Automated Exhaustive Test Case Generation for Safety-critical Software

Sang Hun Lee[1], Seung Jun Lee[2], Sung Min Shin[3], Eun-chan Lee[4], Hyun Gook Kang[1]

[1]Department of Mechanical, Aerospace and Nuclear Engineering, Rensselaer Polytechnic Institute (RPI), 110 8th street, Troy, NY, USA, 12180, lees35;kangh6@rpi.edu
[2]School of Mechanical, Aerospace and Nuclear Engineering, Ulsan National Institute of Science and Technology (UNIST), 50 UNIST-gil, Ulsan, Republic of Korea, 44919, sjlee420@unist.ac.kr
[3]Reactor System Safety Research Division, Korea Atomic Energy Research Institute (KAERI), 111 Daedeok-daero, 989beon-gil, Yuseong-gu, Daejeon, Republic of Korea, 34057, smshin@kaeri.re.kr
[4]Korea Hydro & Nuclear Power Co., Ltd., 1655 Bulguk-ro, Gyeongju-si, Gyeongsangbuk-do, Republic of Korea, 38120, eclee5639@khnp.co.kr

INTRODUCTION

As existing nuclear power plants (NPPs) are employing digital technologies in both safety-critical and safety-related systems, an issue of incorporating the software reliability to NPP probabilistic risk assessment (PRA) model revealed the importance of the reliability quantification of safety-critical software. In response, various quantitative software reliability methods (QSRMs) such as software reliability growth model (SRGM) [1], Bayesian belief network (BBN) model [2], and test-based method [3, 4] have been proposed and adopted in the nuclear field. However, the limitations of current state-of-the-art methods include: 1) the uncertainty in estimating model parameters, 2) a long testing time for each test case, and 3) and the limitation on demonstrating the test inputs match to the actual operation profile.

By its nature, the software is a logical matter and determines the function of hardware in the digital system. The space that digitalized inputs and internal variables construct is considered as the domain that the software may encounter during its operation, which may be very large but not infinite. For many safety-critical systems, the number of software variables is limited, and they have limited resolution. Therefore, if we can perform the testing over the whole space, the issues related to input selection and model parameter estimation can be resolved.

This paper proposes an automated exhaustive test case generation framework for NPP safety-critical software testing. From the viewpoint of NPP safety, the testing of NPP safety software needs to focus on the failure of its dedicated safety function when demand comes (i.e., failure-on-demand). As the software output is determined by the combinations of the states of software input and internal variables, the exhaustive test case generation for safety software can be considered as a problem of finding the solutions that satisfy its on-demand situation, i.e. Satisfiability Modulo Theories (SMT) problem [5]. The proposed framework formally translates the function block diagram (FBD) program into a semantically-equivalent SMT formula and generates exhaustive test cases by iteratively solving the translated SMT formula using SMT solver.

METHODS

Translation of FBD into SMT formula

FBD program is a network of function blocks executed sequentially according to their predefined execution order [6]. To generate exhaustive test cases for FBD program, we first formally define the FBDs based on the ideas discussed in previous studies [7] and translate FBD to semantically-equivalent SMT formula to be later solved by SMT solver given test requirement. Fig. 1 shows an overview of the translation algorithm in the flowchart form. The translation starts with generating SMT formulas for individual function blocks (FBs) used in the FBD program and continues for component FBD and system FBD.

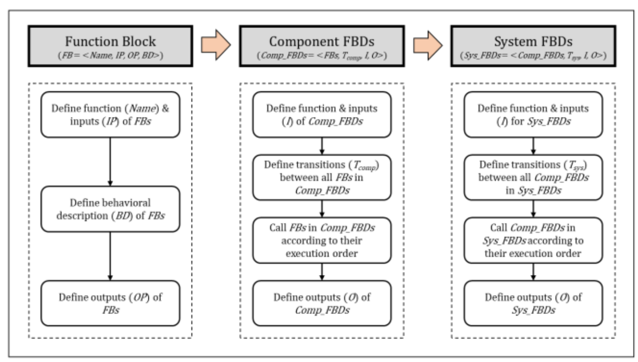


Fig. 1. An overview of FBD-to-SMT translation scheme.

Function Block Translation

A FB is the basic program organizing unit that has assigned input parameters and static variables [6]. A FB is defined as a tuple composed of a name (*Name*), input ports (*IP*), output ports (*OP*), and its behavior description (*BD*). Fig. 2 shows an example of how a FB can be translated into an equivalent SMT formula. First, the components of FB including its *Name*, *IP*, and *OP* are declared, and the value of the output variable is determined by its specific *BD* according to their types (e.g., arithmetic and logical, selection, comparison operation). Finally, a function definition ends with returning the outputs of FBs.

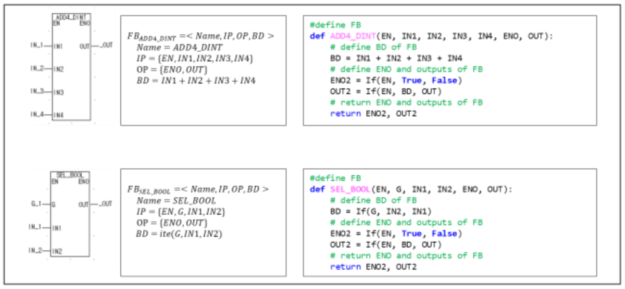


Fig. 2. FBD-to-SMT translation for function blocks.

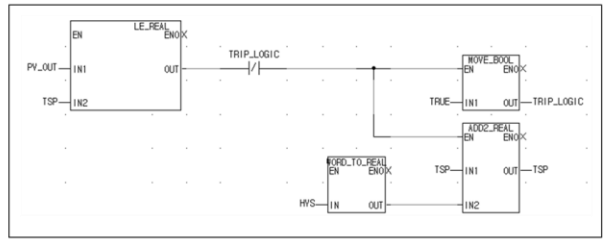
Component FBD Translation

A component FBD (*Comp_FBD*) is a logical block that is composed of FBs according to their sequential execution orders. It is defined as a tuple consisting of a set of function blocks (*FB*s), a set of transitions (T_{comp}) between the function blocks, a set of input ports (I), and a set of output ports (O). Fig. 3 shows an example of translation for component FBDs composed of four FBs with three transition relations. First, the elements of component FBD, including the name of *Comp_FBD*, I and O, are defined. As the behavior of component FBD is defined as sequential calls of function blocks described in T_{comp}, the program flow is defined by calling *FB*s according to their execution order. Here, the temporary variables that store the value of output ports of FBs in *Comp_FBD* or the ladder connections at each rung are defined to model a set of transitions between FBs. The definition ends by returning its outputs.

System FBD Translation

The system FBD (*Sys_FBD*) is a set of component FBDs connected according to their sequential execution orders. It is defined as a tuple of component FBDs (*Comp_FBD*), a set of transition between component FBDs (T_{sys}), a set of input ports (I), and a set of output ports (O).

The translation rules for system FBDs are similar to those of component FBDs while it is composed of sequential function calls of component FBDs instead of FBs and uses temporary variables to model a set of transitions between *Comp_FBDs*. The output of component FBDs are used as inputs to other component FBDs according to the transitions (T_{sys}), and they are called in *Sys_FBD* according to their execution orders. In this study, the whole NPP safety software is considered as one system FBD, and the enables of system FBD (*EN* and *ENO*) are not defined in the translation since the programmable logic controller (PLC) operates a system FBD periodically with specific scan cycle.



```
# Define function and arguments of Comp_FBD
def RESET_FALLING__(EN, HYS, PV_OUT, ENO, TRIP_LOGIC, TSP):

    # Define Comp_FBD operation and all translation relation
    ENO__1__RESET_FALLING, OUT__1__RESET_FALLING = LE_REAL(True, PV_OUT,
        TSP, ENO__1__RESET_FALLING, OUT__1__RESET_FALLING)
    __tbool = If(True, And(OUT__1__RESET_FALLING, Not(TRIP_LOGIC)), __tbool)
    ENO__2__RESET_FALLING, OUT__2__RESET_FALLING = WORD_TO_REAL(True, HYS,
        ENO__2__RESET_FALLING, OUT__2__RESET_FALLING)
    ENO__3__RESET_FALLING, OUT__3__RESET_FALLING = MOVE_BOOL(__tbool, True,
        ENO__3__RESET_FALLING, OUT__3__RESET_FALLING)
    TRIP_LOGIC1 = If(__tbool, OUT__3__RESET_FALLING, TRIP_LOGIC)
    ENO__4__RESET_FALLING, OUT__4__RESET_FALLING = ADD2_REAL(__tbool, TSP,
        OUT__2__RESET_FALLING, ENO__4__RESET_FALLING, OUT__4__RESET_FALLING)
    TSP1 = If(__tbool, OUT__4__RESET_FALLING, TSP)

    # Define ENO and output of Comp_FBD
    ENO1 = If(EN, True, False)
    TRIP_LOGIC2 = If(EN, TRIP_LOGIC1, TRIP_LOGIC)
    TSP2 = If(EN, TSP1, TSP)

    # return ENO and outputs of Comp_FBD
    return ENO1, TSP2, TRIP_LOGIC2
```

Fig. 3. FBD-to-SMT translation for component FBDs.

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 input and internal variables of the FBD program that satisfy the selected test requirement. In this study, an FBD exhaustive test case generation (*FBDET*) algorithm is developed which checks the satisfiability of the translated FBD program and retrieve the models for software input and internal variables given certain software output. Generating exhaustive test cases for FBD program consists of three steps: 1) defining the FBD program and variables under test, 2) defining the test requirements, and 3) retrieving the model for FBD variables that satisfies the test requirements. Fig. 4 shows an overview of the exhaustive test case generation scheme.

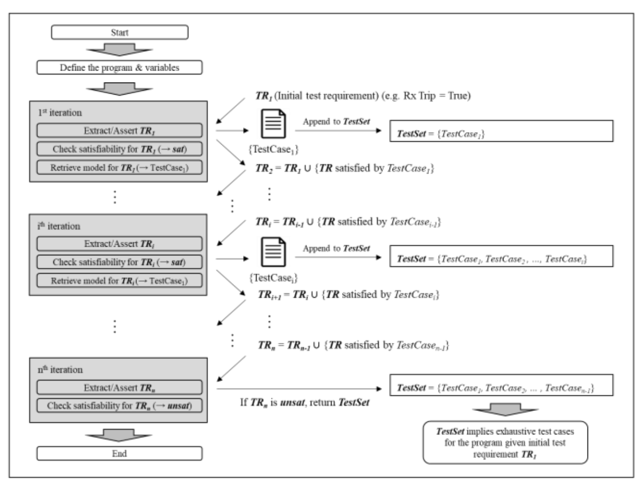


Fig. 4. Test case generation procedure for FBD program.

In the first part, the program information required for test case generation, such as the FBD program and variables, are defined. 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. For NPP safety software, the initial test requirement can be defined as the safety signal initiation (e.g., reactor trip signal = true) by FBD program. In the third part, the algorithm invokes SMT solver [8] to derive the model that satisfies the SMT formula given test requirement. If the formula is satisfiable meaning there exists at least one interpretation for the variables that evaluate a given formula that satisfies the test requirement, the model that makes the formula satisfiable is retrieved and saved as a single test case for FBD program. To derive another model for the formula, the negation of the derived model at each iteration ($TestCase_i$ at i-th iteration) is added as a constraint to the formula which will be used for next iteration (TR_{i+1}). The new test requirement 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 at previous iterations. The process is repeated until the SMT solver returns unsatisfiable for the formula and returns the derived models (*TestSet*) that represent the exhaustive test cases for the initial test requirement where each test case is exclusive to each other.

CASE STUDY

The effectiveness of the proposed exhaustive test case generation framework was demonstrated with the trip logic software of a fully digitalized reactor protection system (IDiPS-RPS), developed under the Korea Nuclear Instrumentation & Control Systems (KNICS) project [9]. As a case study, the exhaustive test cases for pressurizer-pressure-low (PZR_PR_LO) trip logic were developed.

Target NPP Safety Software

In IDiPS-RPS system, BP compares process variables transmitted from plant sensors with the pre-defined trip set-points and generates the trip signals which are transmitted to coincidence processors (CPs) for the voting logic. In BP, 19 different trip logic modules are defined and programmed with FBD language. A system FBD of KNICS IDiPS-RPS BP trip logic is composed of 13 different modules where a total of 475 rungs of ladder logic, 242 FBs, and 122 component FBDs are defined.

Among 19 trip logics, the PZR_PR_LO) trip logic is one of the most complicated logics among BP trip logics which has various functions such as reset delay timer and the trip set-point reset by an operator, making it a good example to demonstrate the applicability of the proposed method to a complex safety system. The detailed operation logic of the PZR_PR_LO trip can be found in [10].

Exhaustive Test Case Generation of Target Software

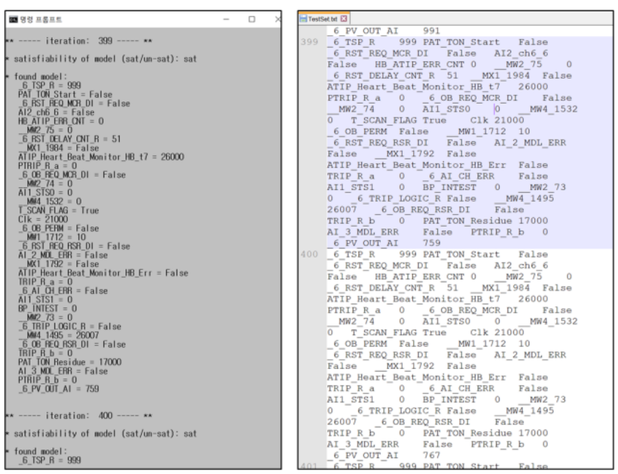
The proposed FBD-to-SMT translation rules were applied to the FBD program of KNICS IDiPS-RPS BP trip logic. For this purpose, a prototype of the FBD-to-SMT translator was developed which generates the test module and the SMT formula translated from the FBD program, as depicted in Fig. 5. It includes four windows: 1) one for the test module which invokes *FBDET* algorithm to generate exhaustive test cases from the defined test requirement for FBD program, and 2) the others for three levels of translated SMT files (i.e., FBs, component FBDs, and a system FBD).

Fig. 5. Generated files of the FBD-to-SMT translator: (a) function block; (b) Component FBDs; (c) System FBD; (d) test module for PZR_PR_LO trip logic.

In the case study, the desired software output was set to be the PZR_PR_LO trip signal, and 35 variables among a total of 612 variables defined in BP software were identified as the input and internal variables that contribute to generating the PZR_PR_LO trip signal. The exhaustive test cases that represent the trip generation by the FBD program were generated for the target scenario that: 1) the software is running in normal operation, 2) there are no errors in BP hardware and heartbeat signal, 3) operators do not request trip bypass or trip set-point reset. Based on the available software specification documents [11], the possible states of program input and internal variable were identified and defined in the test module shown in Fig. 5-(d).

After the software variables are defined, the test module generates the test cases for the target scenario based on the *FBDET* algorithm. The initial test requirement was set to be the desired software output, which is the PZR_PR_LO trip signal generation (*QX0_3_2* = True). Fig. 6 shows the generated exhaustive test cases from the *FBDET* execution. Each line at Fig. 6-(b) shows a single test case which describes the states of software input and internal variables that makes the software output to be true. In result, a total of 147,694,036 test cases were generated as exhaustive test cases for the target scenario.

An important characteristic of the proposed framework includes: 1) the number test sets for exhaustive testing of the safety-critical software can be quantitatively derived, 2) combining the proposed method with the software test-bed [12] can enable effective testing to demonstrate the completeness of the software in terms of its safety function.



(a) (b)

Fig. 6. Exhaustive test case generation for BP PZR_PR_LO trip logic: (a) Screenshot of the test module execution, (b) A part of generated exhaustive test cases for target scenario.

CONCLUSION

This paper proposed a novel automated exhaustive test case generation framework for the FBD programs. The proposed framework formally translates the FBD program to SMT formula and implements the *FBDET* algorithm developed for exhaustive test case generation. As an application of the proposed software test method, the exhaustive test cases for the PZR_PR_LO trip of KNICS IDiPS-RPS BP trip logic software were derived.

The contribution of the proposed framework are as follows: First, the translation rules from FBD to SMT formula is developed based on the FBD formal semantics and implemented to the FBD-to-SMT translator. Second, the proposed test case generation framework provides a method to generate the exhaustive test cases of NPP safety software that can prove that the software error-free for its safety function. As a future work, we plan to develop a tool suite which supports and automates the entire process of proposed software test method from translating the FBD program of NPP safety software to test result generation.

ACKNOWLEDGEMENTS

This work was supported by the project of 'Evaluation of human error probabilities and safety software reliabilities in digital environment (L16S092000),' which was funded by the Central Research Institute (CRI) of the Korea Hydro and Nuclear Power (KHNP) company.

REFERENCES

1. M. C. KIM, et al., “Possibilities and Limitations of Applying Software Reliability Growth Models to Safety Critical Software,” *Nuclear Engineering and Technology*, **39**, 145-148 (2007).
2. H. G. KANG, et al., “Development of a Bayesian Belief Network Model for Software Reliability Quantification of Digital Protection Systems in Nuclear Power Plants,” *Annals of Nuclear Energy*, **120**, 62-73 (2018).
3. H. G. KANG, et al., “Input-profile-based Software Failure Probability Quantification for Safety Signal Generation Systems,” *Reliability Engineering & System Safety*, **94**, 1542-1546 (2009).
4. T.-L. CHU, et al., “Development of a Statistical Testing Approach for Quantifying Safety-Related Digital System on Demand Failure Probability,” NUREG/CR-7234, U.S. NRC (2017).
5. C. BARRETT et al., *Satisfiability Modulo Theories*, In Handbook of Model Checking, Springer, Cham (2008).
6. IEC, “Programmable Controllers - Part 3: Programming Languages,” IEC 61131-3, IEC (1993).
7. J. YOO, et al., “Verification of PLC Programs written in FBD with VIS,” *Nuclear Engineering and Technology*, **41**, 79–90 (2009).
8. L. D. MOURA, et al., “Z3: An Efficient SMT Solver,” *In International conference on Tools and Algorithms for the Construction and Analysis of Systems*, Berlin, Heidelberg, Springer (2008).
9. K. C. KWON et al., “Technical Review on the Localized Digital Instrumentation and Control Systems,” *Nuclear Engineering and Technology*, **41**, 447-454 (2009).
10. J. G. CHOI et al., “Development of RPS Trip Logic based on PLD Technology,” *Nuclear Engineering and Technology*, **44**, 697-708 (2012).
11. Y. H. GOO et al., “BP SDS for Reactor Protection System,” KNICS-RPS-SDS231 Rev. 3, Doosan Heavy Industries and Construction Co., Ltd (2008).
12. S. H. LEE, et al., “Development of Simulation-based Testing Environment for Safety-critical Software,” *Nuclear Engineering and Technology*, **50**, 570-581 (2018).