# 동적 데이터 기반 SW 유닛들 사이의 관련도 메트릭 활용

KAIST SWTV 연구실

김문주

# Research Roadmap on Automated Testing and Debugging

- **Past: Runtime Verification and Model Checking**
  - Process Algebra — Early Ph.D
  - → Runtime Verification — FMSD'04
  - → Robot SW Verification using Esterel — ICSE'05, ICRA'05
  - → Flash Mem Model Checking — Spin'08, FACJ'12
  - → Flash Mem SW Model Checking — ASE '08, TSE'11

- **Current: Auto. Testing and Fault Localization (FL)**
  - Concolic Testing — FSE'11a, SBMF'09
  - → Distributed Concolic Testing — FSE'11b, ICTAC'10
  - Hybrid Algorithm (e.g., w/ Genetic Alg) — ISSRE '11, ICST'14
  - Extended Concolic Application — ASE'13, ICSE'15, ICSE'18, ICST'18, FSE'19
  - Mutation-based Fault Localization — ICST'14, ASE'15, IST'17, TOSEM 19

- **Future: Data driven scalable SW analysis framework**
  - SBFL
  - Mutation-based FL
  - Bug Prediction
  - Concolic Testing
  - Cov-guided Fuzzing
  - Static Analysis
  - SMT Solver
  - ML Engine

*Better Industrial Application*

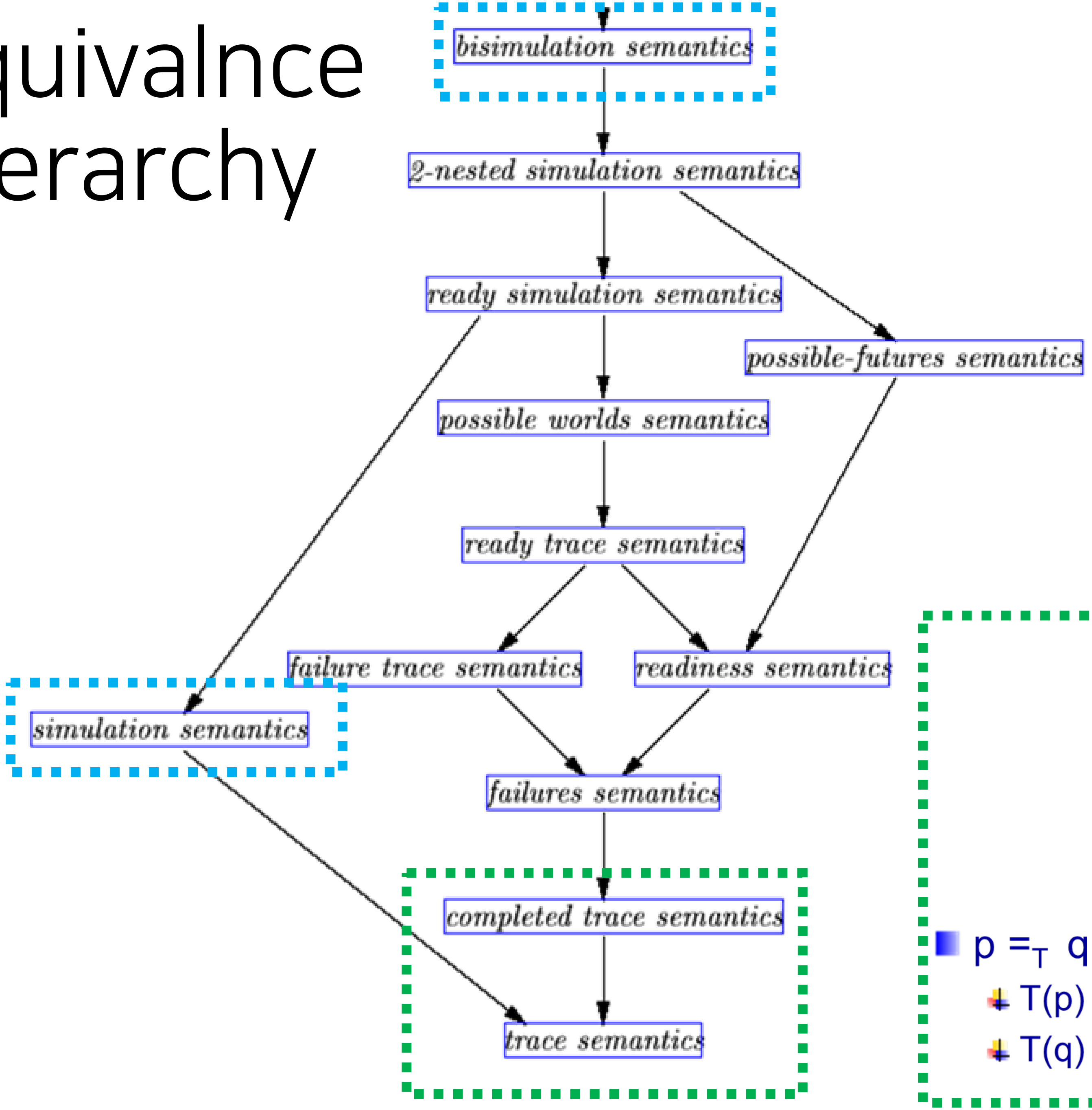
# 철학적 이야기. SW의 본질이란?

- SW analysis (static analysis, dynamic analysis, etc.) is to understand the "essence" of SW

- However, it is very difficult to define the essense of SW
  - Ex1. SW medium itself (e.g., CD/USB)
  - Ex2. Information in the SW medium (e.g., a sequence of bytes)
  - Ex3. Execution information of SW (e.g., execution traces, etc.)

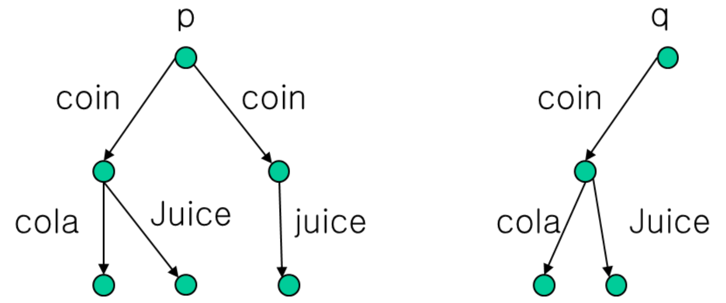
- In process algebra community, the essence of an agent is indirectly defined by "equivalence" (or preorder) between agents.

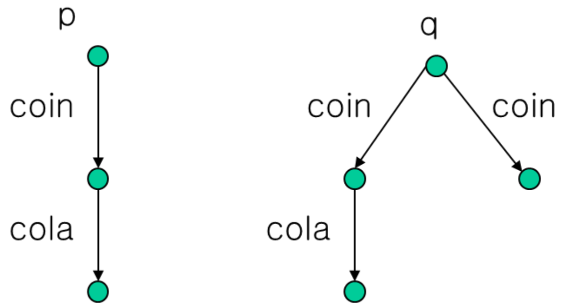
# Equivalnce Hierarchy

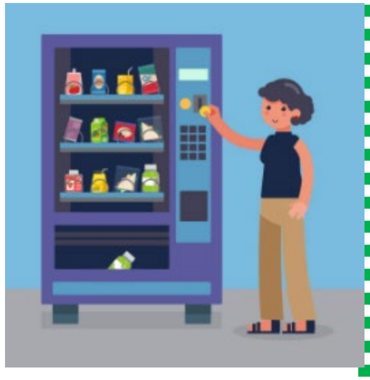



$p \neq_B q$

$p =_S q$



- $p =_T q$
  - T(p) = {coin.cola, coin}
  - T(q) = {coin.cola, coin}
- $p \neq_{CT} q$
  - CT(p) = {coin.cola}
  - CT(q) = {coin.cola, coin}

# Correlation can be Used to Identify Essense of Software

- Correlation between SW components (e.g., functions, classes, modules, etc. ) can be obtained more easily than causal-effect relation between the SW components
- Correlation/grouping of SW components have been studied in SE for the following applications:
  - ex1. How to plan software releases
  - ex2. How to build SW developing team structure
  - ex3. How to organize SW segments for various purposes

# Static Relevance between Functions (1/2)

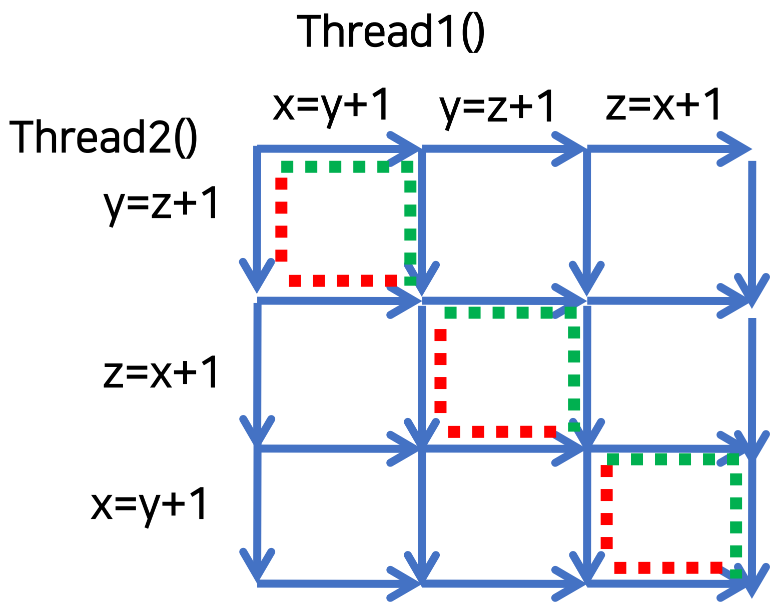
- Until very recently, only static correlation between the SW components have been utilized
  - since researchers preferred stable (and easy to analyze) targets
- For example,
  - Li and Henry [43] proposed a message passing coupling metric which measures the number of method invocations in a class.
  - Chidamber and Kemerer [13] proposed a coupling metric of two classes using the number of accesses of field variables and invocations of the methods of another class.
  - Lee et al. [42] uses the number of method invocations of another class weighted by the number of arguments of the invoked methods.

# Static Relevance between Functions (2/2)

- However, these metrics have limitations to apply to reduce false alarms in automated unit testing (i.e., these metrics are static ones and reports provide too imprecise coupling value)
  - Ex1. The metric using the number of accesses to the common class field variables [13] does not capture the relation constructed by passing arguments.
  - Ex2. Lee et al. [42] considered the number of arguments passed but the number of arguments is often not a good weight because one pointer argument can pass large data structure.

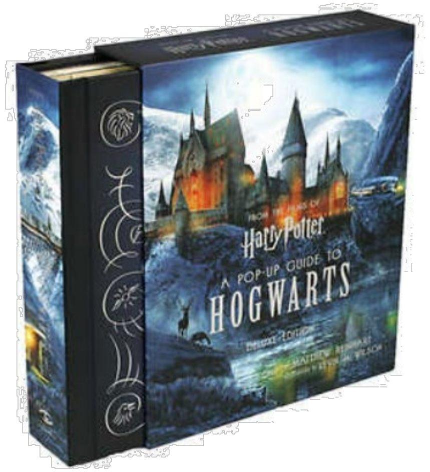
# SW 테스팅 = SW 본질을 찾아가는 과정

- Concurrent programs have very high complexity due to non-deterministic scheduling
- Ex.  int x=0, y=0, z =0;
  - void Thread1() {x=y+1; y=z+1; z= x+1;}
  - Void Thread2() {y=z+1; z=x+1; x=y+1;}
  - Total 20 interleaving scenarios = (3+3)!/(3!x3!)
  - However, only 11 unique outcomes
    - assert(x+y+z > 5)???
    - assert(x+y+z < 15)???

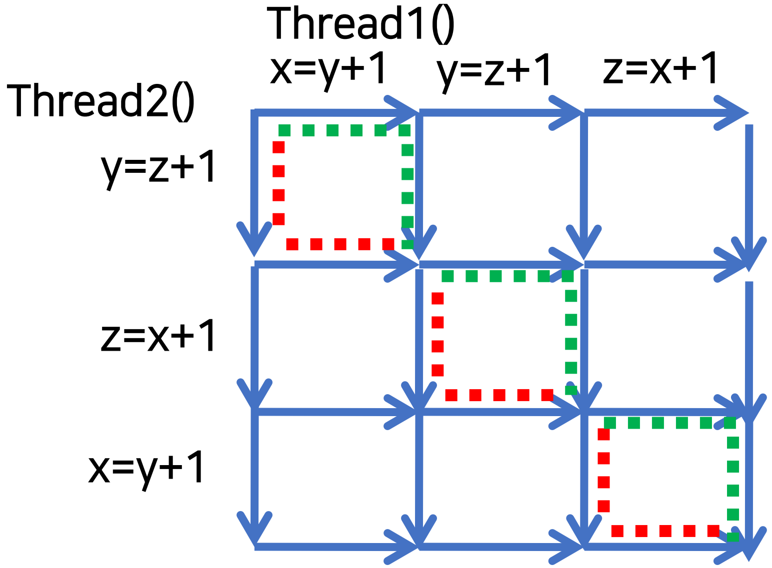



| | |
|---|---|
| Trail1: 2,1,2 | Trail7: 3,2,4 |
| Trail2: 2,1,3 | Trail8: 4,3,2 |
| Trail3: 2,2,3 | Trail9: 4,3,5 |
| Trail4: 2,3,3 | Trail10: 5,4,3 |
| Trail5: 2,4,3 | Trail11: 5,4,6 |
| Trail6: 3,2,3 | |

# Static SW Code vs. Dynamic SW Executions

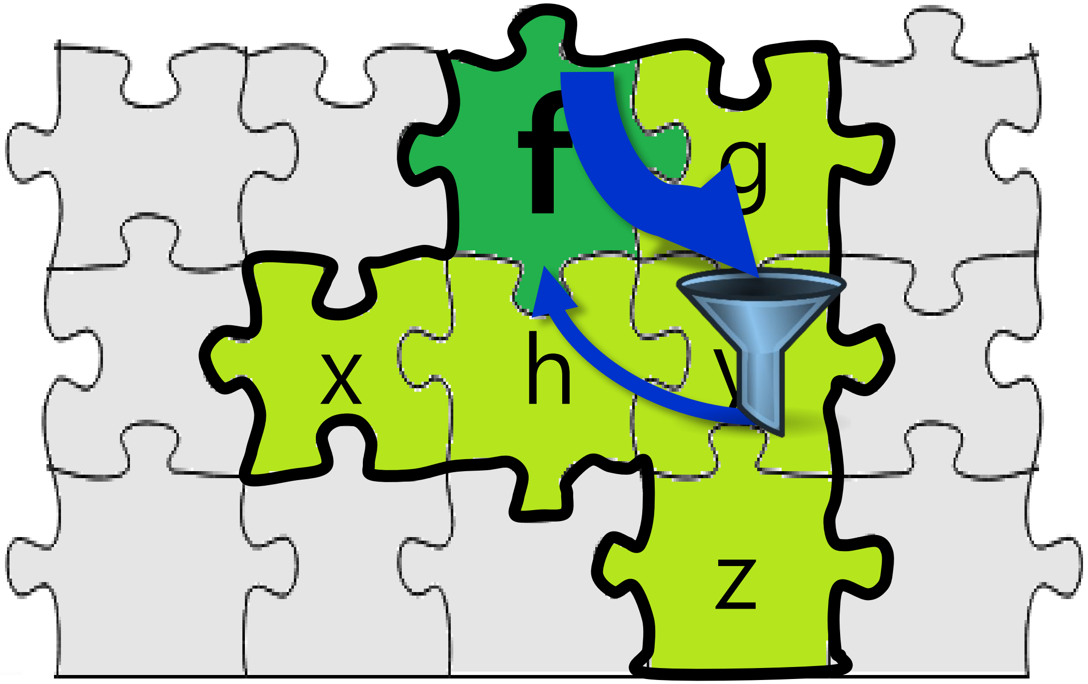



```
int x=0, y=0, z =0;
void Thread1()
{x=y+1; y=z+1; z= x+1;}
void Thread2()
{y=z+1; z=x+1; x=y+1;}
```



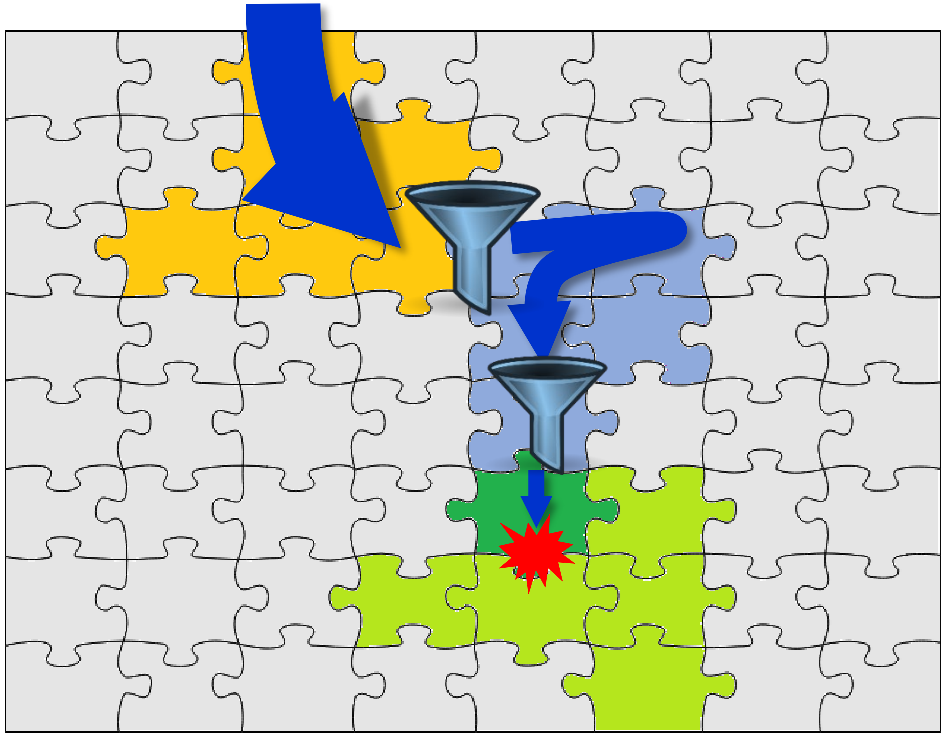
# New Applications of Function Relevance

- Defining Extended Units for Unit Testing



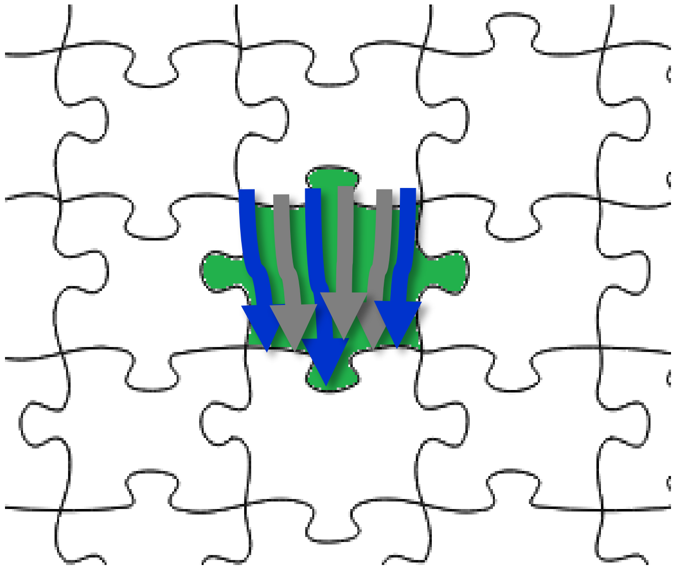

**Green**: Target function *f*
**Light green**: Functions highly relevant to the target
**Grey**: Functions not relevant to the target
**Green + light green: Extended Unit of *f***

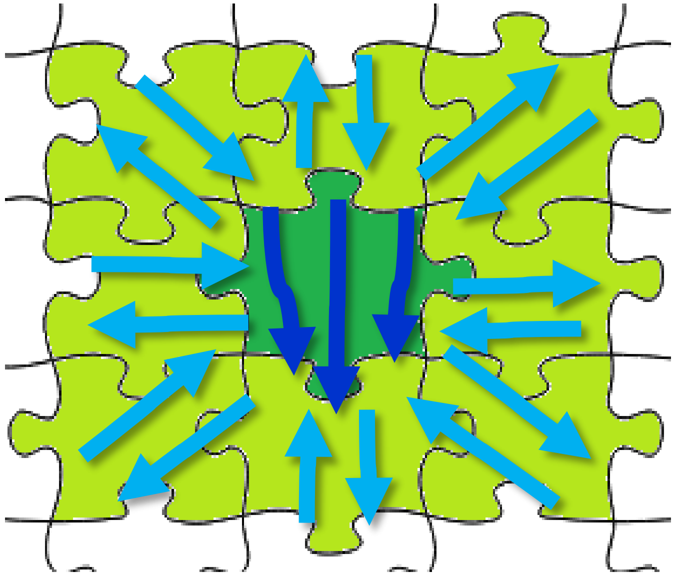
› False Alarm Filtering by Using Closely Relevant Contexts



**Yellow** & **sky blue**: a calling context of a target func. *f*

# Unit Testing *f* without or with Contexts of *f*





**Without Contexts of *f***

Pros: fast exploration of target unit execution paths

Cons: infeasible target unit executions

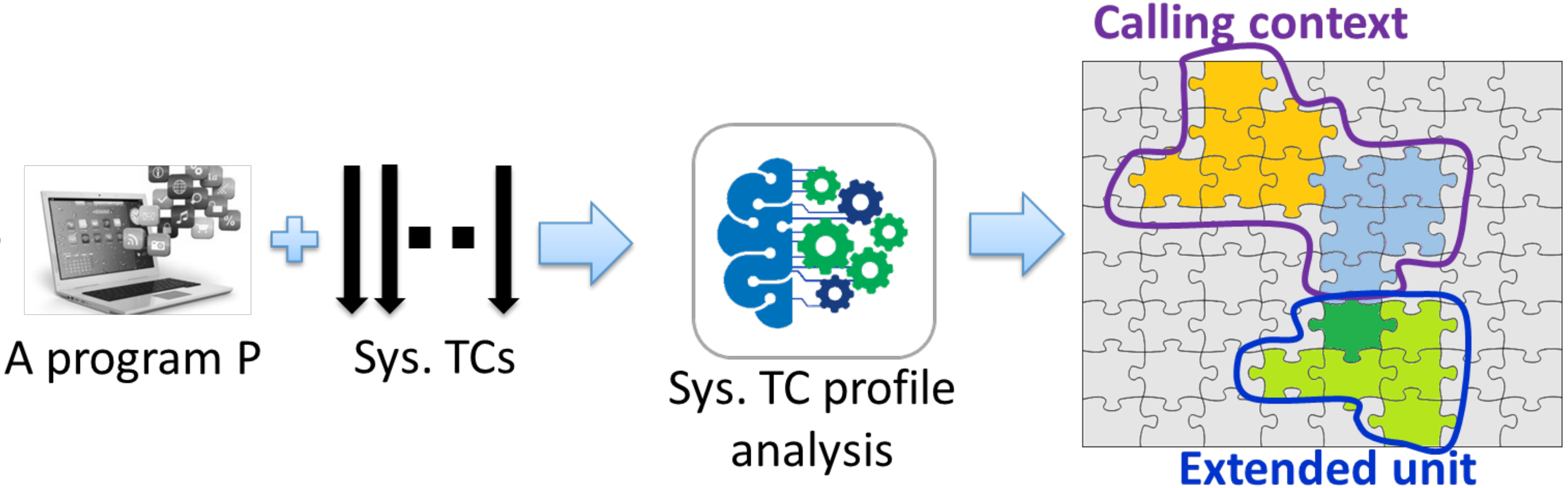
**With Contexts of *f***

Pros: reduced infeasible target unit executions

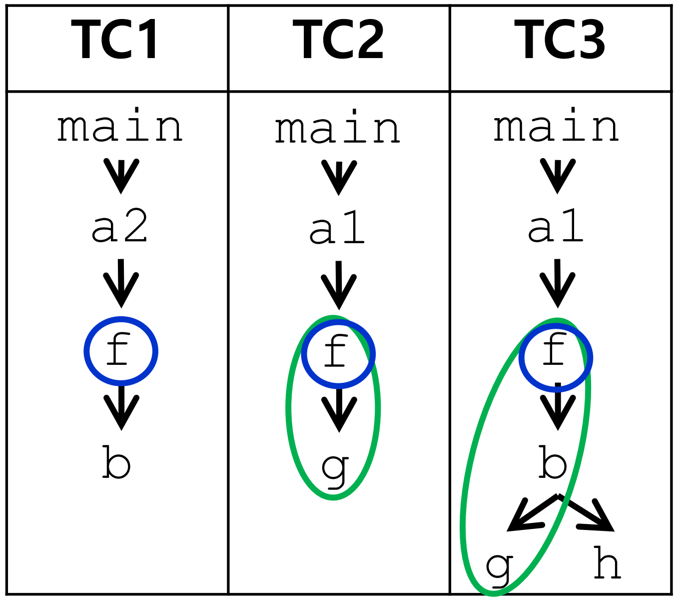
Cons: slow exploration of target unit execution due to large cost of exploring context functions

# Computing Function Relevance based-on System TCs

Phase1:
Defining **extended units**
and **calling contexts**




Function call profile



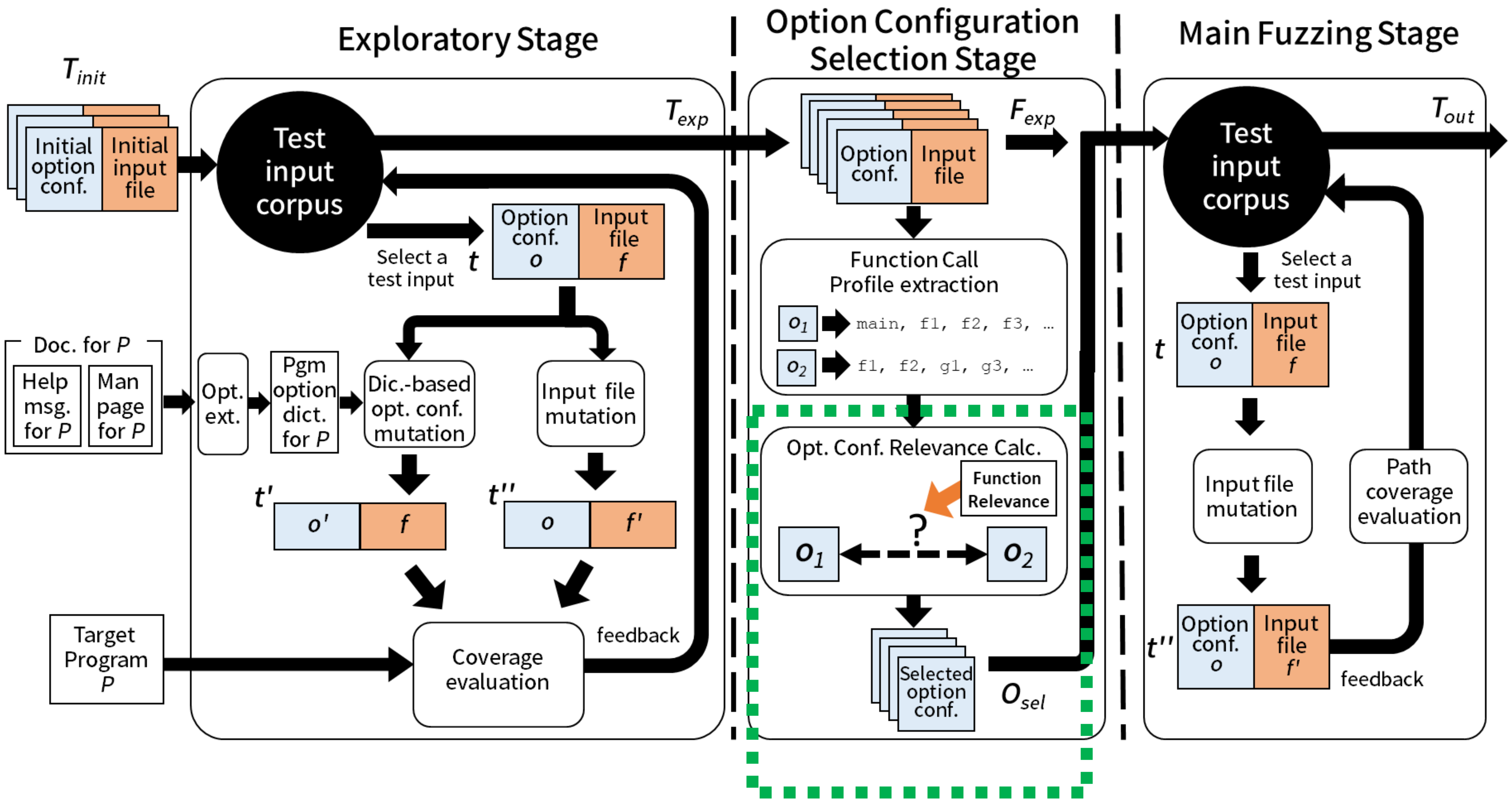

$$P(g|f) = \frac{|f \text{ calls } g|}{|f|} = \frac{|TC2, TC3|}{|TC1, TC2, TC3|} = 0.66$$

Relevance of f
on other functions
(Threshold τ =0.6)

P(b | f) = 0.66
P(g | f) = 0.66
~~P(h | f) = 0.33~~
...

# Recent New Application of Func. Relevance for Fuzzing

A. Lee, I. Ariq, Y. Kim, and M. Kim, **POWER: Program Option-Aware Fuzzer for High Bug Detection Ability**, ICST, 2022

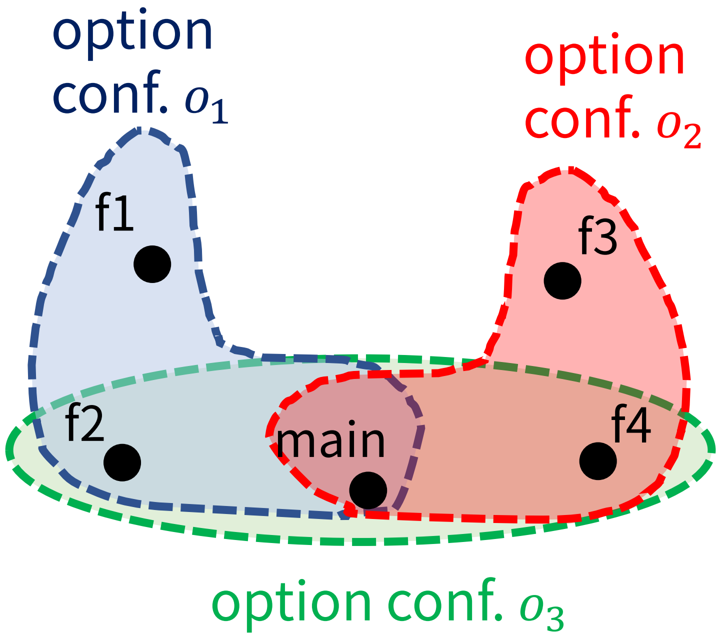


# How to Select Command-line Option Configurations

Problem:

- There exist too many different command-line option configurations for a target program.

- We need to select far different option configurations only




(a)

| Func. Relevance | | |
|---|---|---|
| f1 | f2 | High |
| f1 | f3 | Low |
| f1 | f4 | Low |
| f1 | main | High |
| f2 | f3 | Low |
| f2 | f4 | Low |
| f2 | main | High |
| f3 | main | Low |
| f3 | f4 | High |
| f4 | main | High |
| … | | |

(b)

# Improved Crash Bug Detection Ability by Selecting Option Configurations

- POWER found 2.15 (= 88/41) times more crashes than $POWER_{Rnd}$
  - For example, on tiffinfo, POWER found four times more crashes and covered 18.1% (=(3228.1-2732.4)/2732.4) more branches than $POWER_{Rnd}$.

THE TOTAL NUMBER OF CRASHES DETECTED AND THE AVERAGE NUMBERS OF BRANCHES COVERED BY THE VARIANTS OF POWER

| Targets | $POWER^{Rnd}$ | | $POWER^{KMO}$ | | POWER | |
|---|---|---|---|---|---|---|
| | #uniq. crash | #branch covered | #uniq. crash | #branch covered | #uniq. crash | #branch covered |
| avconv | 4 | 11709.6 | 7 | 17197.7 | 5 | 15006.2 |
| bison | 1 | 5728.0 | 3 | 6637.6 | 5 | 6138.0 |
| cflow | 3 | 1553.2 | 4 | 1689.1 | 2 | 1675.3 |
| cjpeg | 0 | 3920.1 | 0 | 4192.8 | 0 | 4086.7 |
| djpeg | 0 | 2598.1 | 0 | 2651.7 | 0 | 2513.7 |
| dwarfdump | 1 | 6565.5 | 4 | 7563.7 | 2 | 7240.6 |
| exiv2 | 0 | 8679.3 | 0 | 9636.8 | 1 | 9567.0 |
| ffmpeg | 1 | 36252.6 | 1 | 48122.8 | 2 | 45392.8 |
| gm | 0 | 6492.3 | 0 | 9454.0 | 1 | 9710.1 |
| gs | 0 | 22586.2 | 1 | 24905.8 | 0 | 24161.6 |
| jasper | 0 | 3674.4 | 0 | 3660.1 | 0 | 4101.0 |
| mpg123 | 0 | 3744.1 | 1 | 4006.3 | 1 | 3809.3 |
| mutool | 0 | 12423.8 | 0 | 15746.1 | 0 | 13647.7 |
| nasm | 4 | 6403.2 | 3 | 6578.8 | 4 | 6506.6 |
| objdump | 13 | 26237.9 | 8 | 24639.1 | 13 | 33070.5 |
| pdftohtml | 0 | 7184.0 | 0 | 8100.5 | 4 | 7600.7 |
| pdftopng | 0 | 7341.9 | 0 | 8947.8 | 9 | 8687.5 |
| pdftops | 0 | 8177.3 | 0 | 9719.0 | 9 | 9354.9 |
| pngfix | 0 | 1107.8 | 0 | 1191.2 | 0 | 1143.1 |
| pspp | 9 | 3389.2 | 7 | 4462.3 | 8 | 5650.0 |
| readelf | 0 | 9402.0 | 1 | 8799.3 | 8 | 10321.6 |
| size | 1 | 5078.7 | 4 | 7621.5 | 3 | 9054.8 |
| tiff2pdf | 0 | 4126.1 | 0 | 4226.8 | 0 | 4177.1 |
| tiff2ps | 1 | 2950.8 | 1 | 3274.1 | 0 | 3379.0 |
| tiffinfo | 1 | 2732.4 | 1 | 3060.9 | 4 | 3228.1 |
| vim | 0 | 39844.8 | 2 | 45466.5 | 5 | 45654.3 |
| xmlcatalog | 0 | 6598.8 | 0 | 6413.9 | 0 | 7598.9 |
| xmllint | 2 | 14245.7 | 2 | 14406.5 | 2 | 14420.5 |
| xmlwf | 0 | 3590.3 | 0 | 3733.0 | 0 | 3733.8 |
| yara | 0 | 3455.6 | 0 | 3954.2 | 0 | 3118.9 |
| Total | 41 | | 50 | | 88 | |

# On-going Work to Improve Dynamic Func. Relevance Metric (co-work w/ 김윤호 교수님)

- Before:
  - 타겟 함수 f와 f와의 관련도를 측정할 다른 함수 g에 대해서, 얼마나 많은 테스트 입력값이 두 함수 f,g를 같이 실행하는가를 측정
- After:
  - 각 테스트에서 생성된 호출 시퀀스 **(Call sequence)**를 f를 기준으로 여러개의 조각 (segments) 으로 분할 하여, 얼마나 많은 f,g가 같은 조각에 포함되는지를 측정.
  - 단순히 실행되었는가에서 더 나아가 호출 시퀀스를 비교하기 때문에, 더 정교하게 두 함수 사이의 관련도 측정이 가능

# 실험결과

| | | #bugs | | #false alarms | | F/T ratio | |
|---|---|---|---|---|---|---|---|
| | #target bugs | CONBRIO | Seg. Metric | CONBRIO | Seg. Metric | CONBRIO | Seg. Metric |
| Bash | 6 | 5 | 4 | 18 | 17 | 3.6 | 4.3 |
| Flex | 2 | 1 | 1 | 6 | 5 | 6.0 | 5.0 |
| Grep | 5 | 4 | 4 | 13 | 14 | 3.3 | 3.5 |
| Gzip | 2 | 2 | 2 | 5 | 4 | 2.5 | 2.0 |
| Make | 3 | 3 | 3 | 9 | 10 | 3.0 | 3.3 |
| Sed | 2 | 2 | 2 | 5 | 7 | 2.5 | 3.5 |
| Vim | 6 | 5 | 5 | 25 | 25 | 5.0 | 5.0 |
| Perl | 6 | 6 | 6 | 57 | 24 | 9.5 | 4.0 |
| Bzip2 | 2 | 2 | 2 | 10 | 8 | 5.0 | 4.0 |
| Gcc | 15 | 14 | 13 | 79 | 78 | 5.6 | 6.0 |
| Gobmk | 5 | 5 | 5 | 39 | 34 | 7.8 | 6.8 |
| Hmmer | 3 | 3 | 3 | 12 | 12 | 4.0 | 4.0 |
| Sjeng | 2 | 2 | 2 | 8 | 8 | 4.0 | 4.0 |
| libquantum | 3 | 3 | 3 | 5 | 3 | 1.7 | 1.0 |
| h264ref | 5 | 4 | 5 | 17 | 16 | 4.3 | 3.2 |
| **Sum** | **67** | **61** | **60** | | | | |
| **Avg** | | | | **20.5** | **17.7** | **4.5** | **4.0** |

# Questions? Comments?