KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs

20203331 신명근 MyeongGeun Shin

2020.10.14 Wed @ IS893-2020-fall

Testing is Hard!

Cost of Software Bugs

~ $100

when found in early stage

~ $1,500

when found in QA testing

~ $10,000

when found in production

Two ways of testing

symbOLIC	CONCrete
Explore any feasible path Not scalable Environment problem Rely on SMT solver	No false alarms No manual work Hard to hit narrow test

Dynamic Symbolic Execution

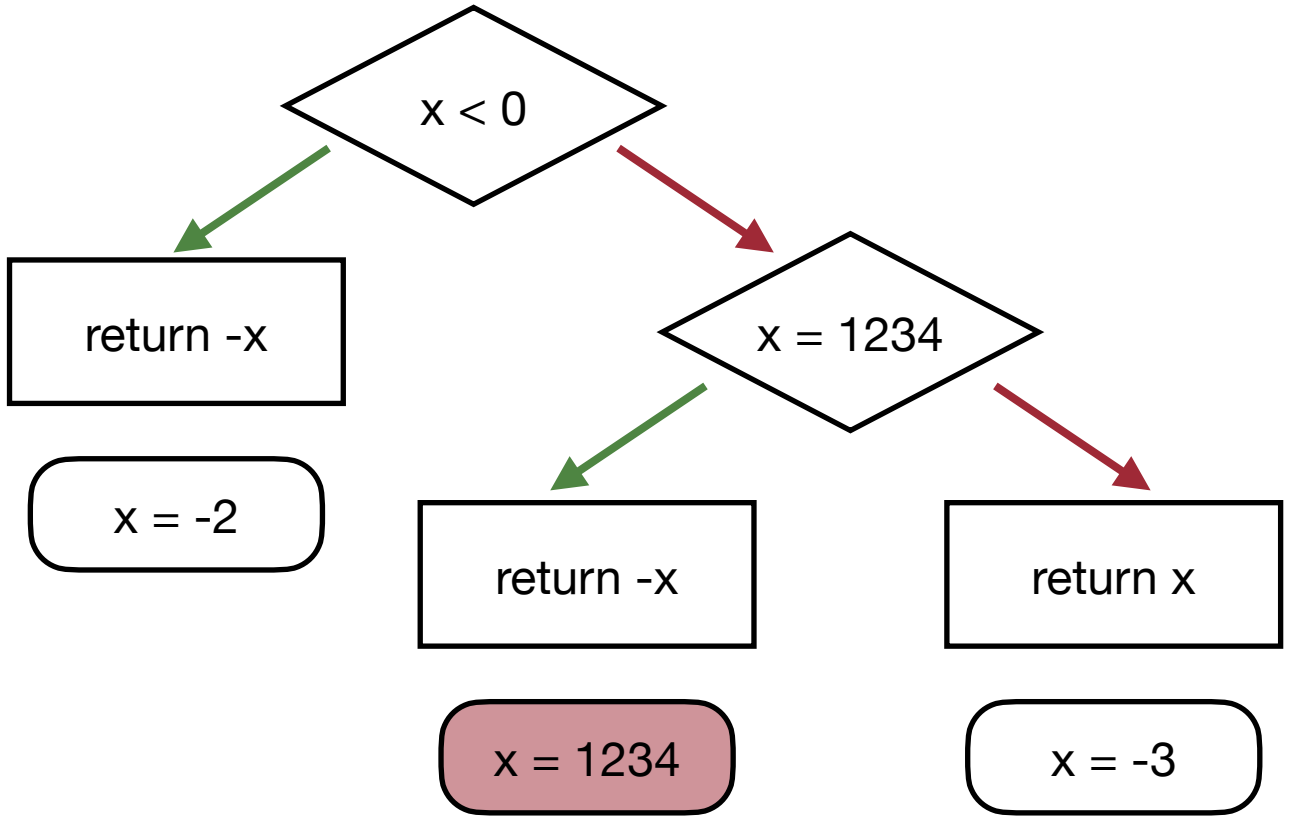
```
1   int bad_abs(int x)
2   {
3       if(x < 0)
4           return -x;
5       if(x == 1234)
6           return -x; // ERROR
7       return x;
8   }
9
```

Fuzzer's case

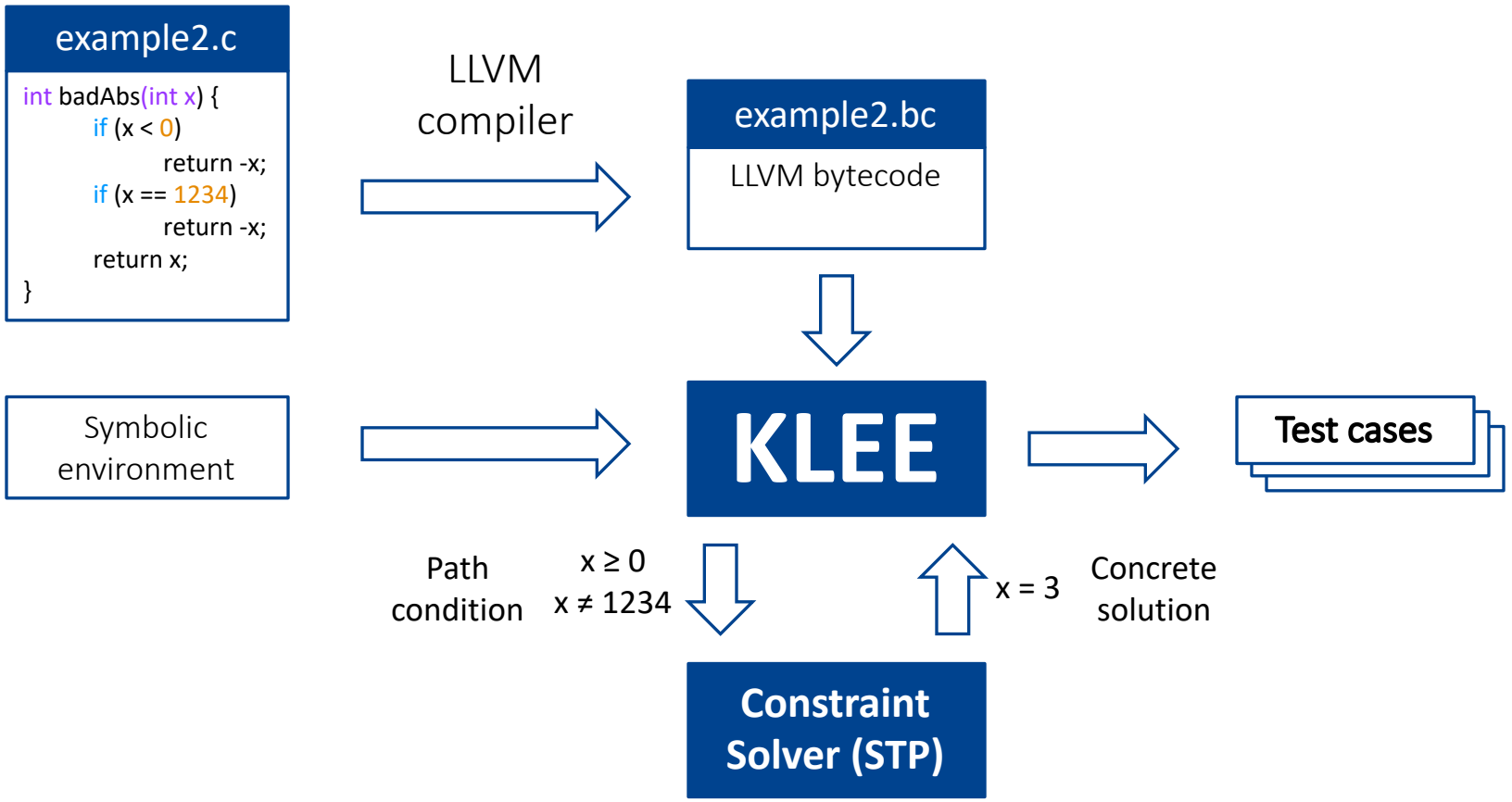
- Very hard to reach Error
- Probability of 1 / 2^32

Dynamic Symbolic Execution

```
1   int bad_abs(int x)
2   {
3       if(x < 0)
4           return -x;
5       if(x == 1234)
6           return -x; // ERROR
7       return x;
8   }
9
```



KLEE architecture



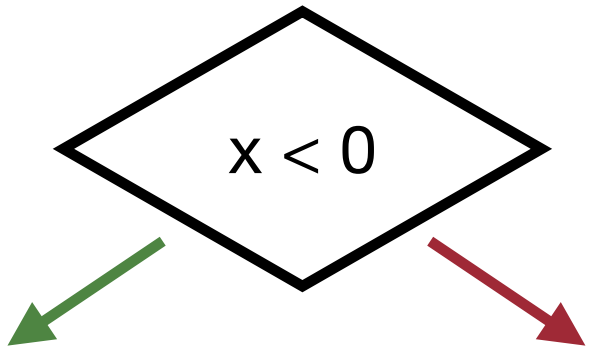
KLEE : symbolic state

- Unlike any other process, KLEE environment needs to keep track of path constraints

- The core of KLEE is an interpreter loop that selects a state and executes single instruction

Symbolic execution

Conditional



Dangerous

X / 0

Overflows

OOB ref

Loop unrolling

```
1   while(cond){
2       stmt;
3   }
4
5   if(cond){
6       stmt;
7       if(cond){
8           stmt;
9           if(cond){
10              ...
11          }
12      }
13  }
```

Maybe infinite!
➡️ have timeout

Three main difficulties

- Number of paths grow exponentially
- Constraint solving is not cheap
- Interaction between environments

Resolve exponential paths

- Random exploration
- Coverage-optimized exploration

- Compact representations
  - Immutable heap
  - Shared states

Constraint solving queries

- Expression rewritings
- Power to bitshift

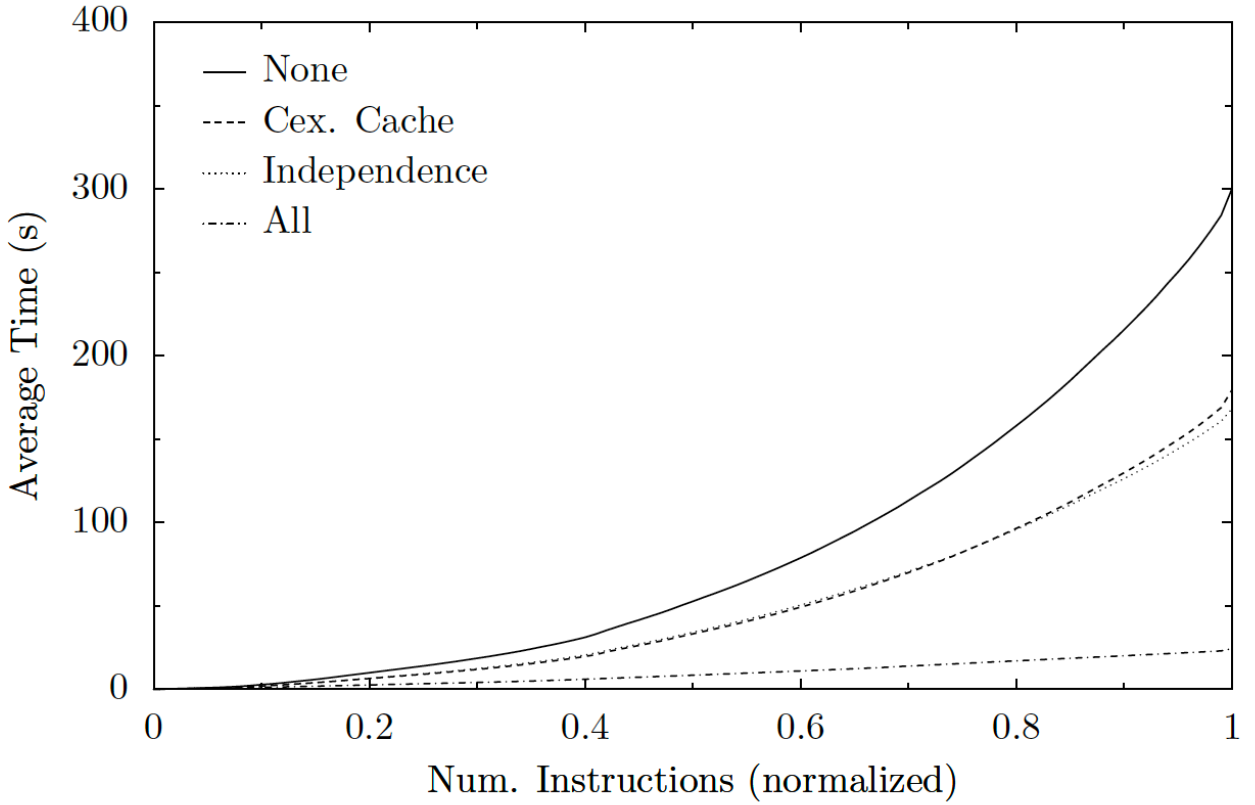
- Linear simplification

Query optimizations

- C is impossible, $C \subseteq C'$, then neither does C'
  - $x = 1 \,\&\, x = 2$ ❌➡️ $x = 1 \,\&\, x = 2 \,\&\, x = 3$ ❌
- C is possible, $C \supseteq C'$, then C' has solution s
  - $x = 1 \,\&\, x < 2$ ✅➡️ $x = 1$ ✅
- C is possible, $C \subseteq C'$, then C' likely has solution s
  - $x = 1 \,\&\, x < 2$ ✅➡️ $x = 1 \,\&\, x < 2 \,\&\, x < 3$ ✨

Query optimization works good

Optimizations	Queries	Time (s)	STP Time (s)
None	13717	300	281
Independence	13717	166	148
Cex. Cache	8174	177	156
All	699	20	10



Environment modeling

- Input/output outside of program
  - argv
  - env
  - files
  - packets ❌

Implemented about 40 syscalls

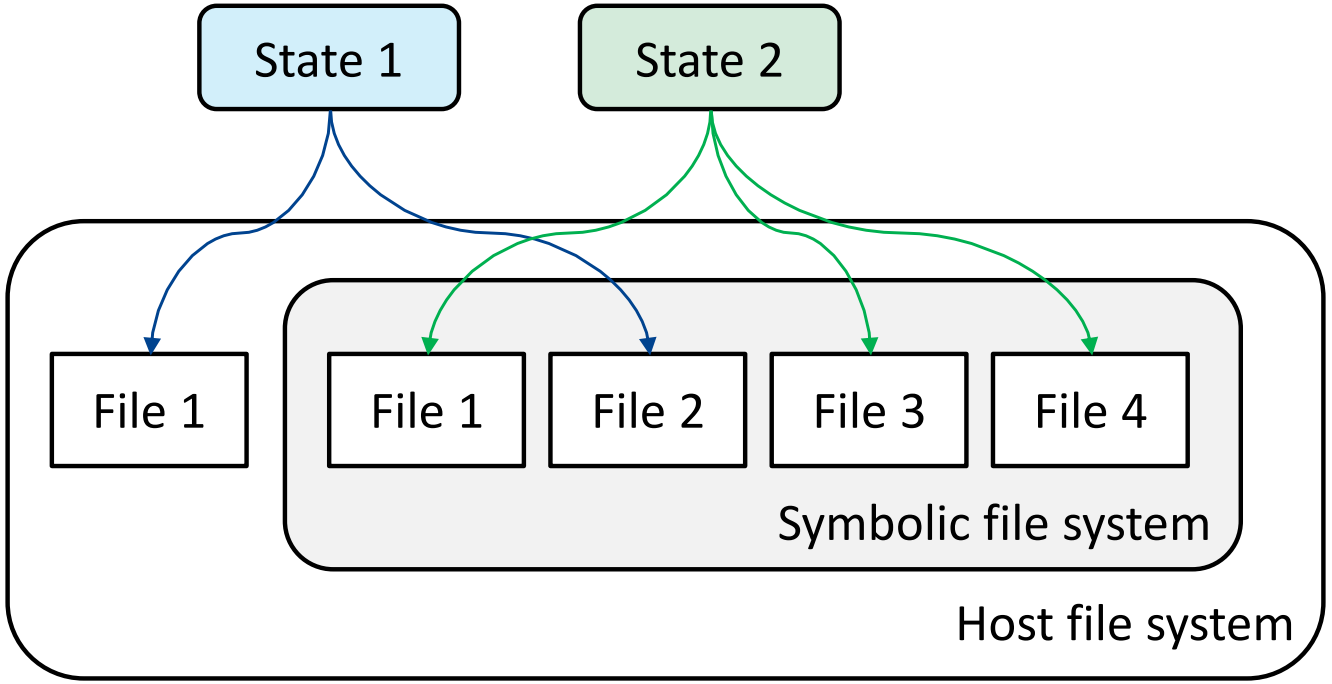
Even Modeling

- File system model:
- Actual concrete file

- Symbolic file

Symbolic Model environment

- Can coexist with real system files

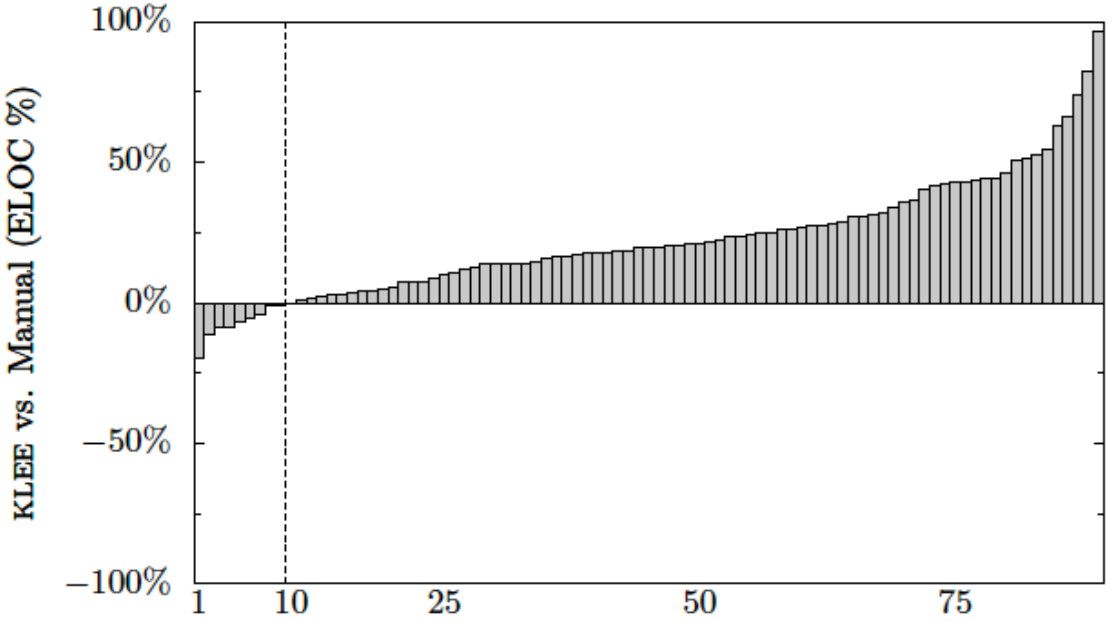


Real example

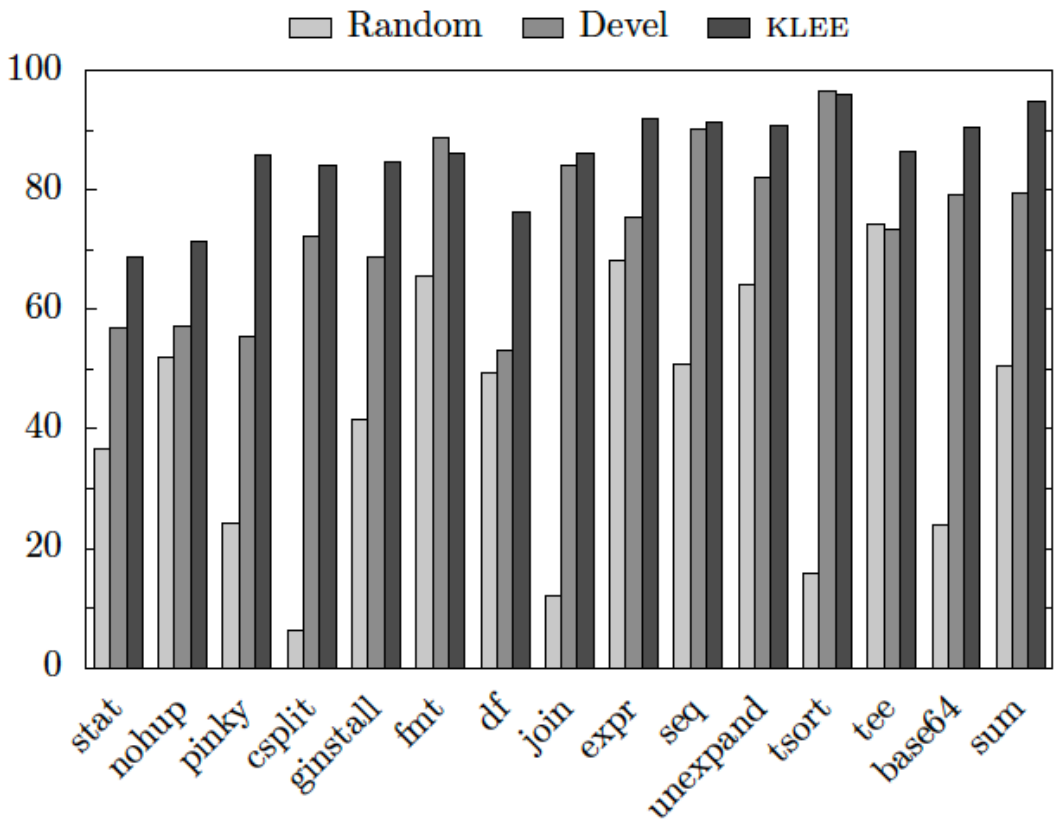
```
1 : ssize_t read(int fd, void *buf, size_t count) {
2 :   if (is_invalid(fd)) {
3 :     errno = EBADF;
4 :     return -1;
5 :   }
6 :   struct klee_fd *f = &fds[fd];
7 :   if (is_concrete_file(f)) {
8 :     int r = pread(f->real_fd, buf, count, f->off);
9 :     if (r != -1)
10:       f->off += r;
11:     return r;
12:   } else {
13:     /* sym files are fixed size: don't read beyond the end. */
14:     if (f->off >= f->size)
15:       return 0;
16:     count = min(count, f->size - f->off);
17:     memcpy(buf, f->file_data + f->off, count);
18:     f->off += count;
19:     return count;
20:   }
21: }
```

Evaluations

Coverage (w/o lib)	COREUTILS KLEE tests	COREUTILS Devel. tests	BUSYBOX KLEE tests	BUSYBOX Devel. tests
100%	16	1	31	4
90-100%	40	6	24	3
80-90%	21	20	10	15
70-80%	7	23	5	6
60-70%	5	15	2	7
50-60%	-	10	-	4
40-50%	-	6	-	-
30-40%	-	3	-	2
20-30%	-	1	-	1
10-20%	-	3	-	-
0-10%	-	1	-	



Evaluations



- Significantly beat handwritten tests
- Found 3 bugs in Coreutils that had been missed for 15 years.
- 56 bugs in 452 apps
- Even in HiStar OS

Questions?