Towards Efficient Data-flow Test Data Generation Using KLEE

Chengyu Zhang[1], Ting Su[2], Yichen Yan[1], Ke Wu[3], Geguang Pu[1]

[1]East China Normal University, China
[2]Nanyang Technological University, Singapore
[3]National Trusted Embedded Software Engineering Technology Research Center, China

2018.4.19

Outline

Introduction

Approach

Implementation

Evaluation

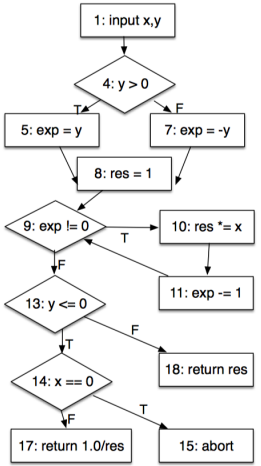
Application

Data-flow Testing

Data-flow testing identifies potential bugs by checking the correctness of variable definitions by observing their corresponding uses.
Several empirical studies have demonstrated that data-flow testing is more effective in fault detection than control-flow testing.

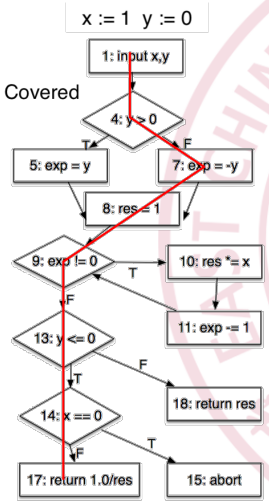
An example

```
1   double power(int x,int y){
2       int exp; double res;
3       if (y>0) exp = y;
4       else exp = -y;
5       res=1;
6       while (exp!=0){
7         res *= x;
8         exp -= 1;
9       }
10      if (y<=0)
11        if(x==0)
12          abort;
13        else
14          return 1.0/res;
15      return res;
16  }
```



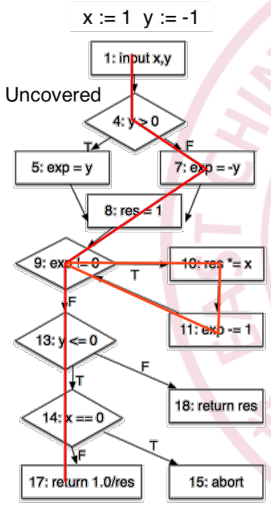
An example

```
1   double power(int x,int y){
2       int exp; double res;
3       if (y>0) exp = y;
4       else exp = -y;
5       res=1; //definition
6       while (exp!=0){
7         res *= x;
8         exp -= 1;
9       }
10      if (y<=0)
11        if(x==0)
12          abort;
13        else
14        return 1.0/res; //use
15      return res;
16  }
```



An example

```
1   double power(int x,int y){
2       int exp; double res;
3       if (y>0) exp = y;
4       else exp = -y;
5       res=1; //definition
6       while (exp!=0){
7         res *= x; //redefinition
8         exp -= 1;
9       }
10      if (y<=0)
11        if(x==0)
12          abort;
13        else
14        return 1.0/res; //use
15      return res;
16  }
```



Approach

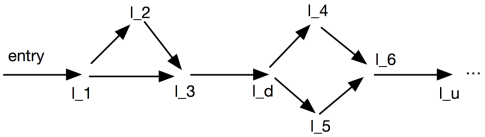
Three heuristic search strategies:

- Cut point guided search
- Backtrack strategy
- Redefinition Path Pruning

Cut point guided search

Definition (Cut Point)

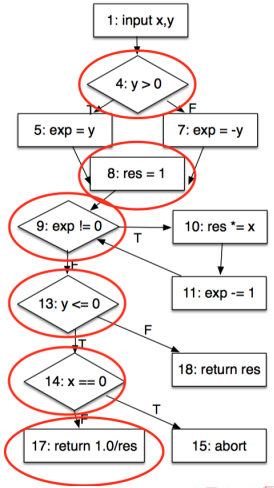
For $du(l_d, l_u, v)$
cut points are a sequence of critical control points $c_1, \ldots, c_i, \ldots, c_n$ that must be passed through in succession by any control flow paths that cover this pair.
$c_1 \gg^I \ldots c_i \gg^I l_d \gg^I \ldots c_n \gg^I l_u$.



Cut points: $\{l_1, l_3, l_d, l_6, l_u\}$

Cut point guided search

```
1   double power(int x,int y){
2       int exp; double res;
3       if (y>0) exp = y;
4       else exp = -y;
5       res=1;
6       while (exp!=0){
7         res *= x;
8         exp -= 1;
9       }
10      if (y<=0)
11        if(x==0)
12          abort;
13        else
14          return 1.0/res;
15      return res;
16  }
```



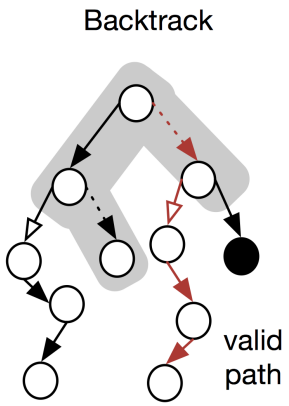
Backtrack strategy

Definition (Backtrack formula)

$$state\_weight(es) = \frac{1}{d^2} + \frac{1}{i^2} \qquad (1)$$

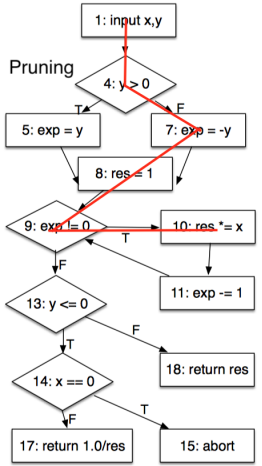
d: instruction distance toward the next uncovered cut point

i: number of instructions since the last new instruction have been covered

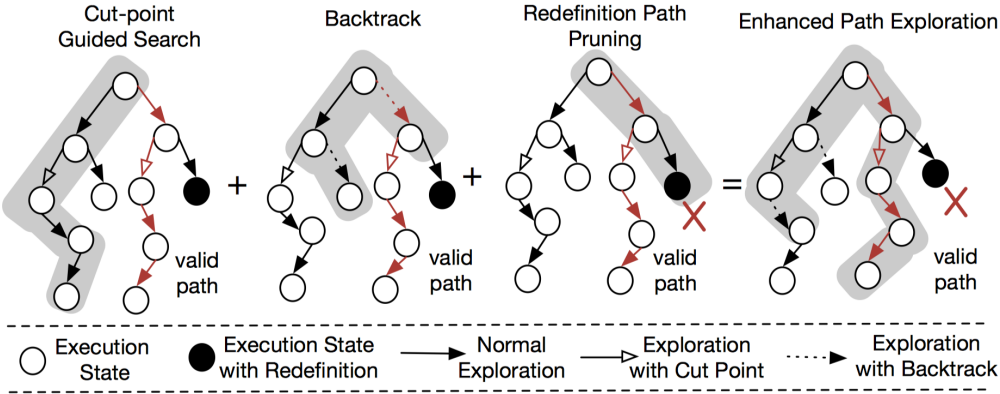


Redefinition Path Pruning

```
1   double power(int x,int y){
2       int exp; double res;
3       if (y>0) exp = y;
4       else exp = -y;
5       res=1; //definition
6       while (exp!=0){
7         res *= x; //redefinition
8         exp -= 1;
9       }
10      if (y<=0)
11        if(x==0)
12          abort;
13        else
14          return 1.0/res; //use
15      return res;
16  }
```



Approach Overview



Why We Choose KLEE?

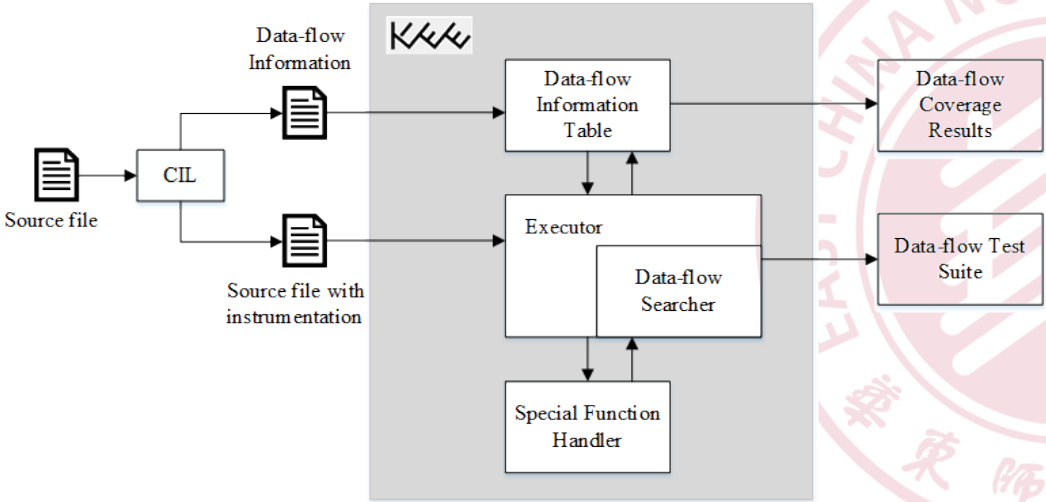
CAUT: https://www.lab301.cn/caut/

-  **Automated coverage-driven test data generation using dynamic symbolic execution**
  T. Su, G. Pu, B. Fang, J. He, J. Yan, S. Jiang, and J. Zhao (2014).
  SERE 2014
-  **Combining symbolic execution and model checking for data flow testing.**
  T. Su, Z. Fu, G. Pu, J. He, and Z. Su. (2015).
  ICSE 2015

Why We Choose KLEE?

- Open-source
- Robust
- Scalable

Data-flow Testing Framework



Experiment Setup

Subjects	#Sub	#LOC	#DU pair
Previous Literature	7	449	346
SIR	7	2,687	1,409
SV-COMP (ntdriver)	6	7,266	2,691
SV-COMP (ssh)	10	5,249	18,347

We evaluated the approach on 30 subjects.

Performance Statistics of Different Search Strategies

Name	Description
DFS	Depth First Search
RSS	Random State Search
RSS-COS:md2u	RSS interleaved with Min-Dist-to-Uncovered heuristic
SDGS	Shortest Distance Guided Search
CPGS	Cut Point Guided Search

Subject	DFS		RSS		RSS-MD2U		SDGS		CPGS	
	N	M (SIQR)	N	M (SIQR)	N	M (SIQR)	N	M (SIQR)	N	M (SIQR)
Total	6218	9.13	7883	13.54	8091	21.89	8035	13.51	8300	12.96

CPGS achieves the best performance in the symbolic execution approach.

Model Checking based Approach

Towards Efficient Data-flow Test Data Generation
T. Su, C. Zhang, Y. Yan, L. Fan, G. Pu, Y. Liu, Z. Fu, Z. Su
https://arxiv.org/abs/1803.10431

Subject	BLAST					CPAchecker					CBMC				
	F	I	U	M_F	M_I	F	I	U	M_F	M_I	F	I	U	M_F	M_I
Total	6720	10199	5874	1.82	5.25	8156	12748	1889	8.64	5.27	8984	13731	78	91.35	110.60

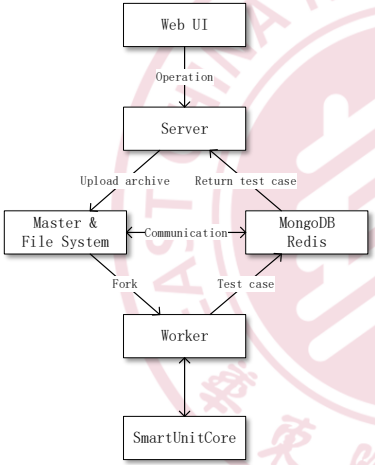
Model Checking VS. Symbolic Execution on Data-flow Testing

Subjects	#Sub	#LOC	#DU pair	Average Coverage		Median Time (s/pair)	
				KLEE	CPA	KLEE	CPA
Previous Literature	7	449	346	60%	72%	0.1	4.3
SIR	7	2,687	1,409	57%	60%	0.7	10.1
SV-COMP (ntdriver)	6	7,266	2,691	75%	51%	1.5	5
SV-COMP (ssh)	10	5,249	18,347	29%	31%	18	5.7

KLEE can easily achieve nearly 60% of data-flow coverage within less than 1 second for each pair in the subjects from previous literature and SIR.

Future Work

We planned to implement the data-flow testing approach in our cloud-based unit test framework.



SmartUnit (ICSE-SEIP'18)

- Automatically generate testing report.
- Automatically generate test case.
- Automatically insert function stub.
- Cloud-based platform for corporations.

It's adapted for LDRA Testbed®[1], Tessy®[2] and other popular automated testing tools for embedded systems. It supports three common control-flow coverage criteria: statement coverage, branch coverage and MC/DC (Modified Condition Decision Coverage).

[1] http://ldra.com/industrial-energy/products/ldra-testbed-tbvision/

[2] https://www.razorcat.com/en/product-tessy.html

SmartUnit (ICSE-SEIP'18)

Statement Coverage (#Functions)			Branch Coverage (#Functions)			MC/DC Coverage (#Functions)		
0%-50%	50%-99%	100%	0%-50%	50%-99%	100%	0%-50%	50%-99%	100%
9.3%	17.5%	73.2%	14.3%	12.5%	73.2%	47.8%	13.6%	38.6%

SmartUnit achieves the 100% statement and branch coverage on most of the functions.

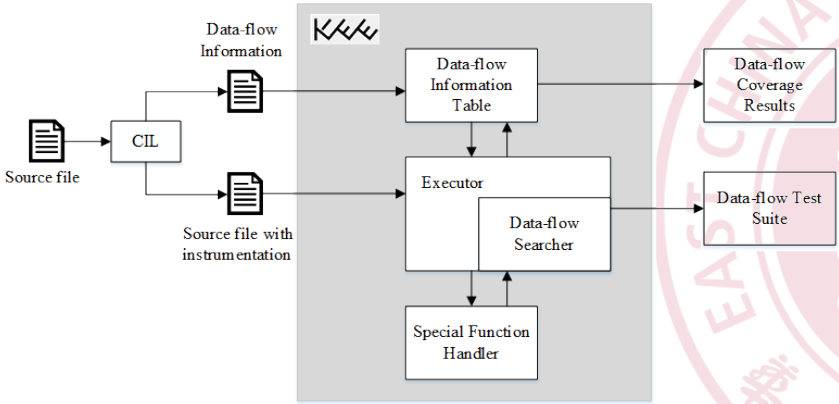
SmartUnit (ICSE-SEIP'18)

We have the cooperation with:

- *China Academy of Space Technology*
- *CASCO Signal Ltd.*
- *The 32nd Institute of China Electronics Technology Group Corporation*
- ...

SmartUnit: Empirical Evaluations for Automated Unit Testing of Embedded Software in Industry
C. Zhang, Y. Yan, H. Zhou, Y. Yao, K. Wu, T. Su, W. Miao, G. Pu(2018).
ICSE-SEIP, May 27-June 3, 2018

Conclusion



https://github.com/muchang/klee https://tingsu.github.io/files/hybrid_dft.html

Towards Efficient Data-flow Test Data Generation Using KLEE

Chengyu Zhang[1], Ting Su[2], Yichen Yan[1], Ke Wu[3], Geguang Pu[1]

[1]East China Normal University, China
[2]Nanyang Technological University, Singapore
[3]National Trusted Embedded Software Engineering Technology Research Center, China

2018.4.19

Definition (Program Paths)

Two kinds of program paths, i.e., control flow paths and execution paths are distinguished during data-flow testing. Control flow paths are the paths from the control flow graph of the program under test, which abstract the flow of control. Execution paths are driven by concrete program inputs, which represent dynamic program executions. Both of them can be represented as a sequence of control points (denoted by line numbers), e.g., $l_1, \ldots, l_i, \ldots, l_n$.

Definition (Def-use Pair)

The test objective of data-flow testing is referred as a def-use pair, denoted by $du(l_d, l_u, v)$. Such a pair appears when there exists a control flow path that starts from the variable definition statement l_d (or the def statement in short), and then reaches the variable use statement l_u (or the use statement in short), but no statements on the subpaths from l_d to l_u redefine the variable v.

Definition (Data-flow Testing)

Data-flow testing requires to generate at least one test case t for each def-use pair (l_d, l_u, v) in the program P under test. The test input t should drive an execution path p that covers the variable definition statement at l_d, and then covers variable use statement at l_u, but without covering any redefinition statements w.r.t v, i.e., the subpath from l_d to l_u is a def-clear path. Such a test adequacy requirement is called all def-use coverage[a] in data-flow testing.

[a] In this paper, we follow the all def-use coverage defined by Rapps and Weyuker, since almost all of the literature that followed uses or extends this definition.

Definition (Cut Point)

Given a def-use pair $du(l_d, l_u, v)$, its cut points are a sequence of critical control points $c_1, \ldots, c_i, \ldots, c_n$ that must be passed through in succession by any control flow paths that cover this pair. The latter control point is the immediate dominator of the former one, *i.e.*, $c_1 \gg^I \ldots c_i \gg^I l_d \gg^I \ldots c_n \gg^I l_u$. Each control point in this sequence is called a cut point.