

# Symbiotic: finding bugs in C programs

---

Marek Chalupa, Masaryk University

DevConf 2019 , 25. 1. 2019

- Nowadays, we usually use testing to find bugs.
- It is hard to write tests that reveal bugs.
- Automatic test generation can be used.

# Symbolic Execution

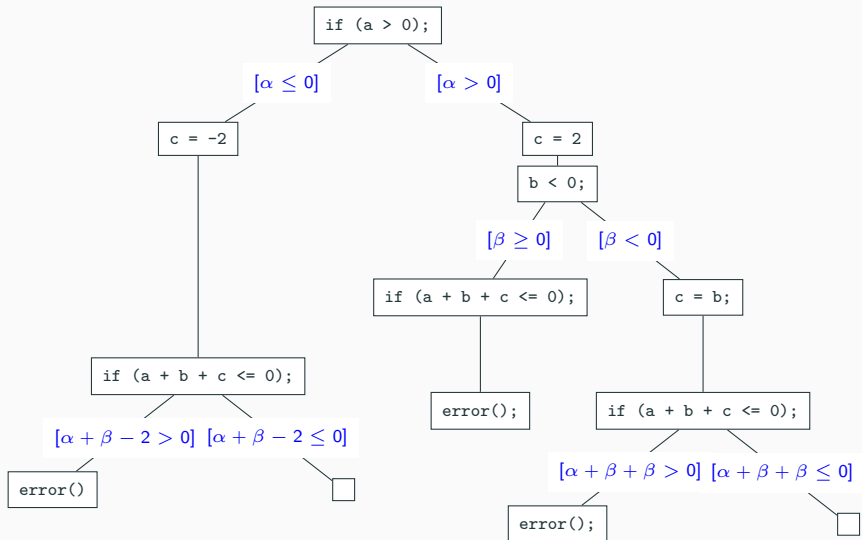
- Given a program with inputs:
  - use symbols instead of inputs,
  - execute the program with these symbols,
  - fork the execution on branchings.
- This way we can enumerate all possible paths in the program.

## Symbolic execution

```
void foo(int a, int b, int c)
{
    if (a > 0) {
        c = 2;
        if (b < 0) {
            c = b;
        }
    } else {
        c = -2;
    }
    if (a + b + c <= 0)
        error();
}
```

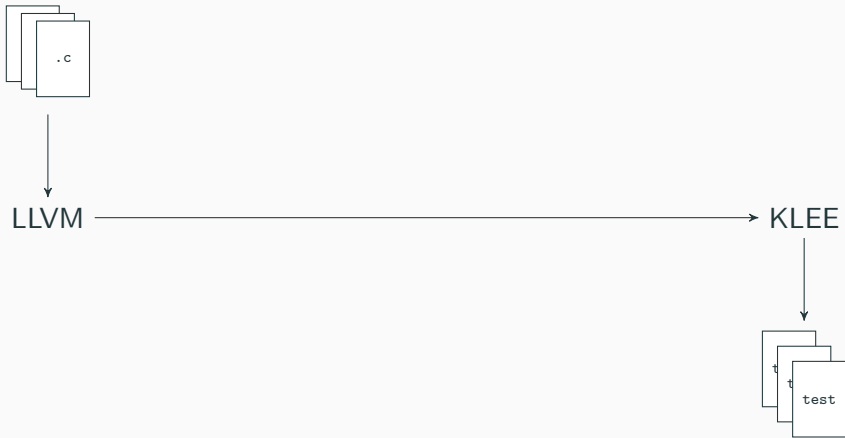
# Symbolic execution

$a := \alpha, b := \beta, c := \gamma$



- Open-source symbolic executor for LLVM bitcode.
- <http://klee.github.io/>





Symbolic execution is computationally very demanding.



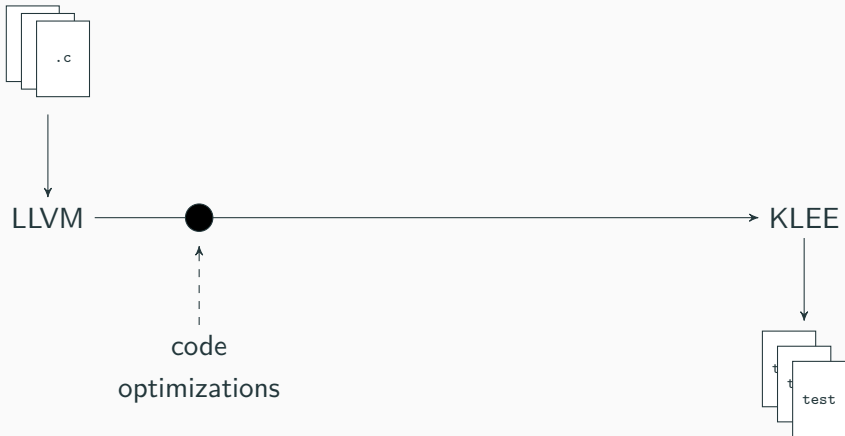
Symbolic execution is computationally very demanding.

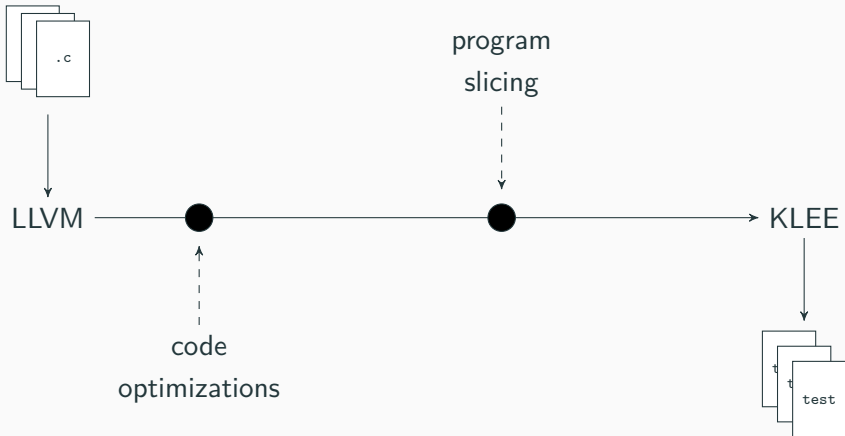


Preprocess the code before giving it to KLEE.



# Symbiotic





- Compute dependencies between instructions.
- We say that instruction A depends on instruction B if:
  - instruction A uses values generated by instruction B, or
  - instruction A is not executed if we go some other way at (branching) B.
- Slicing: keep only the instructions on which the error (transitively) depends.

## Program Slicing – Example

```
int zeroing(char *buf, size_t size)
{
    int n = input();
    for (int i = 0; i < n; ++i) {
        assert(i < size && "Out_of_bounds");
        buf[i] = 0;
    }

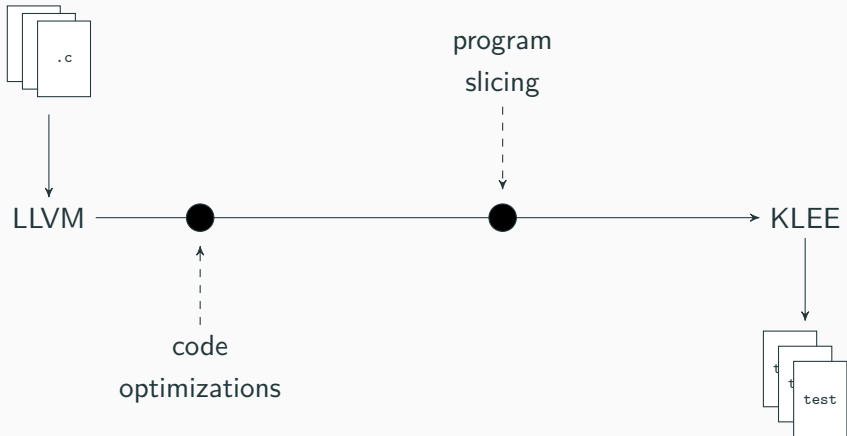
    return 0;
}
```

## Program Slicing – Example

```
int zeroing(char *buf, size_t size)
{
    int n = input();
    for (int i = 0; i < n; ++i) {
        assert(i < size && "Out_of_bounds");
        buf[i] = 0;
    }

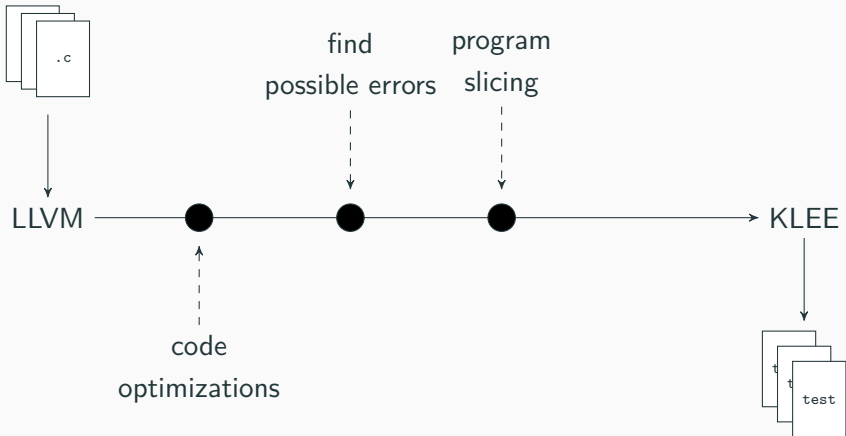
    return 0;
}
```

## Symbiotic - cont.





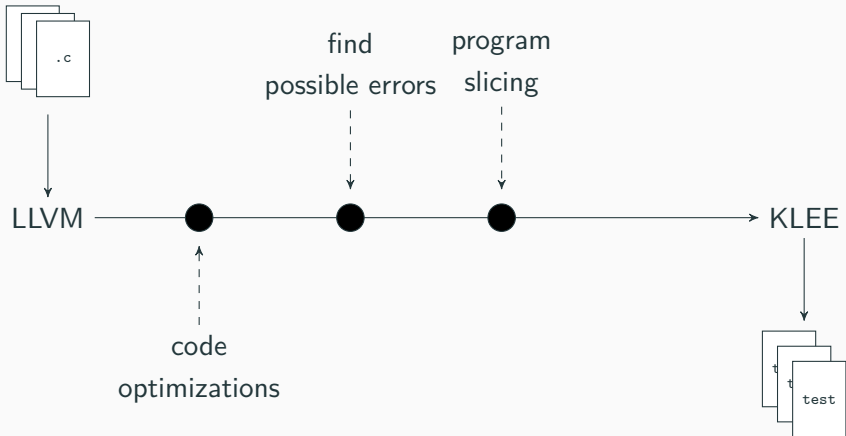
## Symbiotic - cont.



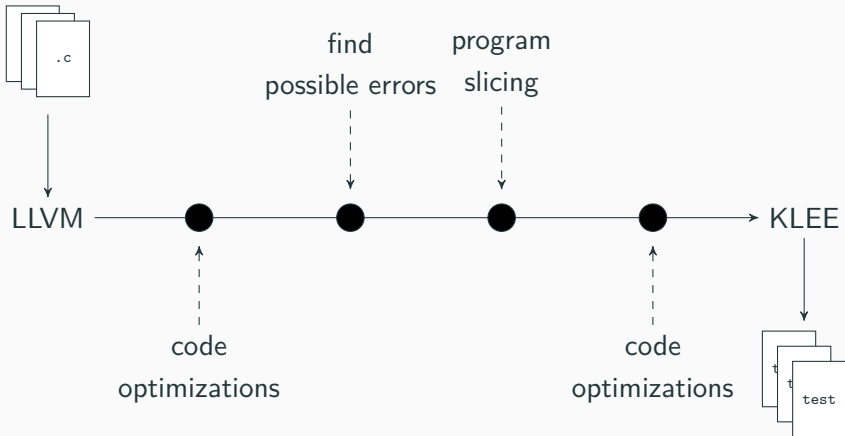
## Finding possible errors

- Symbiotic performs light-weight static analysis before slicing.
- The static analysis looks for chosen errors (integer overflow, double free, dangling pointer dereference,...).
- Instructions that *may* exhibit an error are set as slicing criteria.

## Symbiotic - cont.



## Symbiotic - cont.



- Apart from the already mentioned steps, Symbiotic:
  - automatically marks memory symbolic,
  - replaces undefined functions with symbolic stubs.
- Limits: no C++ (exceptions), no threads (yet).
- Unfortunately, Symbiotic still does not scale to large programs.

# Future Directions

- Scalability
  - slicing (faster analyses),
  - symbolic execution (abstraction),
  - different back-ends than KLEE.
- Better modelling of the environment (POSIX).
- C++ and threads.

# Conclusion

- Symbiotic is a tool for finding bugs in C programs.
- Combines fast static analysis with program slicing and symbolic execution.
- Runs on sequential C code.
- Still needs some work
  - scalability issues,
  - no C++ and threads.

# Conclusion

- Symbiotic is a tool for finding bugs in C programs.
- Combines fast static analysis with program slicing and symbolic execution.
- Runs on sequential C code.
- Still needs some work
  - scalability issues,
  - no C++ and threads.

<https://github.com/staticafi/symbiotic>

Thank you!