

Automatic Unit-Test Generation Based on Symbolic Execution

Tigran Avagyan, Hovhannes Movsisyan,
Levon Shahbazyan, Hayk Aslanyan

Center of Advanced Software Technologies, Yerevan, Armenia

**Fully automated unit-test generation for C programs
using symbolic execution**

- Writing high-coverage unit tests is time-consuming and resource-intensive.
- Edge cases and pointer-heavy code are especially hard to test manually.
- Existing symbolic execution tools such as KLEE require manual setup or manual symbolic memory region markings.

2

Key Contributions

- 1 Fully automated workflow from source code + target function to generated unit tests.
- 2 Automatic symbolization of global variables, function parameters, external function returns, and pointer arguments passed to external functions.
- 3 KLEE-based symbolic execution with improved memory model and symbolic pointer support.
- 4 Pointer-aware reconstruction of symbolic memory using a “pointer forest”.
- 5 Per-path unit-test generation with mock functions and generated main functions.

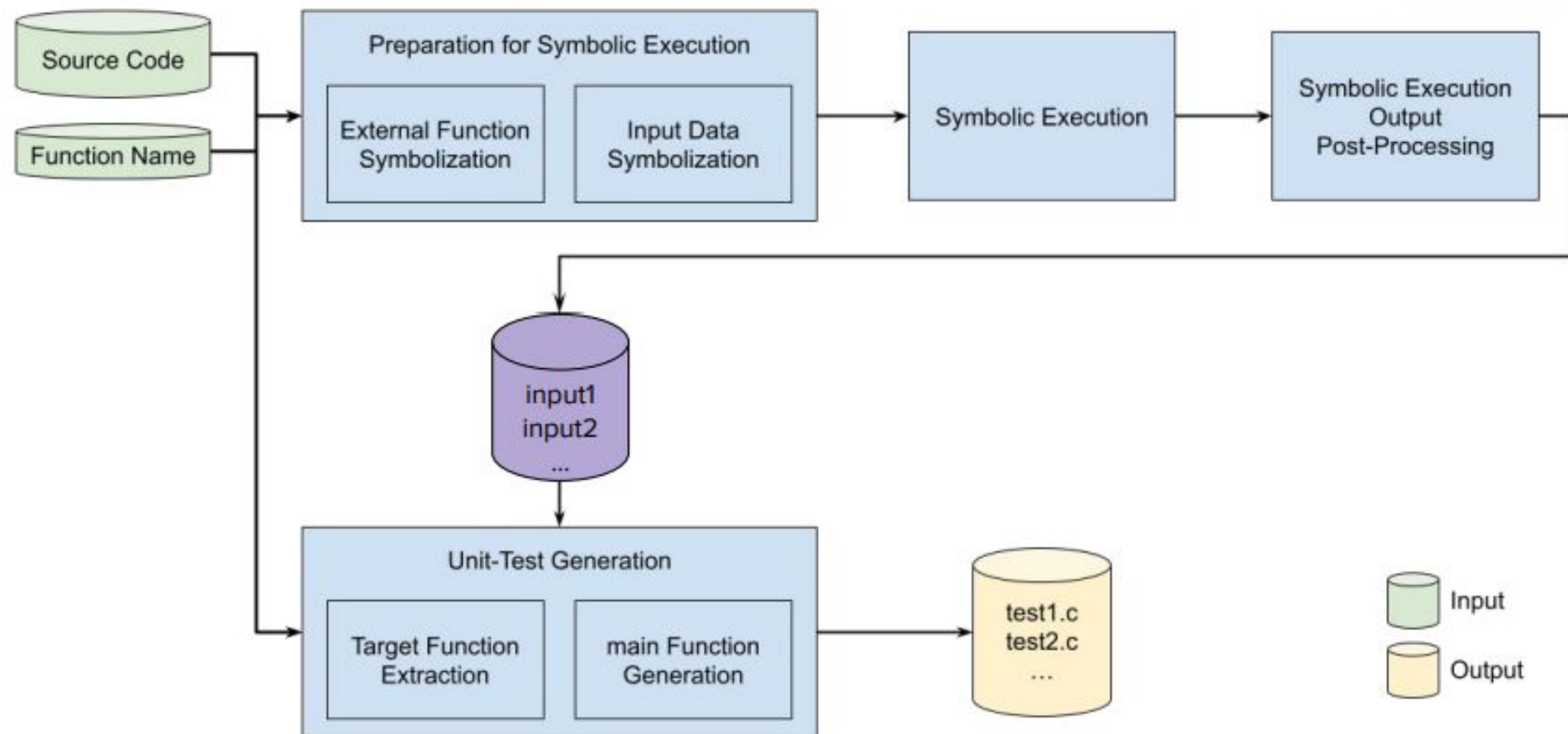
3

Comparison with KLEE

Feature	KLEE	Our Approach
Symbolization	Manual	Fully automated
Symbolic pointers	Insufficient support	Full support

Our approach removes manual setup and extends symbolic execution with practical symbolic-pointer handling for automated test generation.

The Architecture



5

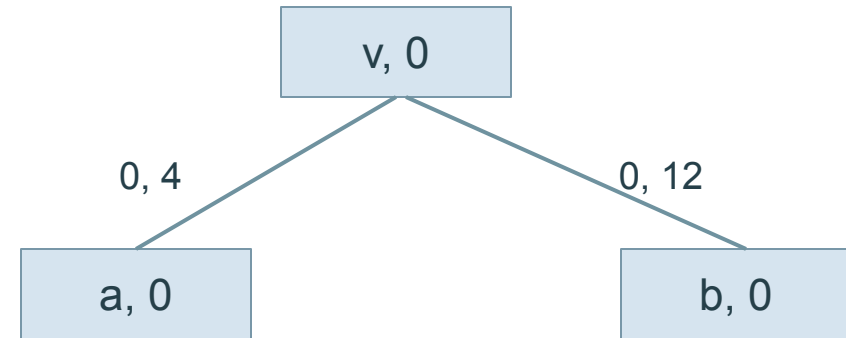
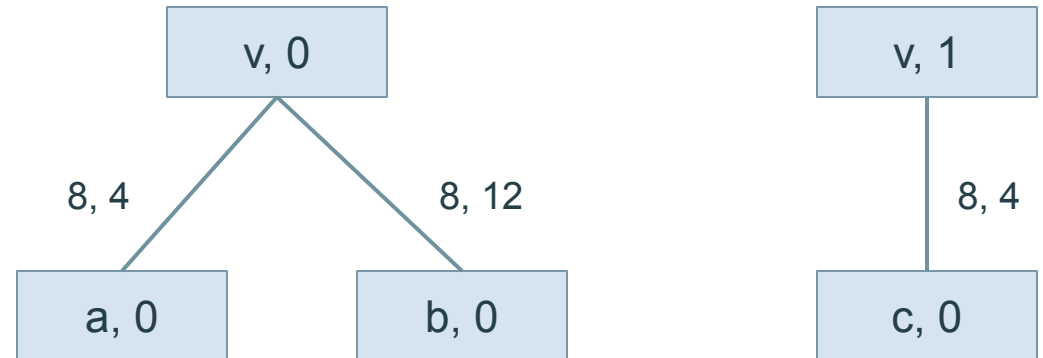
Pointer Forest

Example 1

```
void foo(int* v) {  
    int a = v[1];  
    int b = v[3];  
}
```

Example 2

```
struct Vec {  
    int64_t size;  
    int* data;  
} v;  
void f(int** x);  
void foo() {  
    int a = v.data[1];  
    int b = v.data[3];  
    f(&v.data);  
    int c = v.data[1];  
}
```



- Nodes: memory object + symbolization index.
- Edges: offset pairs for reconstructing pointer-linked inputs.
- The forest supports both simple and multi-level pointer relations.

6

Unit Test Generation

Target function

```
1. void foo(int* x, int** y) {  
2.     if (*x == y[1][3] && x[1] == 7)  
3.         assert(0 && "invalid state");  
4. }
```

Generated test

test1.c	test2.c	test3.c
1. int main() { 2. int32_t *x = malloc(2 * sizeof(*x)); 3. x[0] = 0; 4. x[1] = 7; 5. int32_t **y = malloc(2 * sizeof(*y)); 6. int32_t *y_1 = malloc(4 * sizeof(*y_1)); 7. y_1[3] = 0; 8. y[1] = y_1; 9. foo(x, y); 10. free(y_1); 11. free(y); 12. free(x); 13. }		

Supported Features

Feature	Status
Primitive types	Full support
Pointers	Full support
Arrays	Full support
Pointer to struct	Full support
Small structs (ABI lowering)	Full support
User-defined functions (pointer arguments)	Full support
Library functions (pointer arguments)	Full support
Unions	Partial support
Unnamed structs	Partial support
Function pointers	Not supported yet

- ✓ The approach automatically generates C unit tests by exploring execution paths.
- ✓ The workflow is fully automated.
- ✓ The pointer model reconstructs pointer-linked input data for generated tests.
- **Future work:** Improve unit test readability and add support for function pointers.

Thank you!

Questions?



H. Movsisyan
hovhannes.movsisyan@rau.am



Center of Advanced Software Technologies