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

Presented by Jordan Gillard

A little bit of background

A little bit of background

- Capable of automatically generating high-coverage tests.

A little bit of background

- Capable of automatically generating high-coverage tests.
- The authors use KLEE to generate tests for the GNU Core Utilities suite.

A little bit of background

- Capable of automatically generating high-coverage tests.
- The authors use KLEE to generate tests for the GNU Core Utilities suite.
- KLEE beats the line coverage of the developers own hand-written tests.

A little bit of background

- Capable of automatically generating high-coverage tests.
- The authors use KLEE to generate tests for the GNU Core Utilities suite.
- KLEE beats the line coverage of the developers own hand-written tests.
- KLEE is an effective bug-finding tool.

What is the problem?

What is the problem?

- Many types of errors are difficult to test.

What is the problem?

- Many types of errors are difficult to test.
- Researchers doubt that automatic test generation tools like KLEE can consistently work with real applications.

What is the problem?

- Many types of errors are difficult to test.
- Researchers doubt that automatic test generation tools like KLEE can consistently work with real applications.
- Concerns:
  - Number of paths through code grows exponentially

What is the problem?

- Many types of errors are difficult to test.
- Researchers doubt that automatic test generation tools like KLEE can consistently work with real applications.
- Concerns:
  - Number of paths through code grows exponentially
  - Dealing with code that interacts with the surrounding environment

What technique is proposed?

What technique is proposed?

- Researchers present KLEE - a new symbolic execution tool.

What technique is proposed?

- Researchers present KLEE - a new symbolic execution tool.
- Researchers show that KLEE performs well on real environmentally-intensive programs.

Intro continued

Intro continued

- KLEE gets high coverage on a broad set of problems.

Intro continued

- KLEE gets high coverage on a broad set of problems.
- KLEE gets significantly better line-coverage than years of developer-made tests.

Intro continued

- KLEE gets high coverage on a broad set of problems.
- KLEE gets significantly better line-coverage than years of developer-made tests.
- KLEE works out of the box.

Intro continued

- KLEE gets high coverage on a broad set of problems.
- KLEE gets significantly better line-coverage than years of developer-made tests.
- KLEE works out of the box.
- KLEE finds serious bugs.

Intro continued

- KLEE gets high coverage on a broad set of problems.
- KLEE gets significantly better line-coverage than years of developer-made tests.
- KLEE works out of the box.
- KLEE finds serious bugs.
- KLEE runs on the raw version of the code.

Intro continued

- KLEE gets high coverage on a broad set of problems.
- KLEE gets significantly better line-coverage than years of developer-made tests.
- KLEE works out of the box.
- KLEE finds serious bugs.
- KLEE runs on the raw version of the code.
- KLEE is not limited to low-level errors.

Intro continued

- KLEE gets high coverage on a broad set of problems.
- KLEE gets significantly better line-coverage than years of developer-made tests.
- KLEE works out of the box.
- KLEE finds serious bugs.
- KLEE runs on the raw version of the code.
- KLEE is not limited to low-level errors.
- KLEE works with non-application code.

Overview

Overview

- Complexity

```
1 : void expand(char *arg, unsigned char *buffer) {          8
2 :   int i, ac;                                             9
3 :   while (*arg) {                                        10*
4 :     if (*arg == '\\') {                                11*
5 :       arg++;
6 :       i = ac = 0;
7 :       if (*arg >= '0' && *arg <= '7') {
8 :         do {
9 :           ac = (ac << 3) + *arg++ - '0';
10:          i++;
11:        } while (i<4 && *arg>='0' && *arg<='7');
12:        *buffer++ = ac;
13:      } else if (*arg != '\0')
14:        *buffer++ = *arg++;
15:    } else if (*arg == '[') {                            12*
16:      arg++;                                             13
17:      i = *arg++;                                        14
18:      if (*arg++ != '-') {                               15!
19:        *buffer++ = '[';
20:        arg --= 2;
21:        continue;
22:      }
23:      ac = *arg++;
24:      while (i <= ac) *buffer++ = i++;
25:      arg++;       /* Skip ']' */
26:    } else
27:      *buffer++ = *arg++;
28:  }
29: }
30: ...
31: int main(int argc, char* argv[]) {                        1
32:   int index = 1;                                         2
33:   if (argc > 1 && argv[index][0] == '-') {               3*
34:     ...                                                  4
35:   }                                                      5
36:   ...                                                    6
37:   expand(argv[index++], index);                          7
38:   ...
39: }
```

Overview

- Complexity
- Environmental Dependencies

Overview Con’t

Overview Con’t

- Hit every line of executable code in the program.

Overview Con’t

- Hit every line of executable code in the program.
- Detect at each line potential dangerous operations.

KLEE Architecture

KLEE Architecture

- Core of KLEE is an interpreter loop

KLEE Architecture

- Core of KLEE is an interpreter loop
- KLEE branches out to handle different conditions, and clones its state.

KLEE Architecture

- Core of KLEE is an interpreter loop
- KLEE branches out to handle different conditions, and clones its state.
- KLEE has a unique way of handling state.

KLEE Architecture

- Core of KLEE is an interpreter loop
- KLEE branches out to handle different conditions, and clones its state.
- KLEE has a unique way of handling state.
- KLEE optimizes queries.

Effectiveness of Optimizations

Effectiveness of Optimizations

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

Table 1: Performance comparison of KLEE's solver optimizations on COREUTILS. Each tool is run for 5 minutes without optimization, and rerun on the same workload with the given optimizations. The results are averaged across all applications.

Environment Modeling

Environment Modeling

- Authors created simple models for 40 system calls.

Environment Modeling

- Authors created simple models for 40 system calls.
- Example 1: Modeling the file system

Environment Modeling

- Authors created simple models for 40 system calls.
- Example 1: Modeling the file system
- Failing system calls

Evaluation

Evaluation

- Line coverage as measure of KLEE test case effectiveness.

Evaluation

- Line coverage as measure of KLEE test case effectiveness.
- KLEE minimizes the number of generated test cases.

KLEE Performance - GNU Coreutils

KLEE Performance - GNU Coreutils

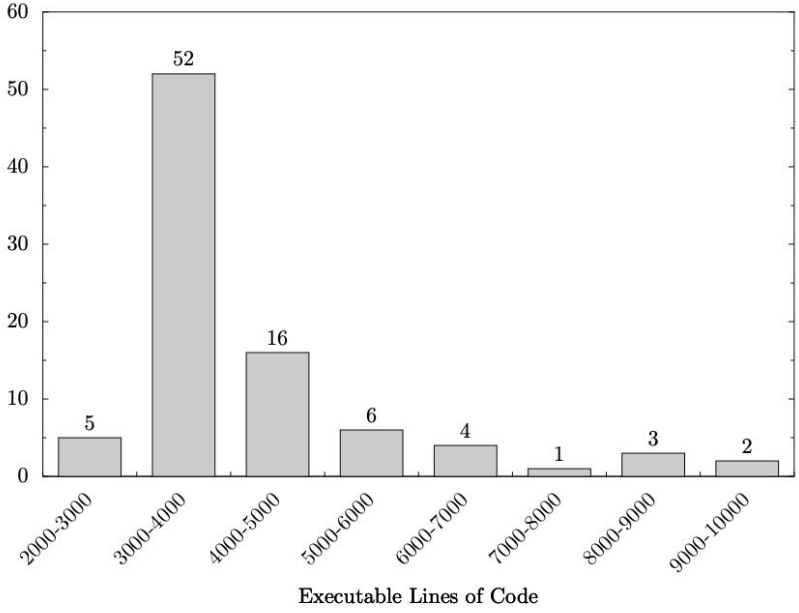


Figure 4: Histogram showing the number of COREUTILS tools that have a given number of executable lines of code (ELOC).

KLEE Performance - GNU Coreutils

	COREUTILS		BUSYBOX	
Coverage (w/o lib)	**KLEE tests**	**Devel. tests**	**KLEE tests**	**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	-	30
Overall cov.	84.5%	67.7%	90.5%	44.8%
Med cov/App	94.7%	72.5%	97.5%	58.9%
Ave cov/App	90.9%	68.4%	93.5%	43.7%

KLEE Performance - GNU Coreutils

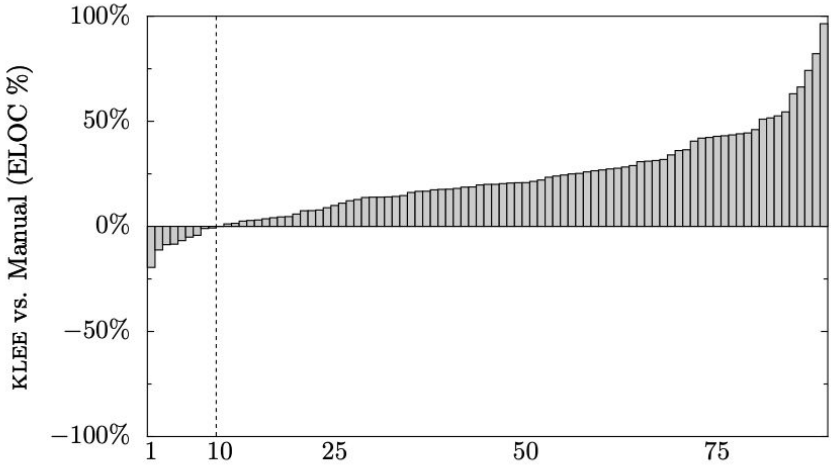


Figure 6: Relative coverage difference between KLEE and the COREUTILS manual test suite, computed by subtracting the executable lines of code covered by manual tests (L_{man}) from KLEE tests (L_{klee}) and dividing by the total possible: $(L_{klee} - L_{man})/L_{total}$. Higher bars are better for KLEE, which beats manual testing on all but 9 applications, often significantly.

KLEE as a bug finder

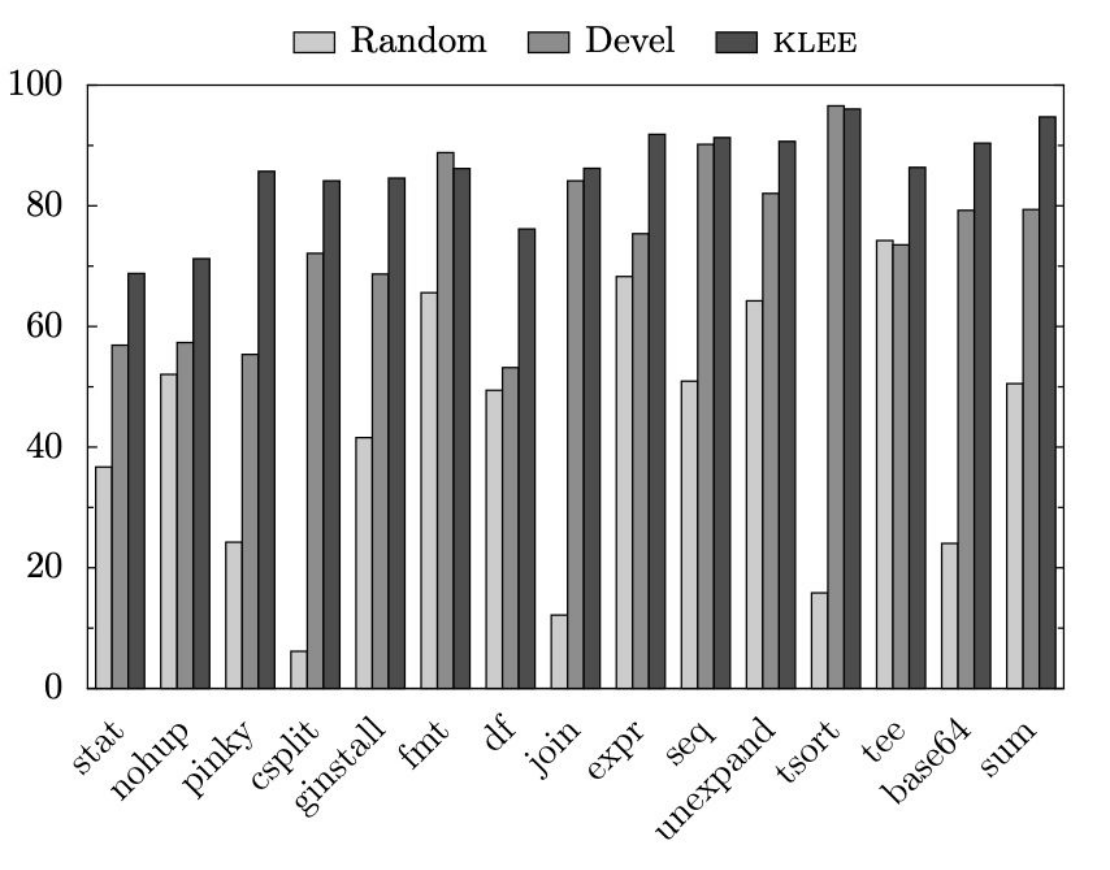
KLEE as a bug finder

```
paste -d\\ abcdefghijklmnopqrstuvwxyz
pr -e t2.txt
tac -r t3.txt t3.txt
mkdir -Z a b
mkfifo -Z a b
mknod -Z a b p
md5sum -c t1.txt
ptx -F\\ abcdefghijklmnopqrstuvwxyz
ptx x t4.txt
seq -f %0 1
```

t1.txt: "\t \tMD5("
t2.txt: "\b\b\b\b\b\b\t"
t3.txt: "\n"
t4.txt: "a"

KLEE tests vs random tests

KLEE tests vs random tests



KLEE usage on BusyBox Utilities

KLEE usage on BusyBox Utilities

- Researchers ran KLEE on the 75 utilities that make up BusyBox’s coreutils - with great success.

KLEE usage on BusyBox Utilities

- Researchers ran KLEE on the 75 utilities that make up BusyBox’s coreutils - with great success.
- Researchers used KLEE to find bugs in BusyBox.

Related Work

Related Work

- Other symbolic execution frameworks do not interact with the environment.

Related Work

- Other symbolic execution frameworks do not interact with the environment.
- More recent tools can interact with the environment but are limited.

Related Work

- Other symbolic execution frameworks do not interact with the environment.
- More recent tools can interact with the environment but are limited.
- More related work is interested in the path-explosion problem.

Concluding Remarks

Discussion