Software: Testing, Verification, and Reliability (STVR) 2022

MuFBDTester: A mutation-based test sequence generator for FBD programs implementing nuclear power plant SW

Lingjun Liu[1], Eunkyoung Jee[2], Doo-Hwan Bae[2]

[1] SURESOFT (슈어소프트테크)
[2] School of Computing, KAIST, Daejeon, Republic of Korea

2023.02.09

Introduction

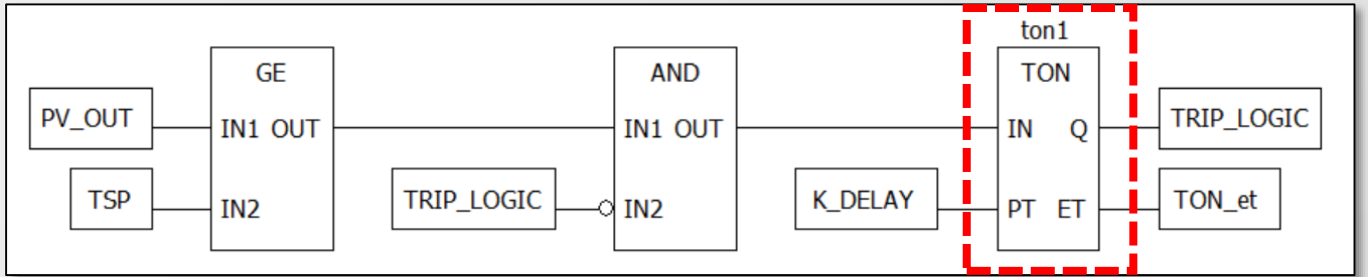
- Function Block Diagram (FBD)
- Mutation testing
- Motivation & Goal
- Related work

Function Block Diagram (FBD)

- **A standard programming language for Programmable Logic Controller (PLC), defined in IEC61131**
  - Graphical language for describing data flows through blocks
  - Consist of functions and function blocks
    - Function blocks deliver outputs based on input variables and the values stored in the internal memory.
  - Executed cyclically in a scan time

Inputs: PV_OUT, TSP, TRIP_LOGIC, K_DELAY
Outputs: TRIP_LOGIC, TON_et

Functions: GE, AND
Function blocks: TON



FBD programs are widely used to implement safety-critical systems such as reactor protection systems, so their testing is essential.

Existing studies on automated test generation for FBD programs

- **Existing studies focus on generating tests based on structural coverage of FBD program's test models.**
  - Converting FBD program into an intermediate model[1,2,3] such as a timed automata
  - Using FBD program as a test model[4]

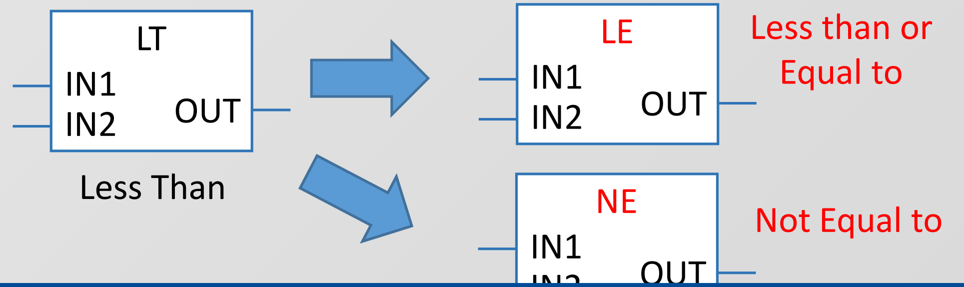
Fault type [5]
Comparison block misplac
Inverter omission
…

Coverage level is not strongly correlated to fault detection. [6]

Test Suite achieving 95% coverage level

TC1
LOGIC = true
X = 0
Y = -1
…

Weak at detecting
Comparison block misplacement [7]

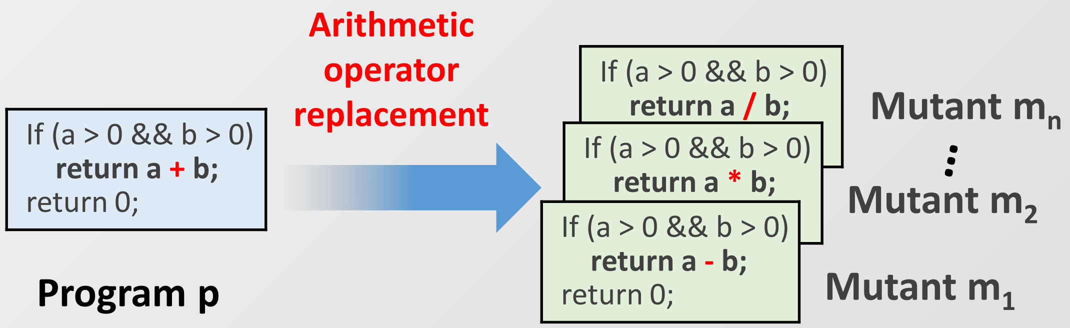


As target systems are mainly safety-critical, a guarantee of fault detection is needed.

Verification and Validation Workshops (ICSTW), pp. 158-167.
[4] Song, J., Jee, E., & Bae, D. H. (2018). FBDTester 2.0: Automated test sequence generation for FBD programs with internal memory states. Science of Computer Programming, 163(1), 115-137.
[5] Oh, Y., Yoo, J., Cha, S., & Son, H. S. (2005). Software safety analysis of function block diagrams using fault trees. Reliability Engineering & System Safety, 88(3), 215-228.
[6] Inozemtseva, L., & Holmes, R. (2014). Coverage is not strongly correlated with test suite effectiveness. In Proceedings of the 36th international conference on software engineering, pp. 435-445.
[7] Shin, D., Jee, E., & Bae, D. H. (2016). Comprehensive analysis of FBD test coverage criteria using mutants. Software & Systems Modeling, 15(3), 631-645.

Mutation testing

- **An error-based testing technique to detect specific types of errors expressed by mutation operators**
- **Application**
  - Evaluate the fault detection effectiveness of test sets
  - Generate mutation-adequate test suite to achieve 100% mutation score
- **Empirical studies on other languages showed that mutation-based tests detected more faults than structural coverage criteria.[1,2,3]**



$$mutation\ score = \frac{\#\ of\ mutants\ detectecd\ by\ a\ test\ suite}{\#\ of\ nonequivalent\ mutants}$$

[1] Li, N., Praphamontripong, U., & Offutt, J. (2009, April). An experimental comparison of four unit test criteria: Mutation, edge-pair, all-uses and prime path coverage. In 2009 International Conference on Software Testing, Verification, and Validation Workshops (pp. 220-229). IEEE.
[2] Offutt, A. J., Pan, J., Tewary, K., & Zhang, T. (1996). An experimental evaluation of data flow and mutation testing. Software: Practice and Experience, 26(2), 165-176.
[3] Frankl, P. G., Weiss, S. N., & Hu, C. (1997). All-uses vs mutation testing: an experimental comparison of effectiveness. *Journal of Systems and Software*, *38*(3), 235-253.

Motivation & Goal

Motivation

- Existing studies on FBD test generation mainly focus on achieving certain structural coverage criteria.
  - Achieving high coverage level can possibly detect errors, but it cannot provide assurance of fault detection. [1]
- Mutation testing is highly effective to identify possible errors expressed by mutation operators.

Goal

- To propose an automated mutation-adequate test generation approach for FBD programs
  - To guarantee detection of various types of faults in Nuclear Power Plant (NPP) SW

[1] Inozemtseva, L., & Holmes, R. (2014). Coverage is not strongly correlated with test suite effectiveness. In Proceedings of the 36th international conference on software engineering, pp. 435-445.

Related work – test generation for FBD programs

1st Author	Type	Testing objectives	Pros	Cons
Lahtinen [1]	**Structural coverage**	Basic Coverage (BC), Input Condition Coverage (ICC), Complex Condition Coverage (CCC)	• Lower time cost	• Lower fault detection effectiveness
Wu [2,3]	**Structural coverage**	FB-Path Complete Condition (FPCC) Test Coverage		
Enoiu [4, 5]	**Structural coverage**	Decision Coverage (DC), Condition Coverage (CC), Modified Condition/Decision Coverage (MC/DC)		
Song and Jee [6, 7]	**Structural coverage**	BC, ICC, CCC		
This study	**Mutation-based**	Mutation score	• Higher fault detection effectiveness • Providing assurance	• Higher time cost due to large amount of mutants

1st Author	Type	Test model	Support engine	Number of mutation operator	Equivalent mutant detection
Enoiu [8]	**Mutation-based**	Timed automata	Model checker	6	X
This study	**Mutation-based**	FBD itself	SMT solver	13	O

SELAB KAIST

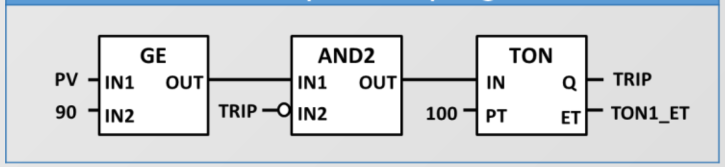
*reference [1-8] is in appendix slide

Background

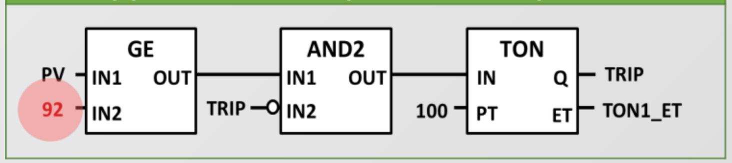
Mutation operator set for FBD programs[1]

	Mutation Operators	
	CVR	Constant Value Replacement
	IID	Inverter Insertion or Deletion
	SWI	SWitched Inputs
Function	ABR	Arithmetic Block Replacement
Function	CBR	Comparison Block Replacement
Function	LBR	Logic Block Replacement
Function	SBR	Selection Block Replacement
Function	NBR	Numerical Block Replacement
Function	ConBR	Converter Block Replacement
Function Block	BBR	Bistable element Block Replacement
Function Block	EBR	Edge detection Block Replacement
Function Block	CouBR	Counter Block Replacement
Function Block	TBR	Timer Block Replacement

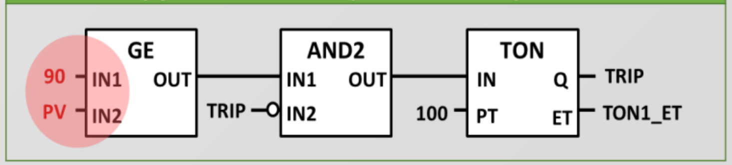
An example FBD program



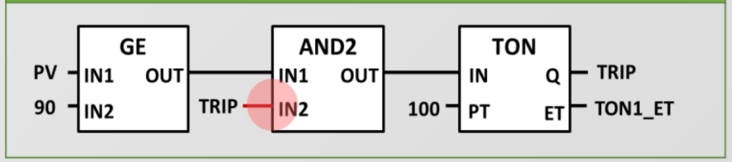
Application example of CVR operator



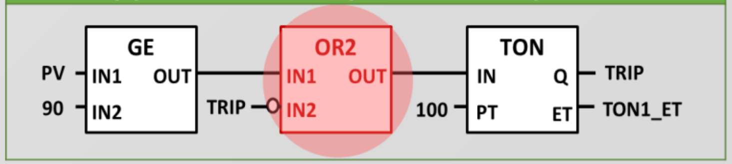
Application example of SWI operator



Application example of IID operator



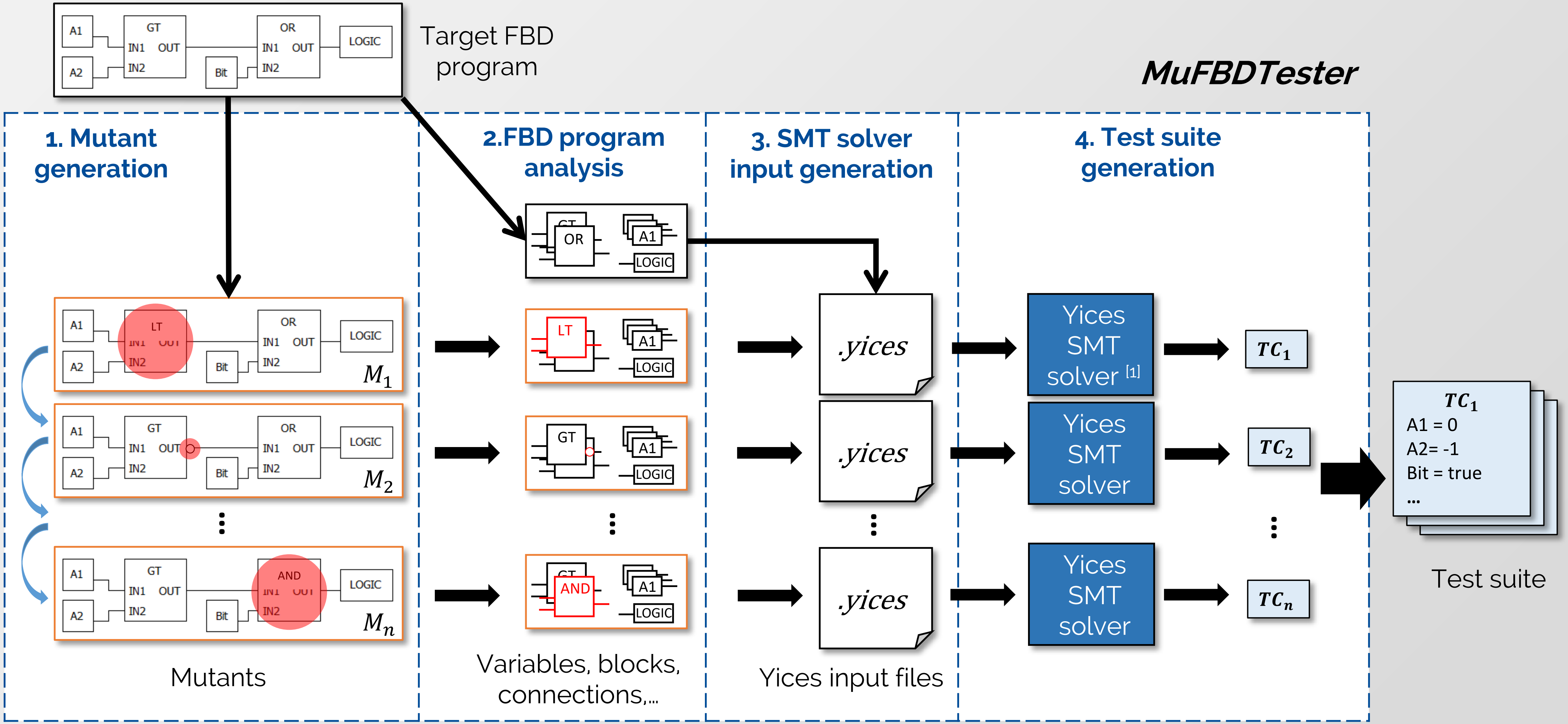
Application example of LBR operator



[1] 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. 11, pp. 589-595, 2018 (in Korean)

Mutation-adequate test sequence generation

Overall approach

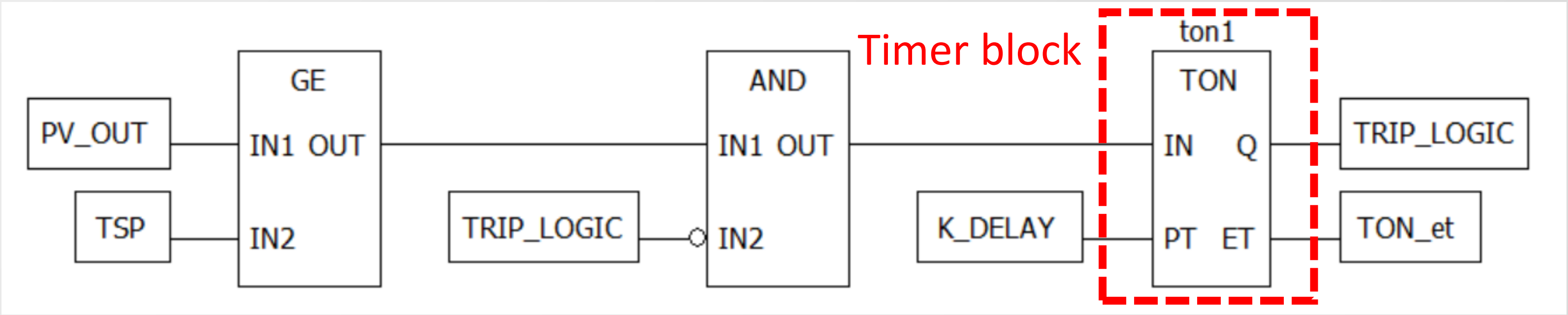


[1] Dutertre B, De Moura L. The yices smt solver. Tool paper at http://yices. csl. sri. com/tool-paper. pdf 2006; 2(2):1–2.

Step 2. FBD program analysis

- **Extract structural information including variables, blocks, and connections between blocks and variables**
- **Test sequence is needed to manage internal memory states when the program includes function blocks.**
  - Determine test sequence length by **Min**imum **iter**ation number (MinIter) suggested in Song et al's work. [1]
    - $Timer\ blocks'\ MinIter\ =\ (Preset\ time\ (PT)\ /\ Scan\ time)\ +\ 1$

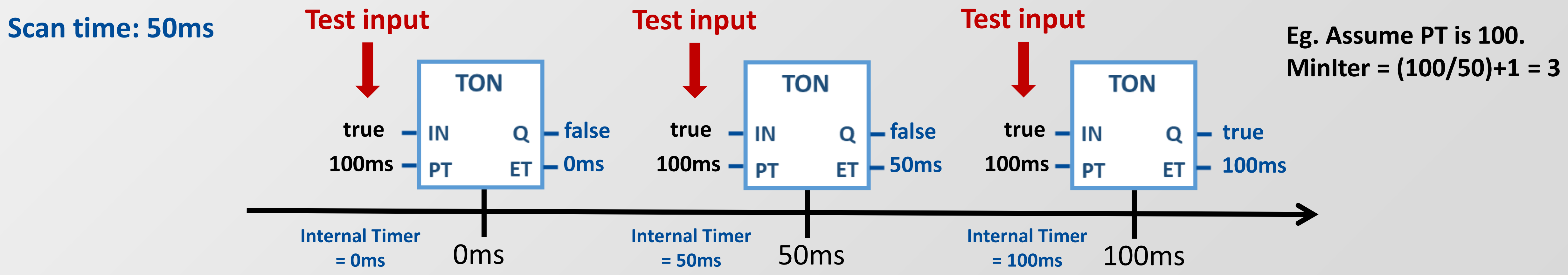
Scan time: 50ms



[1] Song, J., Jee, E., & Bae, D. H. (2018). FBDTester 2.0: Automated test sequence generation for FBD programs with internal memory states. Science of Computer Programming, 163(1), 115-137.

Step 2. FBD program analysis

- **Extract structural information including variables, blocks, and connections between blocks and variables**
- **Test sequence is needed to manage internal memory states when the program includes function blocks.**
  - Determine test sequence length by **Min**imum **iter**ation number (MinIter) suggested in Song et al's work. [1]
    - $Timer\ blocks'\ MinIter = (Preset\ time\ (PT)\ /\ Scan\ time) + 1$

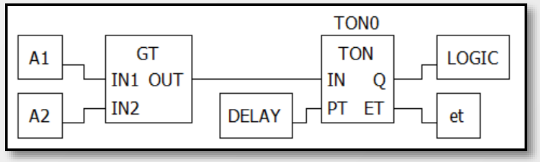


[1] Song, J., Jee, E., & Bae, D. H. (2018). FBDTester 2.0: Automated test sequence generation for FBD programs with internal memory states. Science of Computer Programming, 163(1), 115-137.

Step 3. SMT solver input generation

3-1. Define Target program

- Define inputs
- Define operation of every block
- Unfold the program MinIter cycles



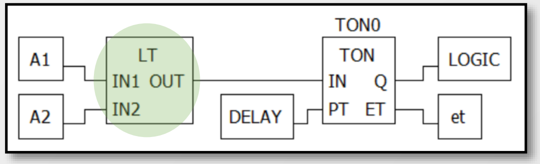
Inputs

Block operations

```
;; 2 cycles ago
(define A1_t2::int )
(define A2_t2::int )
(define GT_out_t2::bool (> A1_t2 A2_t2))
(define et_t2::int (if GT_out_t2 (if (< et_t3 DELAY) (+ et_t3 SCAN_TIME) DELAY) 0))
(define LOGIC_t2::bool (and GT_out_t2 (>= et_t3 DELAY)))
;; 1 cycle ago
(define A1_t1::int )
… ;; current cycle
(define A1::int )
(define A2::int )
(define GT_out::bool (> A1 A2))
(define et::int (if GT_out (if (< et_t1 DELAY) (+ et_t1 SCAN_TIME) DELAY) 0))
(define LOGIC::bool (and GT_out (>= et_t1 DELAY)))
```

3-2. Define the Mutant

- Use the same inputs of target program



CBR(Comparison Block Replacement) GT -> LT

```
(define M_et_t3::int et_t3)
;; 2 cycles ago
(define M_LT_out_t2::bool (< A1_t2 A2_t2))
(define M_et_t2::int (if M_LT_out_t2 (if (< M_et_t3 DELAY) (+ M_et_t3 SCAN_TIME) DELAY) 0))
(define M_LOGIC_t2::bool (and M_LT_out_t2 (>= M_et_t3 DELAY)))
;; 1 cycle ago… ;; current cycle
(define M_LT_out::bool (< A1 A2))
(define M_et::int (if M_LT_out (if (< M_et_t1 DELAY) (+ M_et_t1 SCAN_TIME) DELAY) 0))
(define M_LOGIC::bool (and M_LT_out (>= M_et_t1 DELAY)))
```

3-3. Assert Test requirement

- To distinguish between outputs of target program and the mutant

```
(assert (or(not(= LOGIC M_LOGIC)) (not(= et M_et)) ))
```

3-4. Execute SMT solver command

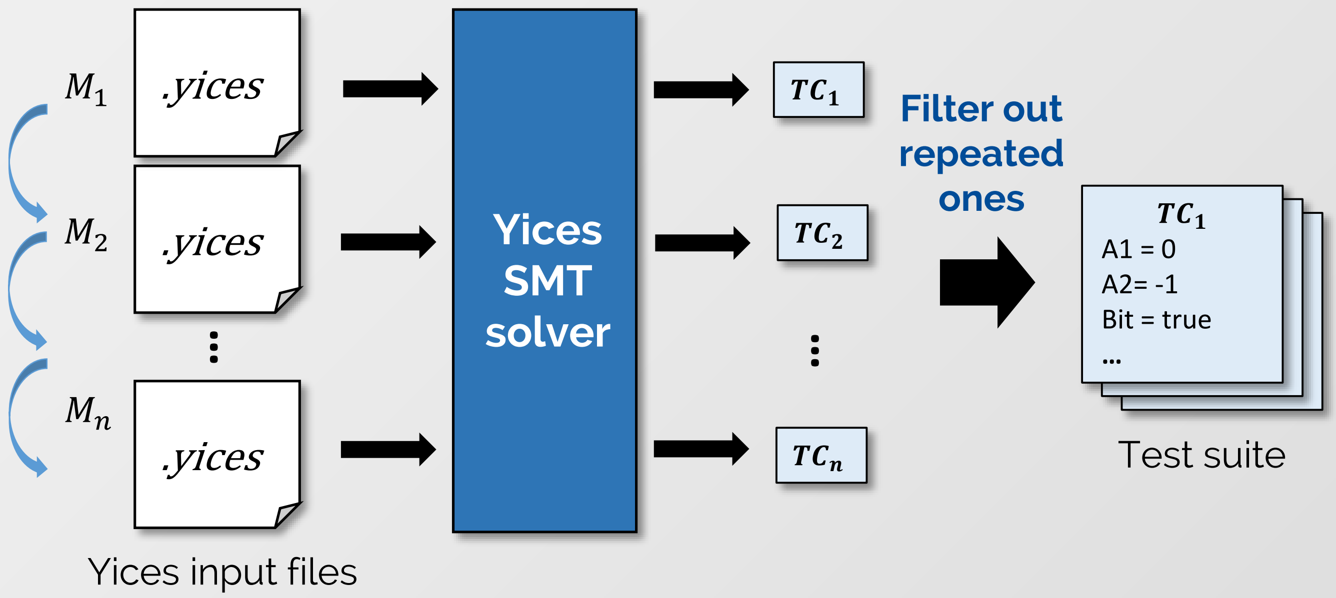
- Check whether the test requirement is satisfiable or not

```
(check)
```

SELAB KAIST

Step 4. Test suite generation

- Execute Yices input files on Yices SMT solver
- Collect results of Yices SMT solver
- Make a union set of gathered test data from SMT solver



Possible results of Yices SMT solver

1. sat

➔ Output test data

2. unsat

➔ The mutant is an equivalent mutant.

➔ **Our approach can achieve 100% mutation scores on generated mutants**

Evaluation

Research Questions

- RQ1: Does the mutation-adequate test suite exhibit better **fault detection effectiveness** than that exhibited by the test suites conforming to structural coverage criteria?

- RQ2: Does MuFBDTester exhibit better **efficiency** than that exhibited by other related tools?
  - **FBDTester 2.0** (from Song et al.[1]):
    - The most advanced FBD structural coverage-based test generation tool
  - **Enoiu et al.'s approach**[2]:
    - Mutation-based FBD test generation using model checking

SELAB KAIST

[1] Song, J., Jee, E., & Bae, D. H. (2018). FBDTester 2.0: Automated test sequence generation for FBD programs with internal memory states. Science of Computer Programming, 163(1), 115-137.
[2] Enoiu, E. P., Sundmark, D., Čaušević, A., Feldt, R., & Pettersson, P. (2016, October). Mutation-based test generation for plc embedded software using model checking. In IFIP International Conference on Testing Software and Systems (pp. 155-171). Springer, Cham.

Experiment Subjects

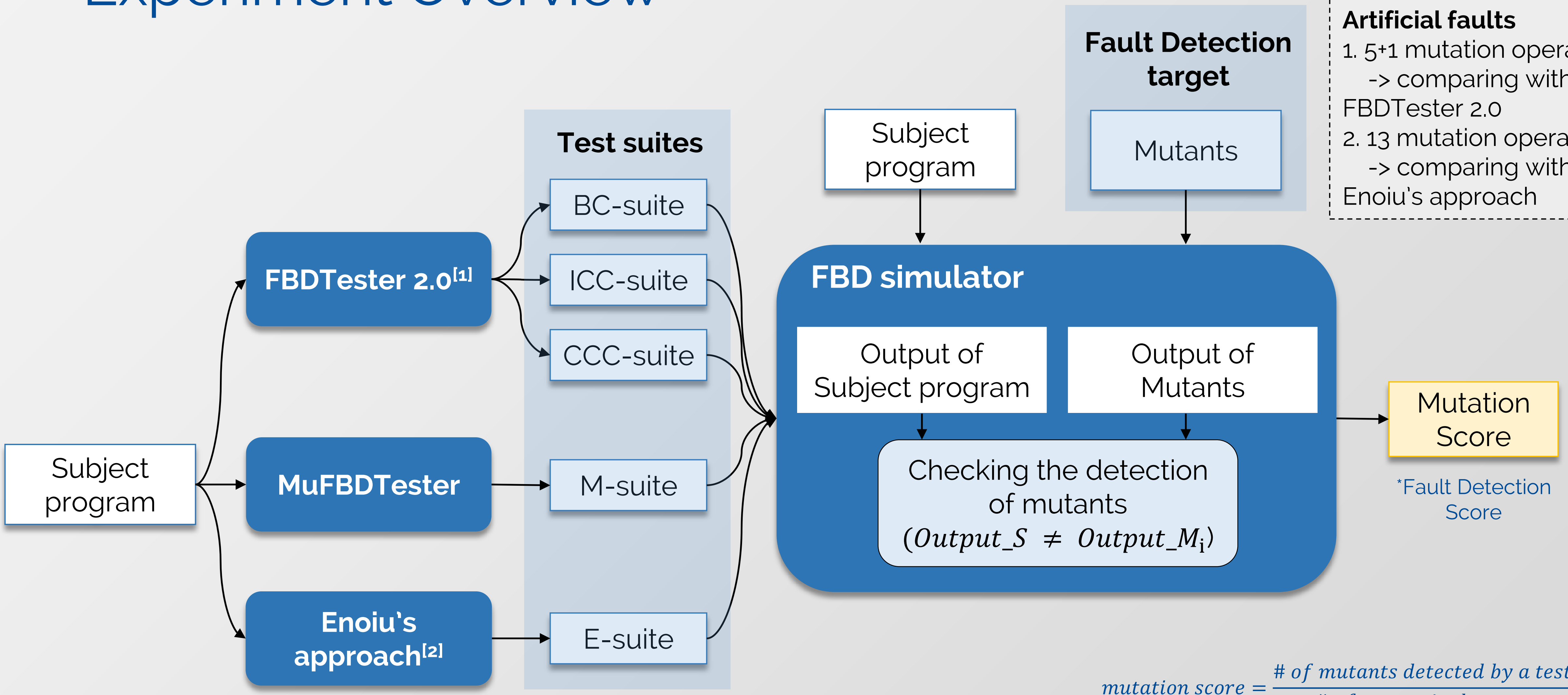
- **6 industrial programs from reactor protection system in KNICS project**
  - KNICS: Korean Nuclear Instrumentation and Control System
- **3 small-sized programs**

Subject	#blocks	#function blocks	#inputs	#outputs
simTRIP	3	Timer: 1	3	2
simGRAVEL	3	Timer: 1, Counter: 1	3	4
LAUNCHER	4	Edge detection: 1, Bistable element: 1	2	1
FFTD	29	Timer: 2	12	8
FRTD	29	Timer: 2	12	8
VFTD	44	Timer: 2	14	8
VRTD	44	Timer: 2	17	8
MFTD	47	Timer: 2	21	8
HB	19	-	6	1

FFTD: Fixed-Falling Trip Decision
VFTD: Variable-rate-Falling Trip Decision
MFTD: Manual-reset-Falling Trip Decision

FRTD: Fixed-Rising Trip Decision
VRTD: Variable-rate-Rising Trip Decision
HB: HeartBeat monitoring

Experiment Overview



Artificial faults

1. 5+1 mutation operators -> comparing with FBDTester 2.0
2. 13 mutation operators -> comparing with Enoiu's approach

*Fault Detection Score

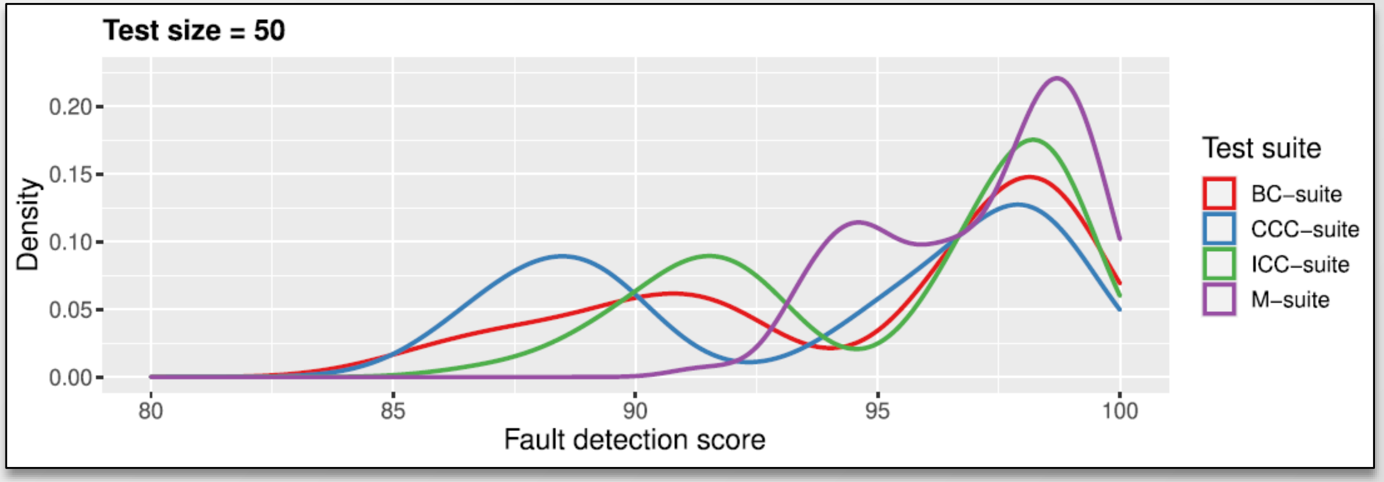
$$mutation\ score = \frac{\#\ of\ mutants\ detected\ by\ a\ test\ suite}{\#\ of\ nonequivalent\ mutants}$$

[1] Song, J., Jee, E., & Bae, D. H. (2018). FBDTester 2.0: Automated test sequence generation for FBD programs with internal memory states. Science of Computer Programming, 163(1), 115-137.
[2] Enoiu, E. P., Sundmark, D., Čaušević, A., Feldt, R., & Pettersson, P. (2016, October). Mutation-based test generation for plc embedded software using model checking. In IFIP International Conference on Testing Software and Systems (pp. 155-171). Springer, Cham.

RQ1. Comparing effectiveness of MuFBDTester and FBDTester 2.0 (1/2)

Experiment design

- Subjects: industrial programs (FFTD, FRTD, VFTD, VRTD, MFTD)
- Test suite: (250 test sets for each BC, ICC, CCC, and M-suite)
  - Took 5 sample test sets with same size for each test suite (10 runs of both tools)
- Fault detection target:
  - 1st order mutants: using 5+1 mutation operators[1], different from those used for mutation-based test generation



Statistical Comparison
- **The Wilcoxon matched-pairs signed rank test**
  - Paired design
    - different test sets executed on the same program
  - Not following normal distribution

SELAB KAIST

[1] Enoiu, E. P., Sundmark, D., Čaušević, A., Feldt, R., & Pettersson, P. (2016, October). Mutation-based test generation for plc embedded software using model checking. In IFIP International Conference on Testing Software and Systems (pp. 155-171). Springer, Cham.

RQ1. Comparing effectiveness of MuFBDTester and FBDTester 2.0 (2/2)

Statistical Comparison Results

- p-value: statistical significance, effect size: practical significance
- **With restricted test sizes, the results indicate that M-suites are statistically and practically superior to BC-, ICC- and CCC-suites in view of detecting artificial faults.**

*p-value < 0.01

*Effect size is large (≥ 0.5).

Test size = 50				
Paired comparisons (winner > loser)	**Agree?**	***p*-value**	**95% confidence interval**	**Effect size**
M-suite > BC-suite	Yes.	7.82e-36	[2.10, 2.80]	0.79
M-suite > ICC-suite	Yes.	7.27e-34	[1.55, 2.34]	0.76
M-suite > CCC-suite	Yes.	4.36e-40	[3.25, 3.93]	0.83

Note: Test size: number of test sequences.

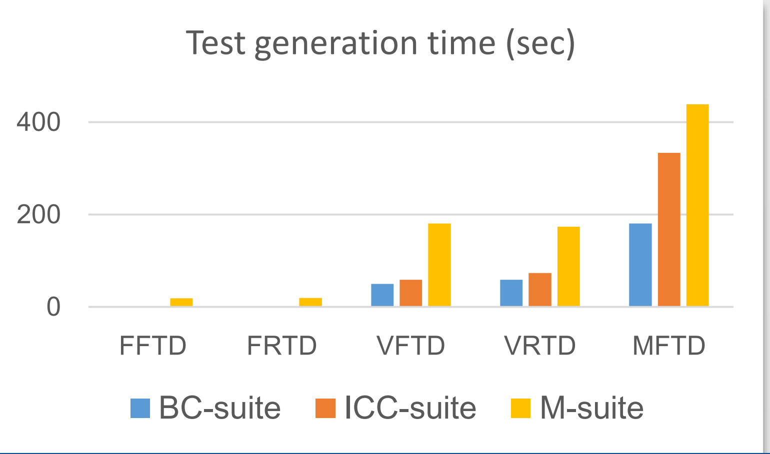
RQ2-1. Comparing efficiency of MuFBDTester and FBDTester 2.0

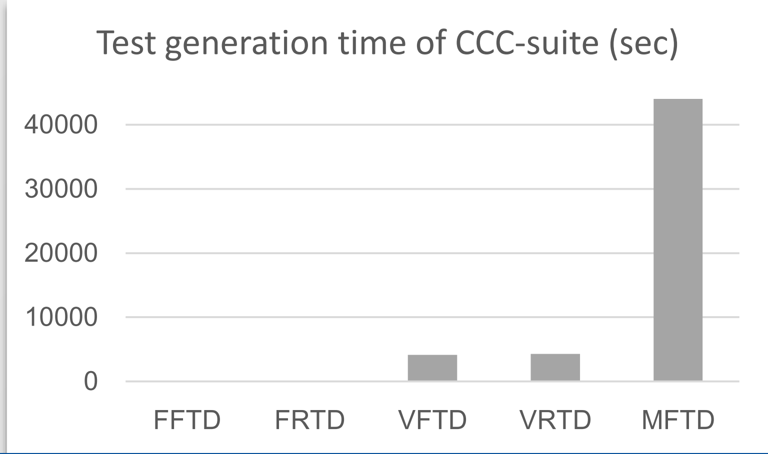
Experiment design

- Compared test generation time between MuFBDTester and FBDTester 2.0

Results

- For VFTD, VRTD, and MFTD modules, the generation time of M-suite were more than that of ICC-suite but less than that of CCC-suite.
- The generation time of CCC-suite increased significantly by up to 12 hours.





Average time per test sequence (sec)	
BC-Suite	1.09
ICC-Suite	1.68
CCC-Suite	20.76
M-Suite	**1.7**

MuFBDTester took a reasonable amount of time to generate highly effective test sequences for complex programs.

RQ2-2. Comparing efficiency of MuFBDTester and other mutation-based approach

Experiment design

- Compared time cost between MuFBDTester and Enoiu's approach
  - Enoiu et al. used UPPAAL model checker to generate mutation-based tests
- Modified MuFBDTester to support the mutation operator set used in Enoiu's work
  - Reduced mutation operators: 13 → 6

Results

- Enoiu's approach cannot scale to the industrial programs due to state explosion.
- Our approach achieved higher mutation scores and reduced 95% of the time to generate 4 times more test input data.

Mutation scores on 2nd order mutants

	simTRIP	LAUNCHER
#non-equivalent mutants	104	151
E-suite	95.1	99.3
M-suite	**100**	**100**

*Second-order mutant is generated from 13 mutation operators proposed from Jee et al.

Test suite generation time and test suite size (sec / total test input length)

	simTRIP	LAUNCHER
E-suite	34.02/11	1.43/3
M-suite	**0.82/40**	**0.83/28**

SELAB KAIST

Conclusion

Conclusion

- **Proposed automated mutation-adequate test generation for FBD**
  - MuFBDTester can provide a certain level of assurance to detect various types of faults in FBD programs.
  - Our approach can detect equivalent mutants automatically.
    - Without suffering from the burden of identifying equivalent mutants manually
  - Experimental results showed MuFBDTester's fault detection effectiveness and efficiency compared to FBDTester2.0 and Enoiu's approach.
- **Future work**
  - Improve mutation operators based on live higher-order mutants
  - Utilize multiple SMT solvers to deal with FBD programs including complex numerical functions
  - Ongoing work: Cost-Effective Test Selection for FBD Programs

Thank You.

Lingjun Liu
riensha@se.kaist.ac.kr

Korea Advanced Institute of Science and Technology (KAIST)

2023.02.09