SSIRI 2011, Jeju island

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

Sanghyun Yoon
Dependable Software Laboratory
Konkuk University, Korea

June 28, 2011

Contents

- Introduction
- Background
  - Software Fault Tree Analysis
  - A Formal Software Requirement Specification method, NuSCR
- Our suggestion : NuFTA
  - Overview of NuFTA
  - Software fault tree templates for NuSCR nodes
  - Experimental Result
- Conclusion & Future Work

Introduction(1)

- Failures of safety-critical systems incur catastrophic disaster
  - The systems require rigorous quality demonstration.

- Safety analysis tries to assure the systems' safety through performing various safety analysis techniques
  - FTA (Fault Tree Analysis), FMEA (Failure Mode and Effect Analysis), HAZOP (Hazard and Operability study).

- Safety experts apply the techniques manually
  - Quality and correctness of the analysis result totally depends on the knowledge and experience of the experts.

Introduction(2)

- Many safety analysis techniques focus on mechanical generation of software fault tree.

- If we restrict the application domain of safety analysis into some critical failures, we can use the safety analysis techniques more efficiently.

- Our target domain was KNICS(Korea Nuclear Instrumentation and Control System) RPS(Reactor Protection System).

- Prototype version of KNICS RPS is specified with NuSCR.

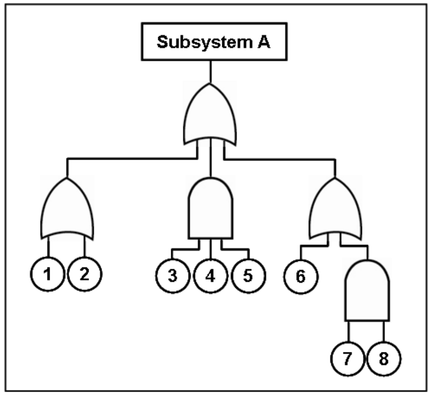
- We propose a CASE tool, *NuFTA*
  - NuFTA is a CASE tool for digital nuclear RPS.
  - NuFTA generates software fault tree mechanically from an *NuSCR* specification.

BACKGROUND

Software Fault Tree Analysis

- Software Fault Tree Analysis(SFTA)
  - Target of SFTA is software of a system.
  - Deductive and top-down method of analyzing system.
  - Identifying all of the associated elements using boolean gate that could cause top event(failure) to occur.

- Minimal cut-set
  - A basic set of events that can cause failure.
  - Safety experts use minimal cut-set to obtain an estimate of reliability for complex fault tree.



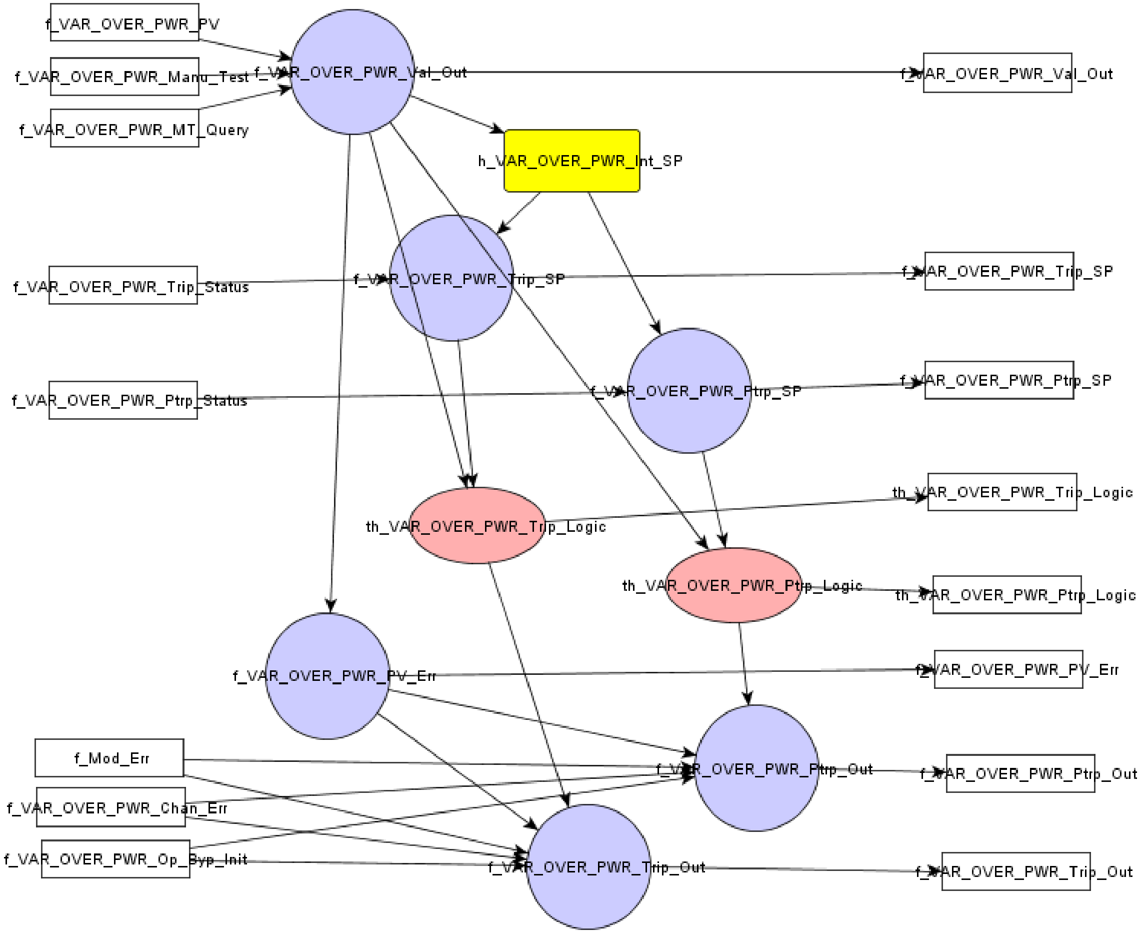
<A fault tree for subsystem A>

Subsystem A =
(1 | 2) | (3 & 4 & 5) | (6 | (7 & 8))

<Minimal cut-set of subsystem A>

A Formal Software Requirement Specification method - NuSCR(1)

- Extended SCR (Software Cost Reduction, Heninger, 1980) for RPS
- Sequential System.
- An FOD(Function Overview Diagram) is composed of variable nodes.
- Variable nodes
  - Function variable node(SDT), prefix : *f*
  - History variable node(FSM), prefix : *h*
  - Timed-history variable node(TTS), prefix : *th*



FOD for *g_VAR_OVER_SP*

□ : Input or Output variable node

🟨 : History variable node(FSM)

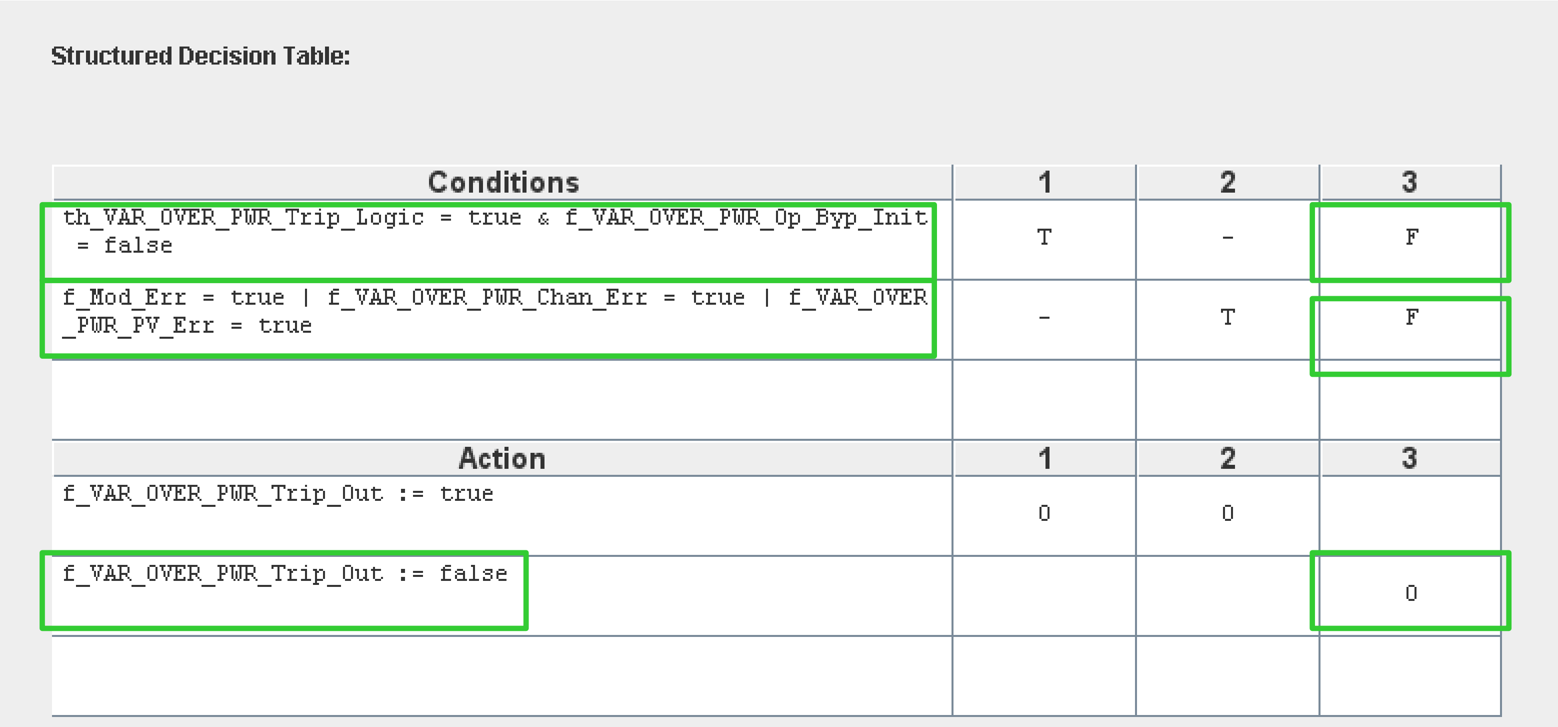
○ : Function variable node(SDT)

⬭ : Timed history variable node (TTS)

→ : Data flow

A Formal Software Requirement Specification method - NuSCR(2)

- Function variable node is defined with structured decision table(SDT).
- SDT is composed of condition statements and action statements.



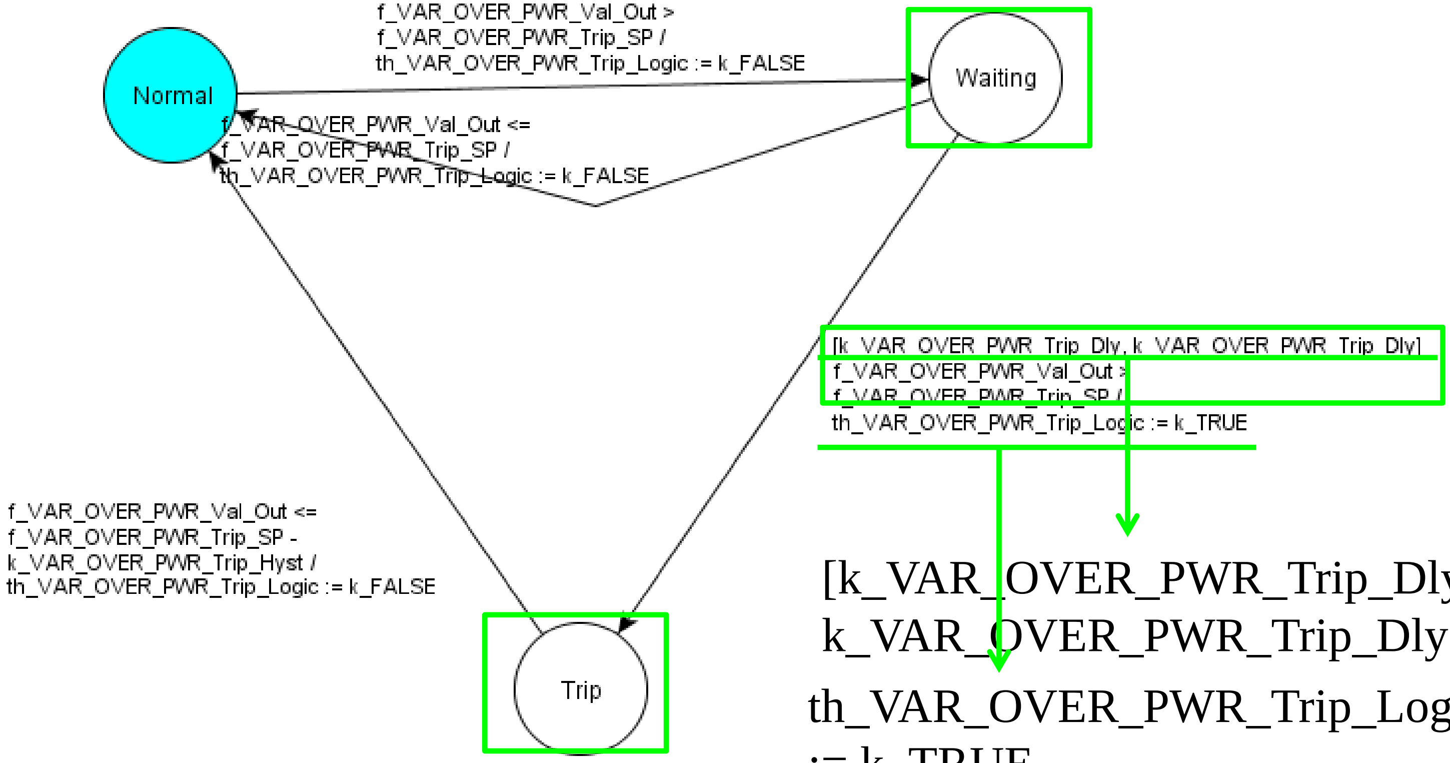
Structured Decision Table:

Conditions	1	2	3
th_VAR_OVER_PWR_Trip_Logic = true & f_VAR_OVER_PWR_Op_Byp_Init = false	T	-	F
f_Mod_Err = true | f_VAR_OVER_PWR_Chan_Err = true | f_VAR_OVER_PWR_PV_Err = true	-	T	F

Action	1	2	3
f_VAR_OVER_PWR_Trip_Out := true	O	O	
f_VAR_OVER_PWR_Trip_Out := false			O

<A definition of function variable node(Structured Decision Table)>

A Formal Software Requirement Specification method - NuSCR(3)


f_VAR_OVER_PWR_Val_Out >
f_VAR_OVER_PWR_Trip_SP /
th_VAR_OVER_PWR_Trip_Logic := k_FALSE
Normal
Waiting
f_VAR_OVER_PWR_Val_Out <=
f_VAR_OVER_PWR_Trip_SP /
th_VAR_OVER_PWR_Trip_Logic := k_FALSE
[k_VAR_OVER_PWR_Trip_Dly, k_VAR_OVER_PWR_Trip_Dly]
f_VAR_OVER_PWR_Val_Out >
f_VAR_OVER_PWR_Trip_SP /
th_VAR_OVER_PWR_Trip_Logic := k_TRUE
f_VAR_OVER_PWR_Val_Out <=
f_VAR_OVER_PWR_Trip_SP -
k_VAR_OVER_PWR_Trip_Hyst /
th_VAR_OVER_PWR_Trip_Logic := k_FALSE
Trip

[k_VAR_OVER_PWR_Trip_Dly,
k_VAR_OVER_PWR_Trip_Dly]

th_VAR_OVER_PWR_Trip_Logic
:= k_TRUE

<Timed-history variable node(Timed Transition System)>

NuFTA

Overview of NuFTA

- Purpose
  - Mechanically generates a software fault tree for analysts.
  - Root node of SFT : trip/pre-trip(shut-down) signal
  - Analysis result : graphical fault tree, logical expression

- Software fault tree constructing process using NuFTA
  1) Analyst selects a node generating shutdown signal in NuSRS (NuSCR supporting tool).
  2) The NuFTA analyzes backwardly causes of the signal throughout all connected nodes in an FOD.
  3) Using fault tree templates for NuSCR nodes, the NuFTA constructs a software fault tree for the node.
  4) The NuFTA produces a logical expression(minimal cut-set) representing the generated software fault tree.

Software fault tree templates for NuSCR nodes(1)

- T. Kim suggested templates for NuSCR nodes in A Synthesis Method of Software Fault Tree from NuSCR Formal Specification using Templates(2005).

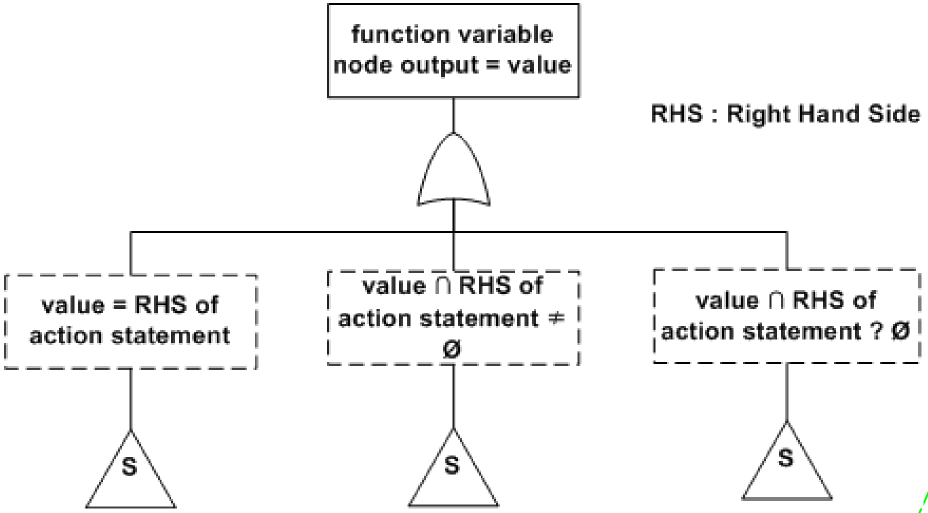
- We modified templates and used for developing NuFTA.

- NuFTA uses software fault tree templates for analyzing variable nodes of NuSCR specifications.

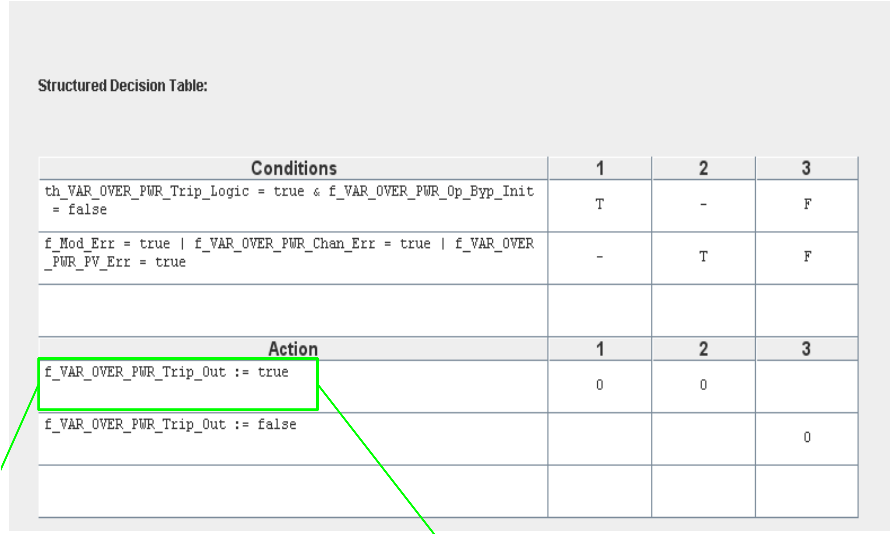
- For analyzing NuSCR nodes, the templates classifies
  - Relational operator of action/assign statement
  - Definition of right hand side of action/assign statement

Software fault tree templates for NuSCR nodes(2)

- This part of SDT template classifies relational operator of action statement.


function variable node output = value
RHS : Right Hand Side
value = RHS of action statement
value ∩ RHS of action statement ≠ Ø
value ∩ RHS of action statement ? Ø
S
S
S

<A template for SDT(1)>

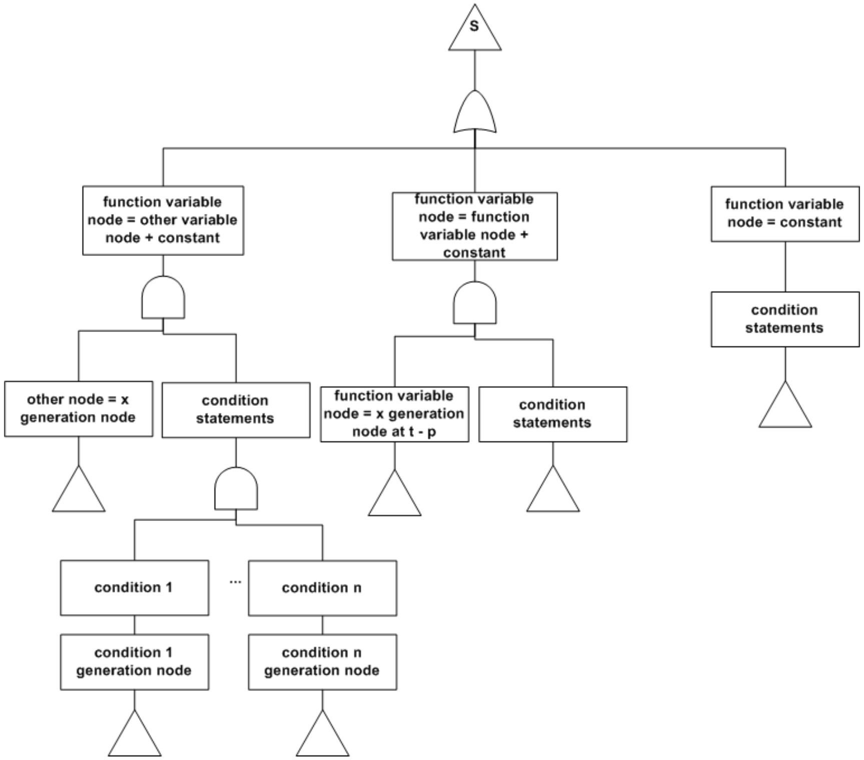


Structured Decision Table:

Conditions	1	2	3
th_VAR_OVER_PWR_Trip_Logic = true & f_VAR_OVER_PWR_Op_Byp_Init = false	T	-	F
f_Mod_Err = true | f_VAR_OVER_PWR_Chan_Err = true | f_VAR_OVER_PWR_PV_Err = true	-	T	F
Action	**1**	**2**	**3**
f_VAR_OVER_PWR_Trip_Out := true	0	0	
f_VAR_OVER_PWR_Trip_Out := false			0

f_VAR_OVER_Trip_Out := true

Software fault tree templates for NuSCR nodes(3)



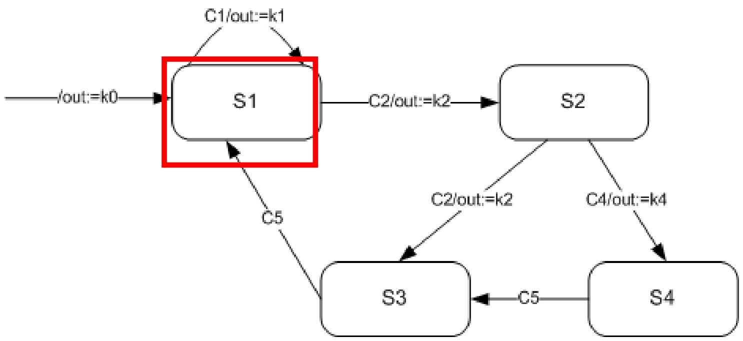
S
function variable node = other variable node + constant
function variable node = function variable node + constant
function variable node = constant
other node = x generation node
condition statements
function variable node = x generation node at t - p
condition statements
condition statements
condition 1
...
condition n
condition 1 generation node
condition n generation node

<A template for SDT(2)>

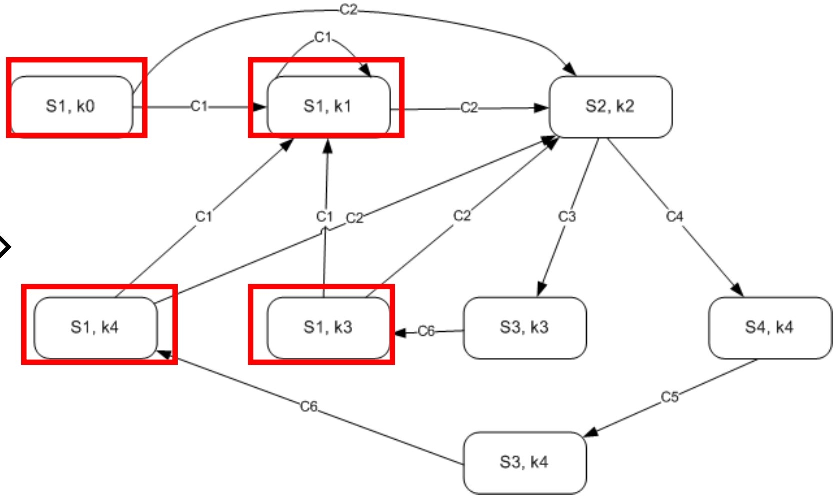
- This part of SDT template classifies definition of RHS of action statement.
- function variable node = constant
  - *e. g. f_X = 1*
- function variable node = other variable node + constant
  - *e.g. f_X = th_Trip_Logic + 1*
  - NuFTA additionally attaches a sub-tree for output value of *th_Trip_Logic*
- function variable node = function variable node + constant
  - RHS has output value of previous cycle
  - *e.g. f_X = f_X + 1*
  - NuFTA additionally attaches a sub-tree for output value of *f_X* on previous cycle.

Annotated automata

- History and timed-history variable are defined with automata
  - Output values of automata are not specified on states
  - We need to specify output values on states for algorithmic analysis.
- Our suggestion: Annotated automata
  - Unfolded automata whose states specified own output value.
  - NuFTA unfolds automata then analyze the annotated automata.



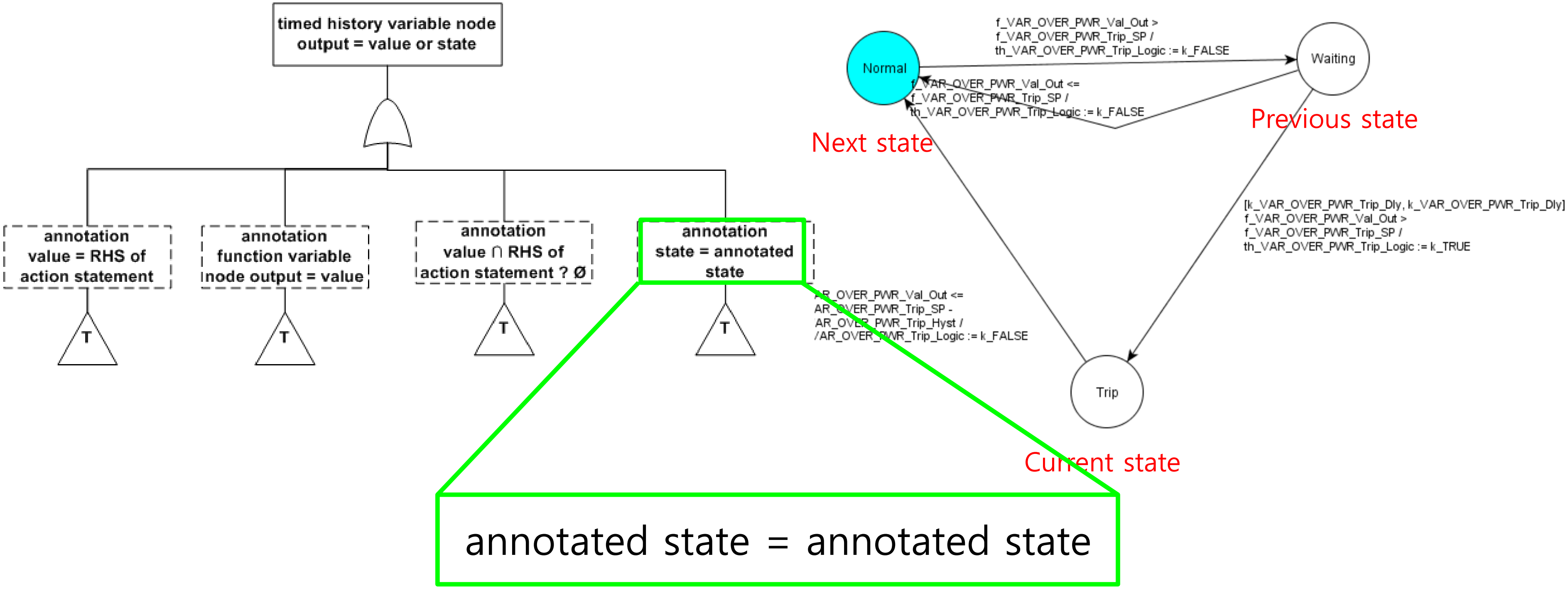
Original automata



Annotated automata

Software fault tree templates for NuSCR nodes(4)

- A template for TTS(1)



timed history variable node output = value or state

annotation value = RHS of action statement

annotation function variable node output = value

annotation value ∩ RHS of action statement ? Ø

annotation state = annotated state

T

T

T

T

annotated state = annotated state

Normal

Waiting

Trip

f_VAR_OVER_PWR_Val_Out > f_VAR_OVER_PWR_Trip_SP / th_VAR_OVER_PWR_Trip_Logic := k_FALSE

f_VAR_OVER_PWR_Val_Out <= f_VAR_OVER_PWR_Trip_SP / th_VAR_OVER_PWR_Trip_Logic := k_FALSE

[k_VAR_OVER_PWR_Trip_Dly, k_VAR_OVER_PWR_Trip_Dly] f_VAR_OVER_PWR_Val_Out > f_VAR_OVER_PWR_Trip_SP / th_VAR_OVER_PWR_Trip_Logic := k_TRUE

AR_OVER_PWR_Val_Out <= AR_OVER_PWR_Trip_SP - AR_OVER_PWR_Trip_Hyst / /AR_OVER_PWR_Trip_Logic := k_FALSE

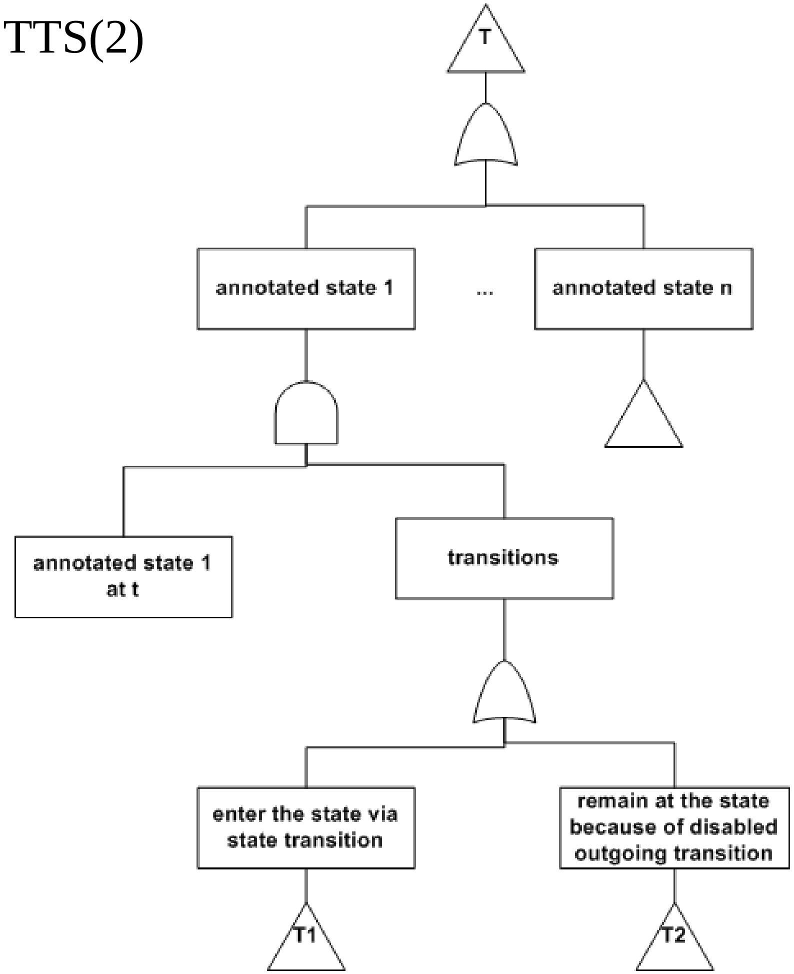
Next state

Previous state

Current state

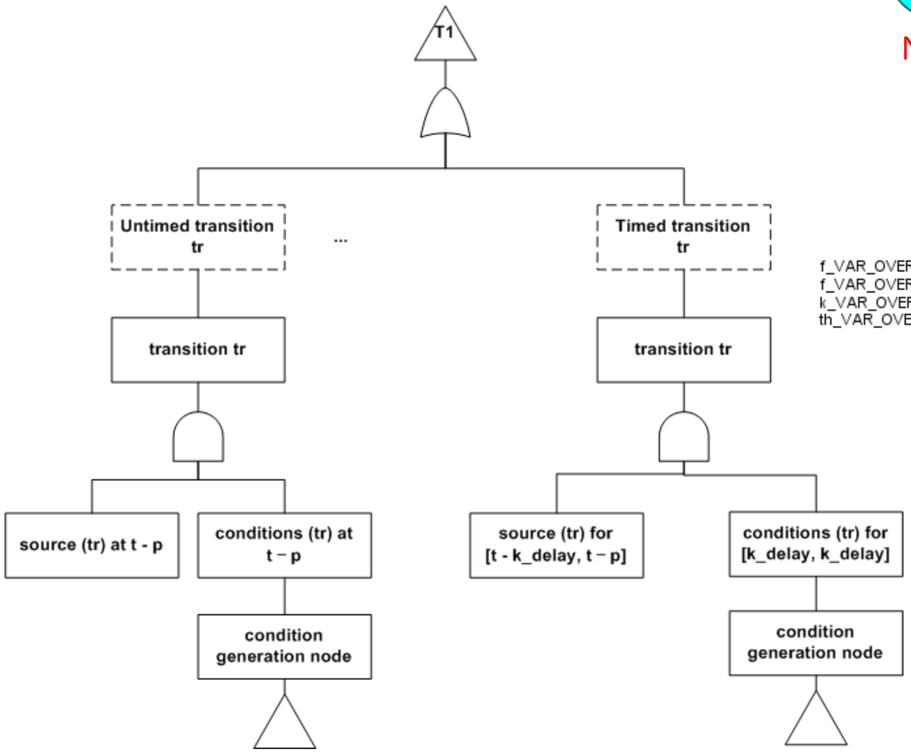
Software fault tree templates for NuSCR nodes(5)

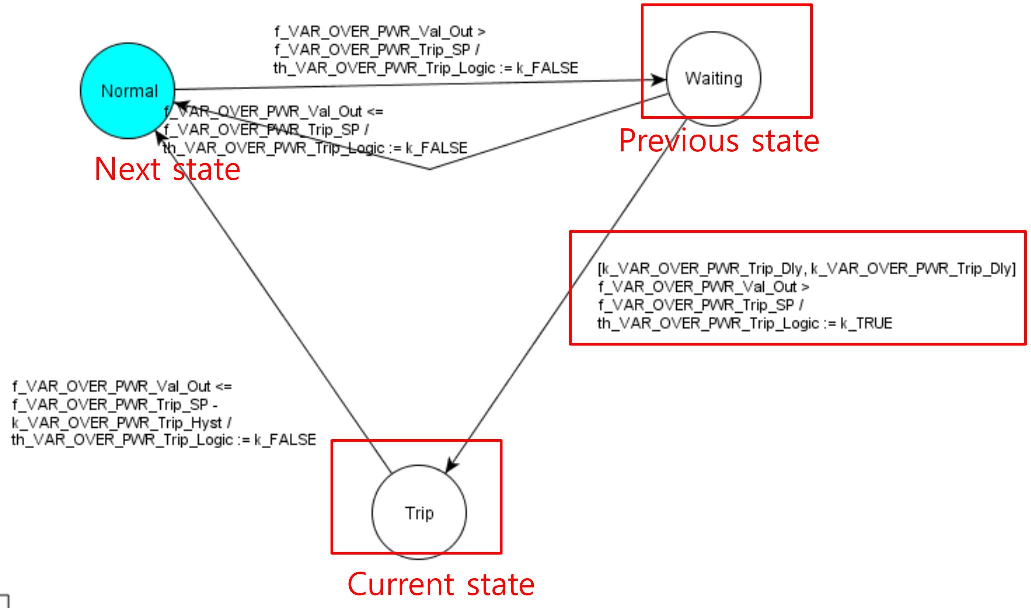
- A template for TTS(2)


T
annotated state 1
...
annotated state n
annotated state 1 at t
transitions
enter the state via state transition
remain at the state because of disabled outgoing transition
T1
T2

Software fault tree templates for NuSCR nodes(6)

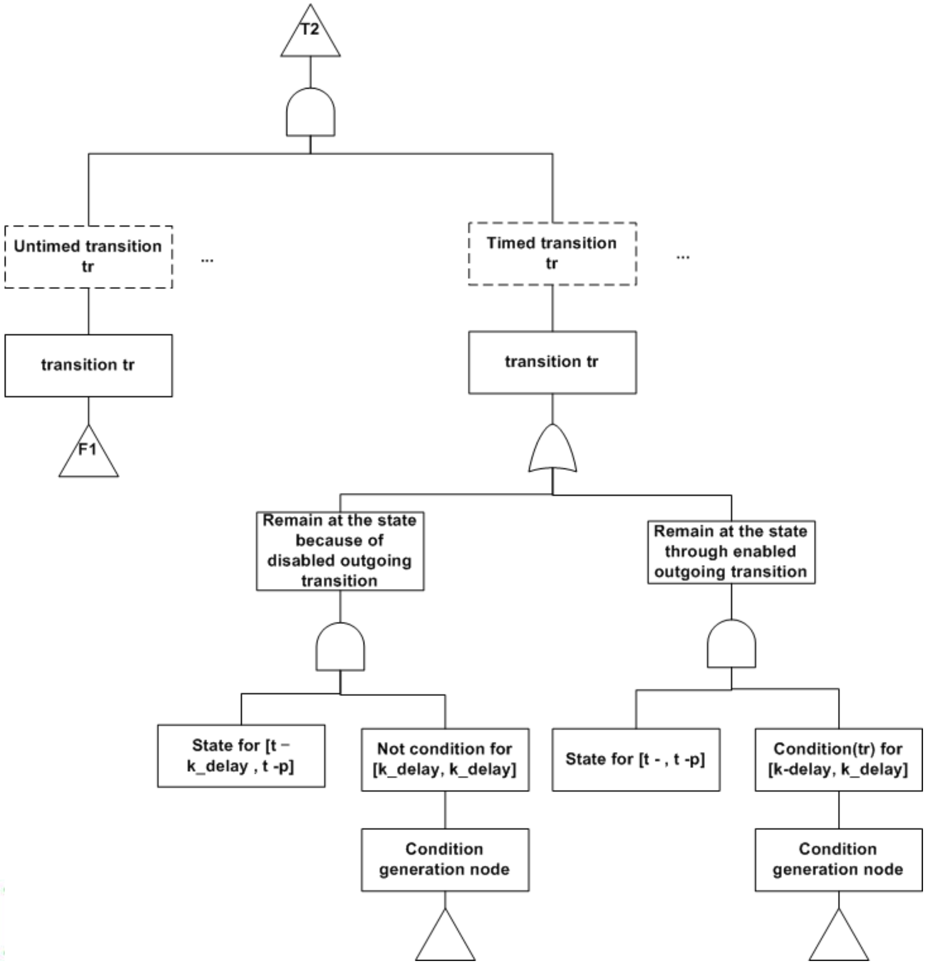
- A template for FSM(3) –enter the state

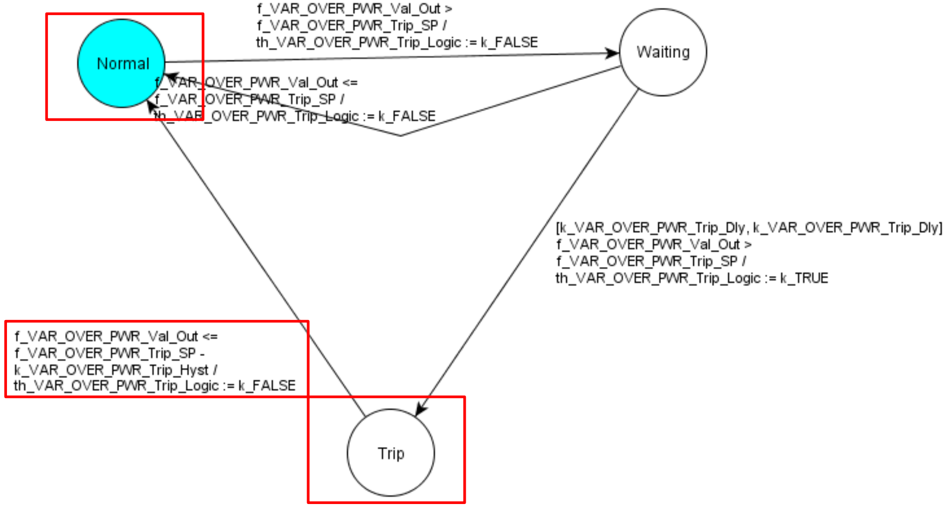

T1
Untimed transition tr
...
Timed transition tr
transition tr
transition tr
source (tr) at t - p
conditions (tr) at t – p
source (tr) for [t - k_delay, t – p]
conditions (tr) for [k_delay, k_delay]
condition generation node
condition generation node

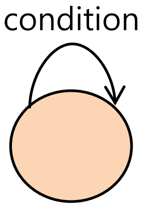

Normal
Waiting
Trip
f_VAR_OVER_PWR_Val_Out >
f_VAR_OVER_PWR_Trip_SP /
th_VAR_OVER_PWR_Trip_Logic := k_FALSE
f_VAR_OVER_PWR_Val_Out <=
f_VAR_OVER_PWR_Trip_SP /
th_VAR_OVER_PWR_Trip_Logic := k_FALSE
[k_VAR_OVER_PWR_Trip_Dly, k_VAR_OVER_PWR_Trip_Dly]
f_VAR_OVER_PWR_Val_Out >
f_VAR_OVER_PWR_Trip_SP /
th_VAR_OVER_PWR_Trip_Logic := k_TRUE
f_VAR_OVER_PWR_Val_Out <=
f_VAR_OVER_PWR_Trip_SP -
k_VAR_OVER_PWR_Trip_Hyst /
th_VAR_OVER_PWR_Trip_Logic := k_FALSE
Next state
Previous state
Current state

Software fault tree templates for NuSCR nodes(7)

- A template for FSM(4) – remain at the state

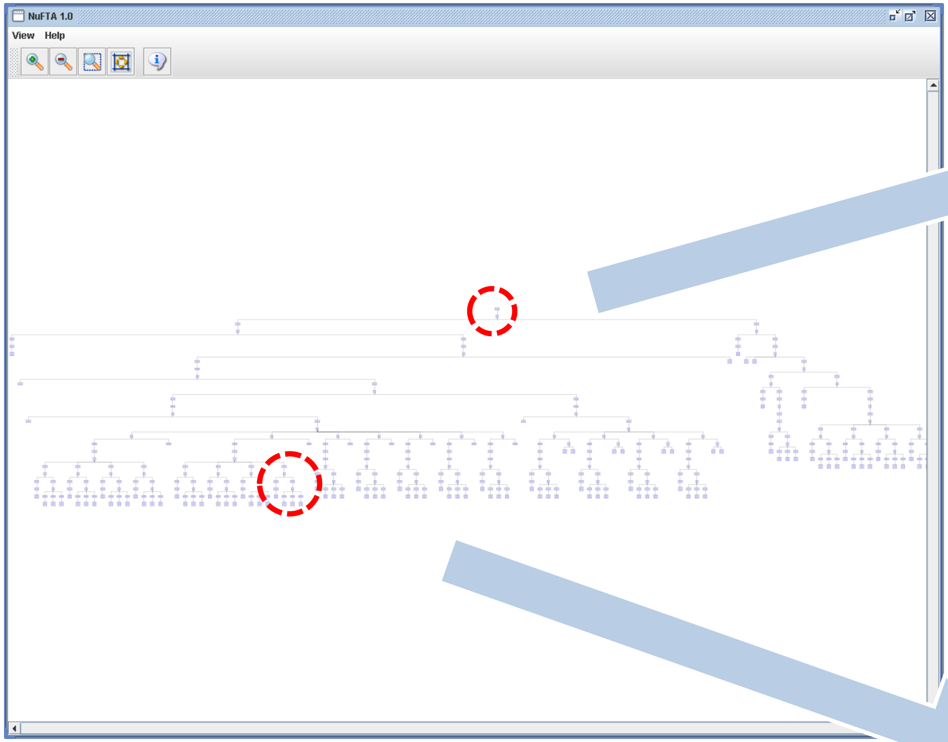

T2
Untimed transition tr
...
Timed transition tr
...
transition tr
transition tr
F1
Remain at the state because of disabled outgoing transition
Remain at the state through enabled outgoing transition
State for [t – k_delay , t -p]
Not condition for [k_delay, k_delay]
State for [t - , t -p]
Condition(tr) for [k-delay, k_delay]
Condition generation node
Condition generation node


Normal
Waiting
Trip
f_VAR_OVER_PWR_Val_Out >
f_VAR_OVER_PWR_Trip_SP /
th_VAR_OVER_PWR_Trip_Logic := k_FALSE
f_VAR_OVER_PWR_Val_Out <=
f_VAR_OVER_PWR_Trip_SP /
th_VAR_OVER_PWR_Trip_Logic := k_FALSE
[k_VAR_OVER_PWR_Trip_Dly, k_VAR_OVER_PWR_Trip_Dly]
f_VAR_OVER_PWR_Val_Out >
f_VAR_OVER_PWR_Trip_SP /
th_VAR_OVER_PWR_Trip_Logic := k_TRUE
f_VAR_OVER_PWR_Val_Out <=
f_VAR_OVER_PWR_Trip_SP -
k_VAR_OVER_PWR_Trip_Hyst /
th_VAR_OVER_PWR_Trip_Logic := k_FALSE

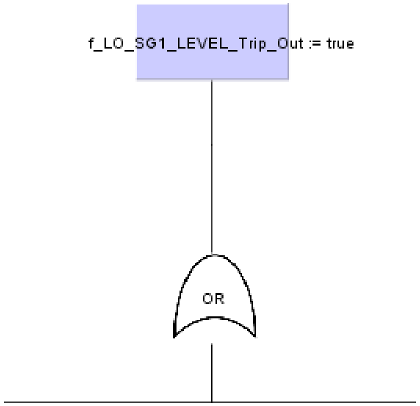


Condition of self-cycling transition is satisfied

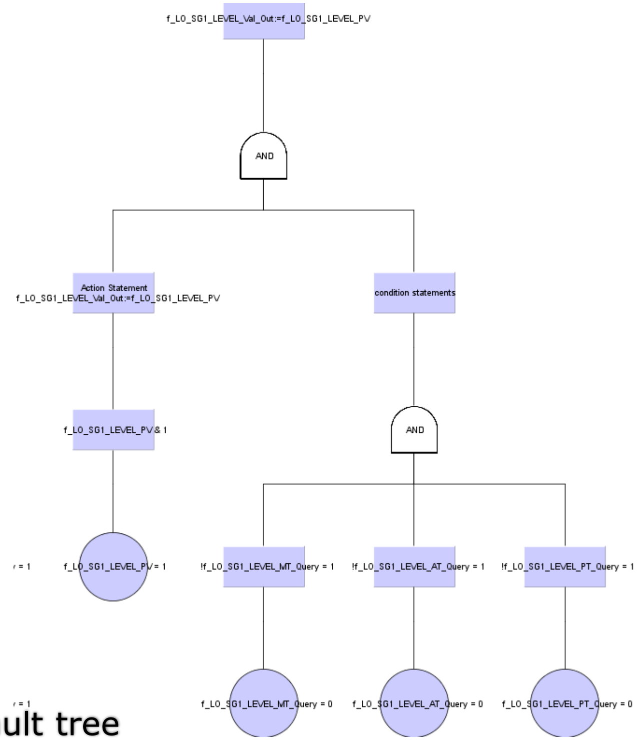
A screen dump of NuFTA



A full generated software fault tree using NuFTA



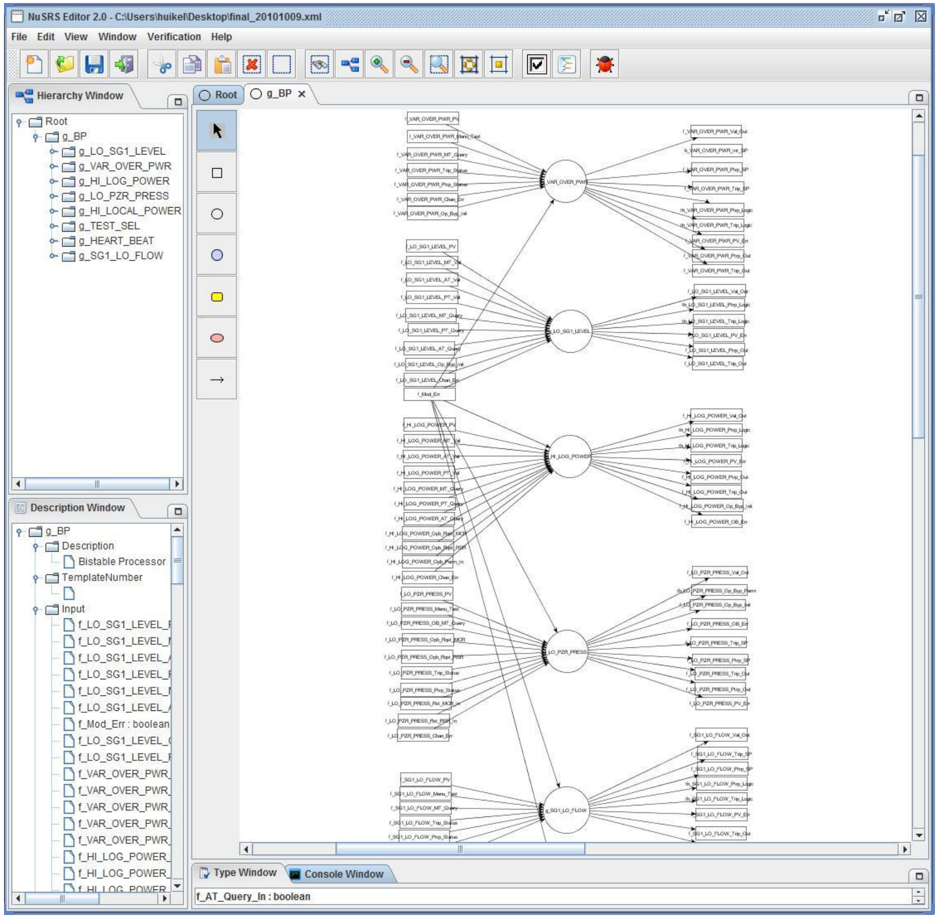
Root node(failure)


f_LO_SG1_LEVEL_Val_Out:=f_LO_SG1_LEVEL_PV
AND
Action Statement
f_LO_SG1_LEVEL_Val_Out:=f_LO_SG1_LEVEL_PV
condition statements
f_LO_SG1_LEVEL_PV & 1
AND
= 1
f_LO_SG1_LEVEL_PV = 1
!f_LO_SG1_LEVEL_MT_Query = 1
!f_LO_SG1_LEVEL_AT_Query = 1
!f_LO_SG1_LEVEL_PT_Query = 1
= 1
f_LO_SG1_LEVEL_MT_Query = 0
f_LO_SG1_LEVEL_AT_Query = 0
f_LO_SG1_LEVEL_PT_Query = 0

Sub fault tree

Experimental Result(1)

- We used a prototype version of requirement specification of KNICS RPS.



f_AT_Query_In : boolean

<FODs of *g_BP*>

Experimental Result(2)

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

- NuFTA constructed SFT from FODs, except the most complex FOD.
- Cause of this problem : state explosion problem
  - Optimization of source code and data structure is required.

Conclusion & Future Work

- Conclusion
  - NuFTA is a CASE tool supporting software fault tree analysis for analysts.
  - We restricted application domain of safety analysis into specific type of critical failure, ‘*shut down*’.
  - We automated large part of safety analysis.

- Future work
  - Optimization of code and data structure
  - Definition of semantics for time constraints