, and S. S. Cha. Formal Modeling and Verification of Safety-Critical Software. *IEEE Software*, 26(3):4249, May/June 2009.

[22] J. Yoo, T. Kim, S. Cha, J.-S. Lee, and H. S. Son. A Formal Software Requirements Specification Method for Digital Nuclear Plants Protection Systems. *Journal of Systems and Software*, 74(1):7383, 2005.