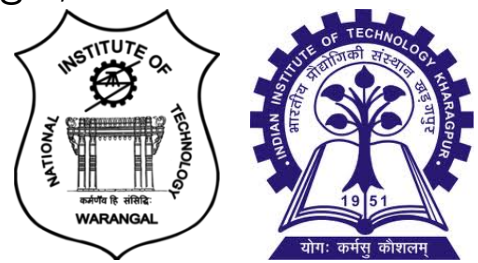
The KLEEMA Prototype

Sangharatna Godboley[1], G. Monika Rani[2], and Arpita Dutta[3]

NIT Warangal, India[1,2] and IIT Kharagpur, India[3]



KLEE Workshop 2021

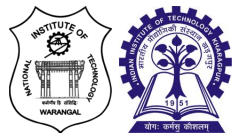
Organized by the Software Reliability Group at Imperial College London,
10-11 June, 2021, Online event*

May 28, 2021

Plan of the talk

Idea

KLEEMA = KLEE + Mutation Analyser



Use

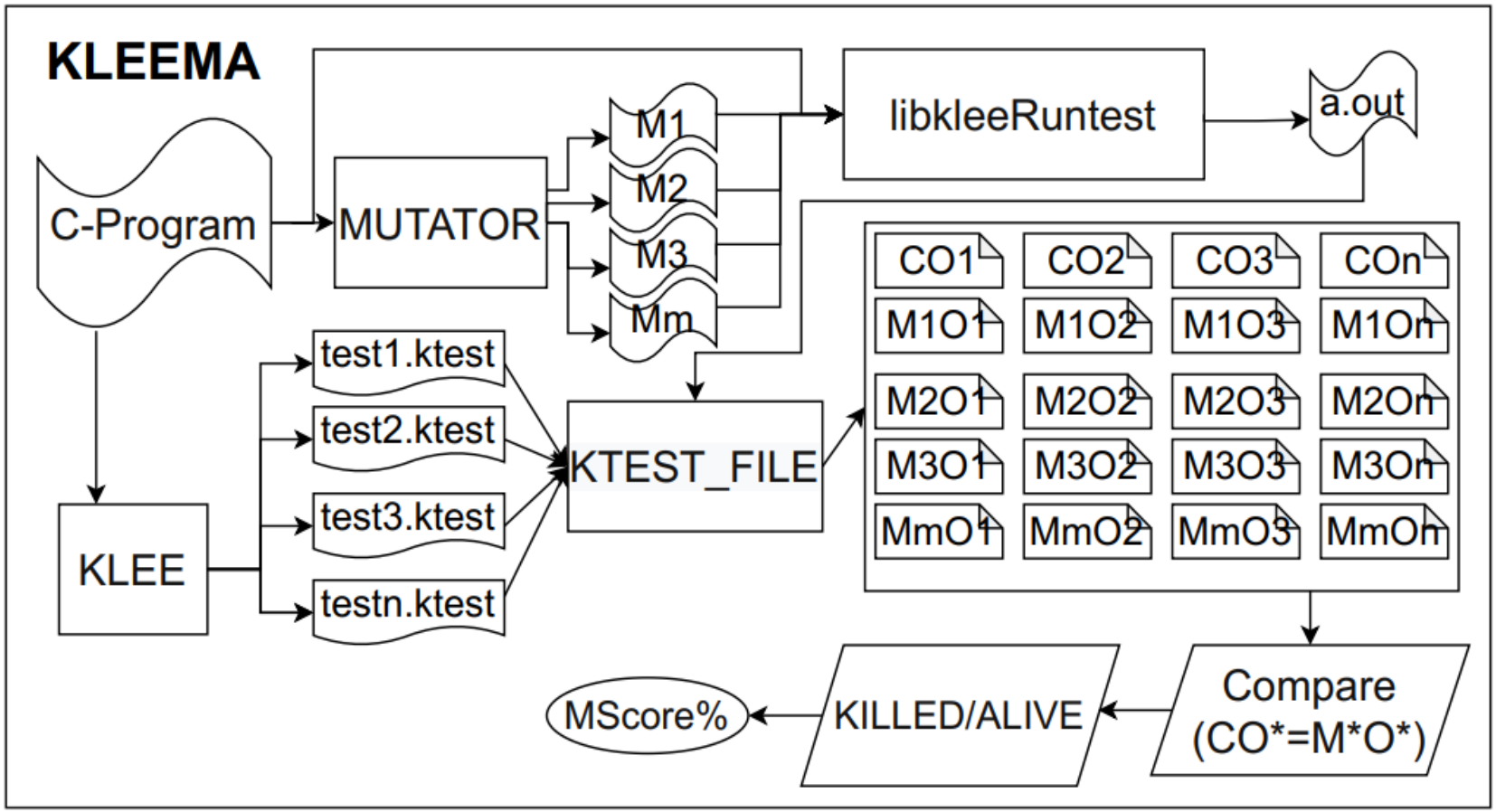
- Mutation Testing measures the quality of test inputs
- Mutation Testing is flexible with the injection of artificial faults
- KLEEMA can be plugged with any Dynamic Symbolic Executor (DSE), developed on top of KLEE
- KLEEMA helps in comparing test suites generated by two or more DSEs


Fault Types

Current version of KLEEMA supports following types of faults.

- Logical Operator Faults (LOF)
- Arithmetic Operator Faults (AOF)
- Relational Operator Faults (ROF)
- Literal Negation Faults (LNF)
- Predicate Negation Faults (PNF)


Framework


KLEEMA
C-Program
MUTATOR
M1
M2
M3
Mm
libkleeRuntest
a.out
CO1
CO2
CO3
COn
M1O1
M1O2
M1O3
M1On
M2O1
M2O2
M2O3
M2On
M3O1
M3O2
M3O3
M3On
MmO1
MmO2
MmO3
MmOn
test1.ktest
test2.ktest
test3.ktest
testn.ktest
KTEST_FILE
KLEE
Compare
(CO*=M*O*)
KILLED/ALIVE
MScore%

Details

- To achieve a maximal Mutation Score for a program, one needs to use KLEE and generate test cases.
- But, utilisation of generated test cases (.ktest format) by KLEE is not straight forward.
- There is a replay process of test cases described in KLEE website[a].

[a] https://klee.github.io/tutorials/testing-function/

Details

- KLEE has a replay library, which simply replaces the call to *klee_make_symbolic* with a call to a function that assigns to input the value stored in the *.ktest* file.
- To use replay tool, one needs to link the program with the *libkleeRuntest* library and set the *KTEST_FILE* environment variable to point to the name of the desired test case as shown in Listing 1.


Listing 1: Systematically replay a .ktest with a C-Program and store the output.

```
$ export LD_LIBRARY_PATH=path-to-klee-build-dir/lib/
    :$LD_LIBRARY_PATH
$ gcc -I ../../include -L path-to-klee-build-dir/lib/
    C-Program.c -lkleeRuntest
$ KTEST_FILE=test1.ktest ./a.out > CO1.txt
$ KTEST_FILE=test2.ktest ./a.out > CO2.txt
$ KTEST_FILE=test3.ktest ./a.out > CO3.txt
.....
$ KTEST_FILE=testn.ktest ./a.out > COn.txt
```


Results

Table 1: Results on sample programs

Programs	LOCs	#TM	#DM	#RM	#AM	#KM	MS
sample.c	98	90	39	51	9	42	82%
P2-L-T-R16.c	98	225	102	123	62	61	49%
tcas.c	293	218	0	218	122	96	44%
test23-B5.c	4677	31455	25015	6440	2480	3960	61%

Where, TM=Total Mutants, DM=Dead Mutants, RM=Reachable Mutants, AM=Alive Mutants, KM=Killed Mutants, MS=Mutation Score

Demo

Let's run a sample program for Demo

Code

`https://github.com/sanghu1790/KLEEMA/tree/master`

References ⋙ I

1. Cristian Cadar, Daniel Dunbar, and Dawson R Engler. *KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs*. In OSDI, pages 209-224, 2008.

2. A Jefferson Offutt and Roland H Untch. *Mutation 2000: Uniting the orthogonal*. In Mutation testing for the new century, pages 34-44. Springer, 2001.


Thank You!