Automated Exhaustive Test Case Generation for FBD Program

Sang Hun Lee [a*], Sung Min Shin[a], Hyun Gook Kang [b], Jong-Gyun Choi[a]
[a]*Korea Atomic Energy Research Institute, 111 Daedeok-daero, 989beon-gil, Yuseong-gu, Daejeon, Republic of Korea*
[b]*Rensselaer Polytechnic Institute, 110 8th Street, Troy, NY, USA*
[*]*Corresponding author: lees@kaeri.re.kr*

1. Introduction

For the function block diagram (FBD) programs that run on programmable logic controllers (PLCs) [1, 2], thorough testing of software is crucial for ensuring the safety of nuclear power plants (NPPs). By its nature, the NPP software is a logical matter and determines the function of hardware in the digitalized environment. The space that digitalized input and internal variables construct can be considered as the domain that the software may encounter during system operation, which may be large but not infinite. If we can perform the software testing over the whole of this space, the limitations related to the state-of-the-art test-based software reliability quantification method, such as the uncertainty in input selection and model parameter estimation [3, 4], can be resolved.

In this study, the automated exhaustive test case generation framework for NPP safety-critical software testing is proposed which formally translates the FBD program into Satisfiability Modulo Theories (SMT) formula and generates exhaustive test cases by iteratively solving the translated SMT formula using an SMT solver.

2. Methods

2.1 Satisfiability Modulo Theory

For the static analysis and program verification, Boolean satisfiability (SAT) and SMT have received considerable attention during the last decade to derive the test cases from various models automatically [5]. The SMT problem is the problem of determining whether such a formula in first-order logic is satisfiable where some function and predicate symbols have additional interpretations such as linear inequalities or function symbols.

As the test case generation is the process of identifying the software variables' states that satisfy the given test requirement, the exhaustive test case generation problem can be formulated as the problem of finding all possible combinations of software variables' states that generate the desired output, as shown in Equation (1).

$$\underset{x_i \in S}{arg}\, f(x_i) = true \cdots\cdots\cdots\cdots\cdots\cdots\cdots (1)$$

In Equation (1), the formula (f) represents the FBD program that defines the logic between the software safety output ($f(x_i)$) and the software input and internal variables, or the literals (x_i). Space (S) is defined as the domain of all possible software input and internal variables' states that may encounter during software operation. For NPP safety software, the test requirement can be set to the safety signal initiation by the software (i.e., $f(x_i)$=$true$). In this case, the derived test cases will reflect all possible safety signal demand situations. In this study, the proposed approach utilizes Z3 [6], to identify the states of the software variables' states that generate the safety signal for FBD program.

2.2 Translation from FBD into SMT

In order to generate exhaustive test cases for FBD program, we first formally defined the FBDs based on the ideas discussed in previous studies [7, 8] and FBD-to-SMT translation rules were developed based on the formal definition of FBDs. As shown in Fig. 1, the translation starts with generating SMT formulas for all individual function blocks (FBs), and continues for component FBD and system FBD.

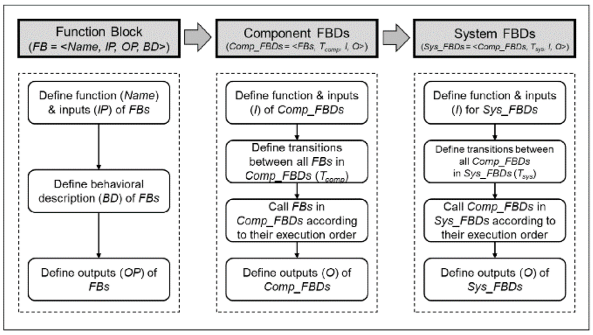

Function Block
(FB = <Name, IP, OP, BD>)
Component FBDs
(Comp_FBDs = <FBs, T_comp, I, O>)
System FBDs
(Sys_FBDs = <Comp_FBDs, T_sys, I, O>)
Define function (Name) & inputs (IP) of FBs
Define behavioral description (BD) of FBs
Define outputs (OP) of FBs
Define function & inputs (I) of Comp_FBDs
Define transitions between all FBs in Comp_FBDs (T_comp)
Call FBs in Comp_FBDs according to their execution order
Define outputs (O) of Comp_FBDs
Define function & inputs (I) for Sys_FBDs
Define transitions between all Comp_FBDs in Sys_FBDs (T_sys)
Call Comp_FBDs in Sys_FBDs according to their execution order
Define outputs (O) of Sys_FBDs

Fig. 1. FBD-to-SMT translation procedure.

2.3 Exhaustive test case generation for FBD program

Given the SMT formulas translated from FBD program, a test case can be generated by finding the states of software input and internal variables that satisfy the selected test requirement. The procedure for generating exhaustive test cases for FBD program consists of three major parts: 1) defining the software variables and FBD program under test, 2) defining the test requirements, and 3) retrieving the model for software variables that satisfies the test requirements. Fig. 2 shows a flowchart of the exhaustive test case generation procedure.

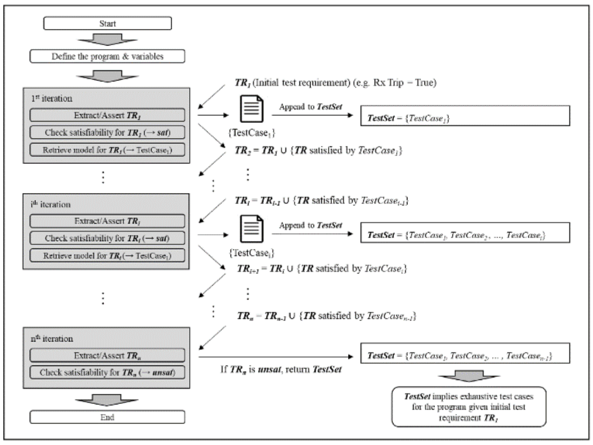

Start
Define the program & variables
1st iteration
Extract/Assert TR1
Check satisfiability for TR1 (→ sat)
Retrieve model for TR1(→ TestCase1)
TR1 (Initial test requirement) (e.g. Rx Trip = True)
Append to TestSet
TestSet = {TestCase1}
{TestCase1}
TR2 = TR1 ∪ {TR satisfied by TestCase1}
ith iteration
Extract/Assert TRi
Check satisfiability for TRi (→ sat)
Retrieve model for TRi(→ TestCasei)
TRi = TRi-1 ∪ {TR satisfied by TestCasei-1}
Append to TestSet
TestSet = {TestCase1, TestCase2, ..., TestCasei}
{TestCasei}
TRi+1 = TRi ∪ {TR satisfied by TestCasei}
TRn = TRn-1 ∪ {TR satisfied by TestCasen-1}
nth iteration
Extract/Assert TRn
Check satisfiability for TRn (→ unsat)
TestSet = {TestCase1, TestCase2, ... , TestCasen-1}
If TRn is unsat, return TestSet
TestSet implies exhaustive test cases for the program given initial test requirement TR1
End

Fig. 2. Flowchart of exhaustive test case generation procedure.

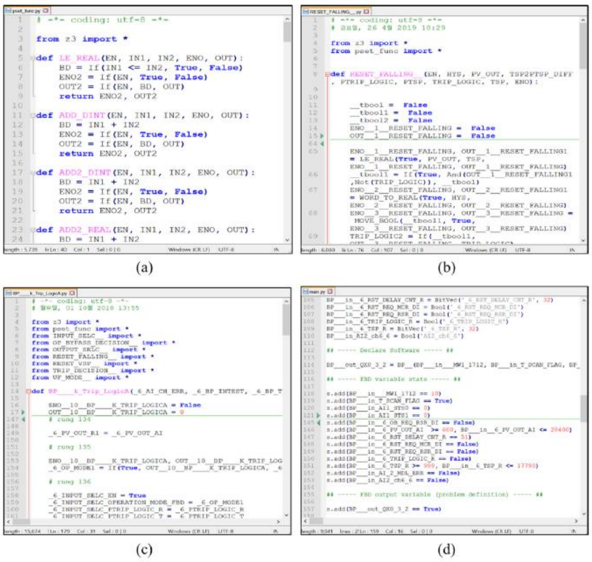
In the first part, the program information required for test case generation, such as program variables and the FBD programs, are defined. Here, the names and possible ranges of all variables under test and constants as well as the SMT formula translated based on FBD-to-SMT translation rules are declared. In the second part, the initial test requirement (TR_1) is defined and added as a set of constraints of the formula to be solved. In the third part, the algorithm at least one interpretation for the defined variables that evaluate a given formula to satisfy test requirement, the SMT solver returns sat as a satisfiability check result. If it is unsatisfiable, it returns unsat. If the formula is satisfiable, the model that makes the formula true is retrieved and saved as a single test case for the FBD program. In order to derive another model, the negation of the derived model at each iteration ($TestCase_i$ at i-th iteration) is added as a new constraint to the test requirement which will be used for next iteration (TR_{i+1}). This requires the SMT solver to find another solution at the next iteration ($TestCase_{i+1}$ at $(i+1)$-th iteration) for the formula except the ones found in previous iterations. At each iteration, the derived model is added to the *TestSet*. The process is repeated until the solver returns unsatisfiable (unsat) as a satisfiability check result for the formula given the test requirement. In result, the derived *TestSet* represent the exhaustive test cases for the initial test requirement where each test case is exclusive to each other

3. Case Study

3.1 Exhaustive test case generation of target software

As a case study, the exhaustive test cases for pressurizer pressure-low reactor trip logic of KNICS IDiPS-RPS bistable processor (BP) trip logic software [9] were derived. The FBD program of BP software was loaded to the FETCG which generates the test module and the translated SMT formula, as shown in Fig. 3. The test module generates exhaustive test cases from the defined test requirement for the FBD program based on the three levels of translated SMT files (i.e., FBs, component FBDs, and system FBD). While the BP trip logic software generates various external outputs including heartbeat and diagnostic signals, the system output variable considered in this study for test case generation purpose was selected as trip signal output.

The FETCG then generates the SMT formula for the software output which the tester provided and derives the software input and internal variables. In the case study, the FETCG derived 35 variables among a total of 612 variables defined in BP software as the program input and internal variables that contribute to generating the pressurizer-pressure-low trip signal output.


pset_func.py
length : 5,739   ln : 40   Col : 1   Sel : 0 | 0   Windows (CR LF)   UTF-8   IN

```
# -*- coding: utf-8 -*-

from z3 import *

def LE_REAL(EN, IN1, IN2, ENO, OUT):
    BD = If(IN1 <= IN2, True, False)
    ENO2 = If(EN, True, False)
    OUT2 = If(EN, BD, OUT)
    return ENO2, OUT2

def ADD_DINT(EN, IN1, IN2, ENO, OUT):
    BD = IN1 + IN2
    ENO2 = If(EN, True, False)
    OUT2 = If(EN, BD, OUT)
    return ENO2, OUT2

def ADD2_DINT(EN, IN1, IN2, ENO, OUT):
    BD = IN1 + IN2
    ENO2 = If(EN, True, False)
    OUT2 = If(EN, BD, OUT)
    return ENO2, OUT2

def ADD2_REAL(EN, IN1, IN2, ENO, OUT):
    BD = IN1 + IN2
```

(a)

RESET_FALLING__.py
length : 6,080   ln : 76   Col : 107   Sel : 0 | 0   Windows (CR LF)   UTF-8   IN

```
# -*- coding: utf-8 -*-
# 금요일, 26 4월 2019 10:29

from z3 import *
from pset_func import *


def RESET_FALLING__(EN, HYS, PV_OUT, TSP2PTSP_DIFF
, PTRIP_LOGIC, PTSP, TRIP_LOGIC, TSP, ENO):


    __tbool =  False
    __tbool1 =  False
    __tbool2 =  False
    ENO__1__RESET_FALLING =  False
    OUT__1__RESET_FALLING =  False

    ENO__1__RESET_FALLING, OUT__1__RESET_FALLING1
    = LE_REAL(True, PV_OUT, TSP,
    ENO__1__RESET_FALLING, OUT__1__RESET_FALLING)
    __tbool1 = If(True, And(OUT__1__RESET_FALLING1
    ,Not(TRIP_LOGIC)), __tbool)
    ENO__2__RESET_FALLING, OUT__2__RESET_FALLING1
    = WORD_TO_REAL(True, HYS,
    ENO__2__RESET_FALLING, OUT__2__RESET_FALLING)
    ENO__3__RESET_FALLING, OUT__3__RESET_FALLING =
     MOVE_BOOL(__tbool1, True,
    ENO__3__RESET_FALLING, OUT__3__RESET_FALLING)
    TRIP_LOGIC2 = If(__tbool1,
```

(b)

BP____k_Trip_LogicA.py
length : 15,674   Ln : 179   Col : 31   Sel : 0 | 0   Windows (CR LF)   UTF-8   IN

```
# -*- coding: utf-8 -*-
# 월요일, 01 10월 2018 13:55

from z3 import *
from pset_func import *
from INPUT_SELC__ import *
from OP_BYPASS_DECISION__ import *
from OUTPUT_SELC__ import *
from RESET_FALLING__ import *
from RESET_VSP__ import *
from TRIP_DECISION__ import *
from UF_MODE__ import *

def BP____k_Trip_LogicA(_6_AI_CH_ERR, _6_BP_INTEST, _6_BP_T

    ENO__10__BP____K_TRIP_LOGICA = False
    OUT__10__BP____K_TRIP_LOGICA = 0
    # rung 134

    _6_PV_OUT_R1 = _6_PV_OUT_AI

    # rung 135

    ENO__10__BP____K_TRIP_LOGICA, OUT__10__BP____K_TRIP_LOG
    _6_OP_MODE1 = If(True, OUT__10__BP____K_TRIP_LOGICA, _6

    # rung 136

    _6_INPUT_SELC_EN = True
    _6_INPUT_SELC_OPERATION_MODE_FBD = _6_OP_MODE1
    _6_INPUT_SELC_PTRIP_LOGIC_R = _6_PTRIP_LOGIC_R
    _6_INPUT_SELC_PTRIP_LOGIC_T = _6_PTRIP_LOGIC_T
```

(c)

main.py
length : 9,941   lines : 2 Ln : 159   Col : 16   Sel : 0 | 0   Windows (CR LF)   UTF-8   IN

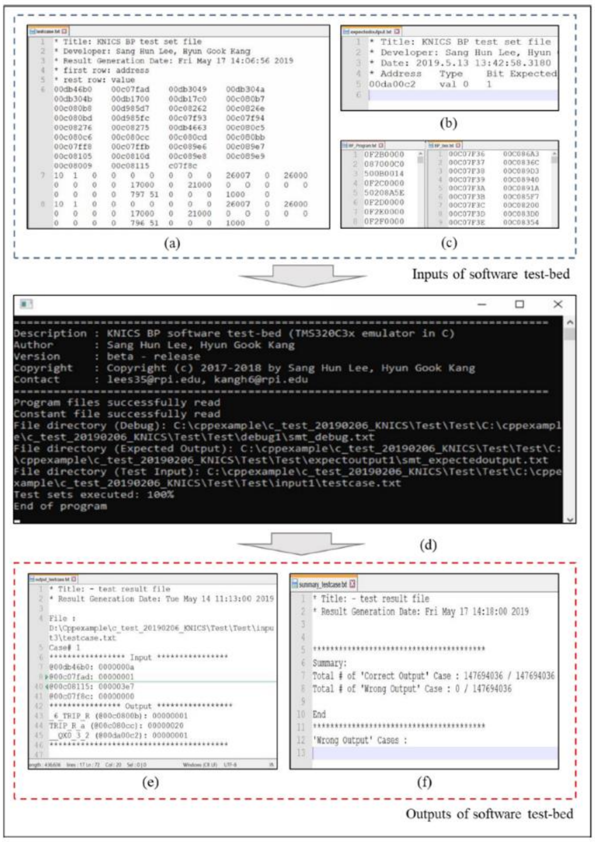
```
BP___in__6_RST_DELAY_CNT_R = BitVec('_6_RST_DELAY_CNT_R', 32)
BP___in__6_RST_REQ_MCR_DI = Bool('_6_RST_REQ_MCR_DI')
BP___in__6_RST_REQ_RSR_DI = Bool('_6_RST_REQ_RSR_DI')
BP___in__6_TRIP_LOGIC_R = Bool('_6_TRIP_LOGIC_R')
BP___in__6_TSP_R = BitVec('_6_TSP_R', 32)
BP___in_AI2_ch6_6 = Bool('AI2_ch6_6')

## ----- Declare Software ----- ##

BP___out_QX0_3_2 = BP__(BP___in___MW1_1712, BP___in_T_SCAN_FLAG, BP_

## ----- FBD variable state ----- ##

s.add(BP___in___MW1_1712 == 10)
s.add(BP___in_T_SCAN_FLAG == True)
s.add(BP___in_AI1_STS0 == 0)
s.add(BP___in_AI1_STS1 == 0)
s.add(BP___in__6_OB_REQ_RSR_DI == False)
s.add(BP___in__6_PV_OUT_AI >= 600, BP___in__6_PV_OUT_AI <= 29400)
s.add(BP___in__6_RST_DELAY_CNT_R == 51)
s.add(BP___in__6_RST_REQ_MCR_DI == False)
s.add(BP___in__6_RST_REQ_RSR_DI == False)
s.add(BP___in__6_TRIP_LOGIC_R == False)
s.add(BP___in__6_TSP_R >= 999, BP___in__6_TSP_R <= 17790)
s.add(BP___in_AI_2_MDL_ERR == False)
s.add(BP___in_AI2_ch6_6 == False)

## ----- FBD output variable (problem definition) ----- ##

s.add(BP___out_QX0_3_2 == True)
```

(d)

Fig. 3. Generated files of the FETCG for IDiPS-RPS BP software: (a) Function block, (b) Component FBDs, (c) System FBD, (d) Test module.

3.2 Test execution and result analysis

The generated exhaustive test cases were post-processed into the format of software test-bed to verify whether the machine code of BP software generates the trip signal given the derived test cases. Fig. 4 shows the test results for the exhaustive test cases using a simulation-based software test-bed developed in authors' previous study [10]. The expected output for all test cases is set as the value of the memory address of output variable to be true (0×1). In result, all the 147,694,036 test cases generated the trip signal, and the test was conducted in 3.45 hrs using 64 units of 3-GHz logical processors (～ 5.38 msec per test case). As all test cases generated correct output, BP software was proven to be error-free in terms of its safety function (pressurizer-pressure-low trip signal generation) for the target scenario.



```
* Title: KNICS BP test set file
* Developer: Sang Hun Lee, Hyun Gook Kang
* Result Generation Date: Fri May 17 14:06:56 2019
* first row: address
* rest row: value
00db46b0    00c07fad    00db3049    00db304a
00db304b    00db1700    00db17c0    00c080b7
00c080b8    00d985d7    00c08262    00c0826e
00c080bd    00d985fc    00c07f93    00c07f94
00c08276    00c08275    00db4663    00c080c5
00c080c6    00c080cc    00c080cd    00c080bb
00c07ff8    00c07ffb    00c089e6    00c089e7
00c08105    00c0810d    00c089e8    00c089e9
00c08009    00c08115    c07f8c
10  1   0   0   0   0   0   0   0   26007   0   26000
0   0   0   0   17000   0   21000   0   0   0   0   0
0   0   0   0   797 51  0   0   0   1000    0
10  1   0   0   0   0   0   0   0   26007   0   26000
0   0   0   0   17000   0   21000   0   0   0   0   0
0   0   0   0   796 51  0   0   0   1000    0
```

```
* Title: KNICS BP test set file
* Developer: Sang Hun Lee, Hyun
* Date: 2019.5.13 13:42:58.3180
* Address   Type    Bit Expected
00da00c2    val 0   1
```

```
0F2B0000
087000C0
500B0014
0F2C0000
50208A5E
0F2D0000
0F2E0000
0F2F0000
```

```
00C07F36      00C086A3
00C07F37      00C0836C
00C07F38      00C089D3
00C07F39      00C08940
00C07F3A      00C0891A
00C07F3B      00C085F7
00C07F3C      00C08200
00C07F3D      00C083D0
00C07F3E      00C08354
```

```
================================================================================
Description : KNICS BP software test-bed (TMS320C3x emulator in C)
Author      : Sang Hun Lee, Hyun Gook Kang
Version     : beta - release
Copyright   : Copyright (c) 2017-2018 by Sang Hun Lee, Hyun Gook Kang
Contact     : lees35@rpi.edu, kangh6@rpi.edu
================================================================================
Program files successfully read
Constant file successfully read
File directory (Debug): C:\cppexample\c_test_20190206_KNICS\Test\Test\C:\cppexampl
e\c_test_20190206_KNICS\Test\Test\debug1\smt_debug.txt
File directory (Expected Output): C:\cppexample\c_test_20190206_KNICS\Test\Test\C:
\cppexample\c_test_20190206_KNICS\Test\Test\expectoutput1\smt_expectedoutput.txt
File directory (Test Input): C:\cppexample\c_test_20190206_KNICS\Test\Test\C:\cppe
xample\c_test_20190206_KNICS\Test\Test\input1\testcase.txt
Test sets executed: 100%
End of program
```

```
* Title: - test result file
* Result Generation Date: Tue May 14 11:13:00 2019

File :
D:\Cppexample\c_test_20190206_KNICS\Test\Test\inpu
t3\testcase.txt
Case# 1
***************** Input ****************
@00db46b0: 0000000a
@00c07fad: 00000001
@00c08115: 000003e7
@00c07f8c: 00000000
**************** Output *****************
_6_TRIP_R (@00c0800b): 00000001
TRIP_R_a (@00c080cc): 00000020
__QX0_3_2 (@00da00c2): 00000001
****************************************
```

```
* Title: - test result file
* Result Generation Date: Fri May 17 14:18:00 2019


****************************************
Summary:
Total # of 'Correct Output' Case : 147694036 / 147694036
Total # of 'Wrong Output' Case : 0 / 147694036

End
****************************************
'Wrong Output' Cases :
```

Fig. 4. Test results of the case study for BP pressurizer-pressure-low trip logic: (a) Test case file, (b) Expected output file, (c) Program file, (d) Screenshot of simulation-based test-bed execution, (e) Test result file, (f) Test summary file.

4. Conclusion

This study proposes an automated exhaustive test case generation method for FBD programs. Testing the NPP software is important in demonstrating that the software generates its dedicated safety function when demand comes. The proposed framework formally translates the FBD program to SMT formula and the FBDET algorithm is developed for the exhaustive test case generation. As an application of the proposed software test method, the exhaustive test cases for the pressurizer-pressure-low trip logic of KNICS IDiPS-RPS BP software were derived and tested using the software test-bed.

Although the proposed framework focused on the exhaustive test case generation method for the FBD program used in NPP digital safety systems, the verification and the performance of the SMT solver used in the framework should be further investigated for its ability to generate a correct model and test cases.

While this study only considered the normal range of the software variables that can occur during its operation; however, the abnormal inputs that can occur due to the faults in the PLC module or memory will be also investigated in the future work.

ACKNOWLEDGEMENT

This work was supported by the Nuclear Safety Research Program through the Korea Foundation Of Nuclear Safety(KoFONS) using the financial resource granted by the Nuclear Safety and Security Commission(NSSC) of the Republic of Korea. (No. 2106027)

REFERENCES

[1] IEC, Programmable controllers - Part 3: Programming languages, International Electrotechnical Commission, Geneva, Switzerland, IEC 61131-3:2013, 2013.
[2] P. S. Acharyulu and P. Seetharamaiah, A framework for safety automation of safety-critical systems operations, Safety Science, Vol. 77, pp. 133-142, 2015.
[3] T. L. Chu et al., Development of a statistical testing approach for quantifying safety-related digital system on demand failure probability, U.S. NRC, Washington, DC, USA, NUREG/CR-7234, 2017.
[4] H. G. Kang, H. G. Lim, H. J. Lee, M. C. Kim and S. C. Jang, Input-profile-based software failure probability quantification for safety signal generation systems, Reliability Engineering System Safety, Vol. 94, No. 10, pp. 1542-1546, 2009.
[5] C. Barrett and C. Tinelli, Satisfiability modulo theories, in Handbook of Model Checking, Cham, Switzerland, Springer, pp. 305-343, 2018.
[6] L. d. Moura and N. Bjørner, Z3: An efficient SMT solver, The 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Budapest, Hungary, Mar. 29 - Apr. 6, 2008.
[7] J. Yoo, E.-S. Kim, J.-S. Lee, A behavior-preserving translation from FBD design to C implementation for reactor protection system software, Nuclear Engineering Technology, Vol 45, No. 4, pp. 489–504, 2013.
[8] D. A. Lee, J. Yoo, J. S. Lee, A systematic verification of behavioral consistency between FBD design and ansi-c implementation using HW-CBMC, Reliability Engineering System Safety, No. 120, pp. 139-149, 2013.
[9] K. C. Kwon and M. S. Lee, Technical review on the localized digital instrumentation and control systems, Nuclear Engineering Technology., Vol. 41, No. 4, pp. 447-454, 2009.
[10] S. H. Lee, S. J. Lee, J. Park, E. C. Lee, H. G. Kang. Development of simulation-based testing environment for safety-critical software. Nuclear Engineering Technology, Vol 50, No. 4, pp. 570-581, 2018.