

# 기능 블록 다이어그램 프로그램에 대한 커플링 효과 가정 분석\*

Lingjun Liu<sup>o</sup>, 지은경, 배두환  
한국과학기술원  
{riensha, ekjee, bae}@se.kaist.ac.kr

## Analysis of coupling effect hypothesis for function block diagram programs\*

Lingjun Liu<sup>o</sup>, Eunkyong Jee, Doo-Hwan Bae  
School of Computing, Korea Advanced Institute of Science and Technology (KAIST)

### Abstract

Testing for Function Block Diagram (FBD) programs has become an important issue since FBD programs have widely been used in safety-critical systems. Mutation testing is considered effective in fault detection. In mutation testing, the coupling effect hypothesis indicates that test data that can detect simple errors can also detect more complex errors. Whether FBD programs hold on this coupling effect hypothesis has not been investigated. We conducted experiments to discuss the coupling effect hypothesis on FBD programs using higher-order mutants. It is experimentally shown that the subject FBD program holds the coupling effect hypothesis.

### 1. Introduction

Function Block Diagram (FBD) is one of the standard Programmable Logic Controller (PLC) programming language defined in IEC 61131-3 [1]. As PLCs have been used to implement safety-critical systems, testing of FBD programs has become an important issue.

Mutation testing is an error-based technique and is effective to measure the fault detection capability of test data. In mutation testing, the coupling effect hypothesis means that test data that can distinguish between the correct program and the faulty programs by simple errors also have the ability to detect more complex errors [2]. Because the mutation testing study of the FBD programs is in its infancy compared to other programming languages, the question of whether the coupling effect hypothesis is still valid in the FBD program has not been answered.

We conducted experiments to investigate the coupling effect hypothesis on FBD programs. We generated higher-order mutants from an industrial example program and prepared test suites that have the ability to detect first-order mutants. The experimental results indicate that the coupling effect hypothesis holds on FBD programs.

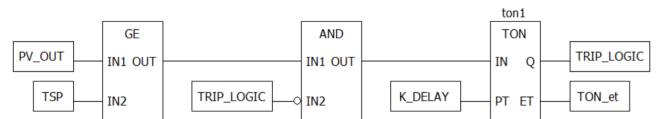


Figure 1. An example FBD program

### 2. Background and related work

#### 2.1. FBD program

FBD is a data flow language with graphical notations. FBD programs are executed on PLC cyclically within a scan time. An FBD program is composed of functions and/or function blocks. Functions deliver outputs by the input values of current cycle while function blocks produce outputs by considering not only the input values but also the data recorded in internal memory. Due to the data in the internal memory, function blocks may return different output results with the same input values.

Figure 1 shows an example FBD program. It consists of a *GE* (Greater or Equal to) function, an *AND* function, and a *TON* (ON-delay Timer) function block. For *TON* function blocks, the output *Q* is *true* when the input *IN* has been *true* for a pulse time (*PT*). The output *ET* represents the elapsed time that the input *IN* has been *true*. In this example, the output *TRIP\_LOGIC* is *true* when the output of the *AND* block has been *true* for a certain time, which is the input *K\_DELAY*. The output *TON\_et* denotes the duration that the output of the *AND* block has been *true*.

#### 2.2. Related work

Existing studies confirmed the validity of the coupling effect hypothesis on different programming languages [3, 4] and logical faults [5]. Offutt [3] conducted experiments to

\*This research was partly supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2019R111A1A01062946), Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2015-0-00250, (SW Star Lab) Software R&D for Model-based Analysis and Verification of Higher-order Large Complex System), and the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2020-2020-0-01795) supervised by the IITP.

investigate the coupling effect hypothesis on Fortran programs using second-order and third-order mutants. The experimental results show that test data developed for first-order mutants killed a higher percentage of mutants when applied to both cases of second-order mutants and third-order mutants. Langdon et al. [4] reformulated mutation testing as a multi-objective search problem, and their goal is to find higher-order mutants that are hard to be killed and syntactically similar to the program under test. For three C benchmark programs from SIR, Monte Carlo sampling is used to produce random higher-order mutants. The tests that kill first-order mutants detected 98 to over 99 % of random higher-order mutants. Kapoor [5] did formal analysis of the coupling effect hypothesis for Boolean formulas with logical faults. Boolean logic formulas are regarded within the context of specification-based testing, control-flow, and domain testing. The coupling effect hypothesis is proved to hold on a large number of logical fault classes.

FBD programming language is different from general procedural languages. There can be non-executed code for procedural languages. However, FBD programs do not have explicit branches. All the blocks in a program are executed every time. That means FBD programs always reach 100% statement coverage. Furthermore, FBD is a graphical language, so the frequent fault type also differs from procedural languages. Nevertheless, no attempts have been made to FBD programming language yet.

### 3. Experimental design

#### 3.1. Subject program

Our subject program is a module in the Bistable Processor (BP) system which is a part of the Reactor Protection System (RPS). RPS is developed in the Korea Nuclear Instrumentation and Control System (KNICS) project [6]. Our subject program is one of trip decision modules in the BP system: Fix Falling Trip Decision module (FFTD). The FFTD program is composed of 29 blocks, 12 input variables, and 8 output variables. The blocks used in the FFTD program are listed as follows: 9 logic blocks, 6 comparison blocks, 8 selection blocks, 2 timer blocks, 4 arithmetic blocks.

#### 3.2. Mutant generation

We use the tool called MuGenFBD [7], which can generate first-order mutants for FBD programs with the mutation operator set defined in Jee et al.'s work [8]. The mutation operator set can be categorized into four types: 10 Block Replacement operators, Constant Value Replacement (CVR) operator, Inverter Insertion or Deletion (IID) operator, and Switched Inputs (SWI) operator. The block replacement operators are as follows: Conversion Block Replacement (ConBR), Numerical Block Replacement (NBR), Arithmetic Block Replacement (ABR), Logic Block Replacement (LBR), Selection Block Replacement (SBR), Comparison Block

---

#### Algorithm 1: Second-order mutant generation

---

```

Input: mutantList – information list of first-order mutants
          P – subject program
for i = 1 → mutantList.length do
  for j = i+1 → mutantList.length do
    if mutantList[i].blockID = mutantList[j].blockID and
      mutantList[i].mutationOp = mutantList[j].mutationOp then
      continue
    end if
    P' = mutation(P, mutantList[i], mutantList[j])
    output(P') /* produce a second-order mutant*/
  end for
end for

```

---

Replacement (CBR), Bistable element Block Replacement (BBR), Edge-detection Block Replacement (EBR), Counter Block Replacement (CouBR), and Timer Block Replacement (TBR). The mutation operator set covers most types of defined functions and function blocks. MuGenFBD generated 133 first-order mutants from the FFTD program. These first-order mutants are used for test suite generation.

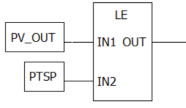
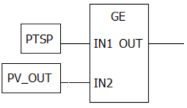
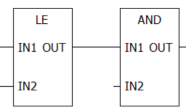
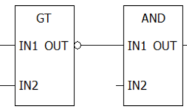
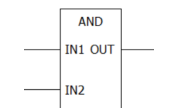
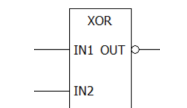
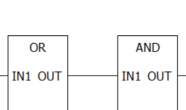
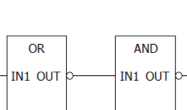
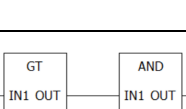
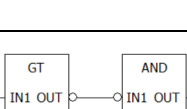
The tool MuGenFBD is slightly modified to generate second-order mutants by applying mutation operators twice. For the subject program, we generated second-order mutants that contained all possible combinations of two mutations. The procedure of generating second-order mutants is described in Algorithm 1.

#### 3.3. Test suite generation

Mutation testing provides test data with a mutation adequacy criterion, which is to achieve a 100% mutation score. The mutation score is calculated as the number of killed mutants divided by the total number of non-equivalent mutants. A mutant is killed when the test case leads to that the output values of the subject program are different from those of the mutant. In this study, we consider whether a mutant is strongly killed or not. Thus, we compared the output sequences of the mutant and subject program after executing both programs.

Not only can mutation testing be used to assess test data, but it can also be used to generate mutation-based test suites. The goal of mutation-based test suites is to achieve a 100% mutation score. The first-order mutants are used to generate mutation-based test suites. For mutation-based test suite generation, we used our developed tool which is designed to combine the subject program and each mutant repeatedly and to search for test input data of each mutant by utilizing the Yices SMT solver. Thus, each test case in the test suite was developed to distinguish the subject program and a mutant. The tool generated 121 test cases for the FFTD program. We checked the mutation score of the test set, and the test set earned a 100% score for first-order mutants.

**Table 1.** Live second-order mutants

Higher-order mutant type	Original program	Mutated program
HF1: Applying CBR and SWI operators to the same block		
HF2: Applying CBR and IID operators to the same block		
HF3: Applying LBR and IID operators to the same block		
HF4: Applying IID operator to the outputs of two connected blocks		
HF5: Applying IID operators to the same connection		

**4. Evaluation**

**4.1. Experimental results**

We generated total 7124 second-order mutants from the FFTD program by applying mutation twice. There were 32 mutants that the mutation-based test suite could not kill, and only four mutants were non-equivalent mutants. The mutation-based test suite achieved up to a 99.94% mutation score. As shown in Table 1, we categorized non-killed mutants into five types. We defined HF as faults caused by higher-order mutation. HF1 contained the application of CBR and SWI mutation operators on the same block. HF2 contained the change by combining CBR and IID mutation operators on the same block. HF3 contained the combination of LBR and IID mutation operators on the same block. Furthermore, HF4 and HF5 contained the changes obtained by applying the IID mutation operator twice. Except for HF3, all types of faults that the mutation-based test suite could not detect were equivalent mutants. In HF3, as shown in Table 1, the LBR mutation operator changed the AND function to the XOR function, and the IID mutation operator injected an inverter on the output of the function. In addition to equivalent mutants, the mutation-based test suite could not detect HF3. While we found some corner cases that cannot be killed by the test suite designed to detect simple faults, we observed that the coupling effect hypothesis generally holds in the case of FBD programs.

**4.2. Threats to validity**

In the experiment, mutants are generated based on the mutation operator set defined in Jee et al.'s work [8]. It cannot be claimed that the mutation operator set is considered as comprehensive. However, the mutation operator set can cover all the functions and function blocks included in our subject.

Our subject program does not include all groups of functions and function blocks, but it is a real example from the industrial case. Considering more various subject programs will form the basis of our future work.

**5. Conclusion**

This paper investigated whether the coupling effect hypothesis holds on FBD programs. The experiments were conducted to check if the test suite that killed first-order mutants is also effective for detecting higher-order mutants. Although our experimental results are preliminary, results show that the test suite can kill up to 99.94% of second-order mutants. Generally, the subject program holds the coupling effect hypothesis even though we found some corner cases in our experiment. In future work, we plan to conduct experiments on more subjects to generalize the results. Also, we plan to generate third-order or even higher-order mutants to investigate the coupling effect on FBD programs.

**References**

- [1] International Electrotechnical Commission (IEC), "IEC61131-3: International Standard for Programmable Controllers - Part 3: Programming Languages," 2013
- [2] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, Vol. 11, No. 4, pp. 34-41, 1978
- [3] A. J. Offutt, "Investigations of the software testing coupling effect," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 1, No. 1, pp. 5-20, 1992
- [4] W. B. Langdon, M. Harman, and Y. Jia, "Multi Objective Higher Order Mutation Testing with Genetic Programming," *2009 Testing: Academic and Industrial Conference-Practice and Research Techniques*. IEEE, pp. 21-29, 2009
- [5] K. Kapoor, "Formal analysis of coupling hypothesis for logical faults," *Innovations in Systems and Software Engineering*, Vol. 2, No. 2, pp. 80-87, 2006
- [6] Doosan Heavy Industry & Construction, "Software design specification for the bistable processor of the reactor protection system," KNICS.RPS.SDS231-01, Rev.01, 2006 (In Korean)
- [7] L. Liu, E. Jee, and D. H. Bae, "Automated mutant generation for function block diagram programs," *Proceedings of the 22nd Korea Conference on Software Engineering (KCSE)*, pp. 154-155, 2020
- [8] E. Jee, J. Song, and D. H. Bae, "Definition and Application of Mutation Operator Extensions for FBD Programs," *Journal of KIISE: Transactions on Computing Practices*, Vol. 24, No. 1, pp. 589-595, 2018 (in Korean)