UTBot Simplifies Auto Test Generation

Samat Gaynutdinov, Saveliy Grigoryev, Pavel Iatchenii, Elena Ilina, Dmitry Ivanov, Vladislav Kalugin, Aleksei Pleshakov, **Pavel Ponomarev**, Konstantin Rybkin, Svetlana Shmidt, Vadim Volodin, Alexey Utkin

KLEE for testing C code

To create a test case for a function you would need to:

- Configure a project
- Introduce KLEE entry point with symbolic variables
- Build the project in LLVM IR
- Run KLEE
- Parse KLEE output for generated test cases
- Write the test code based on parsed KLEE output
- Run the test cases

The process requires a lot of time!

It is hard to generate tests for real-world projects with KLEE

Solution: UTBot for C

- Test generation process automation
- Multiple test generation scenarios:
  - for a whole project;
  - for a folder/file/function;
  - for a line (i.e., generate a test that executes the specific line);
  - for an assert (i.e., generate test that fails given assert);
  - with an expected return value.
- Support for C
  - including floats, data types, function pointers and recursive types
- Key features
  - Context definition, stubs generation, running tests, coverage calculation
- Basic C++ support

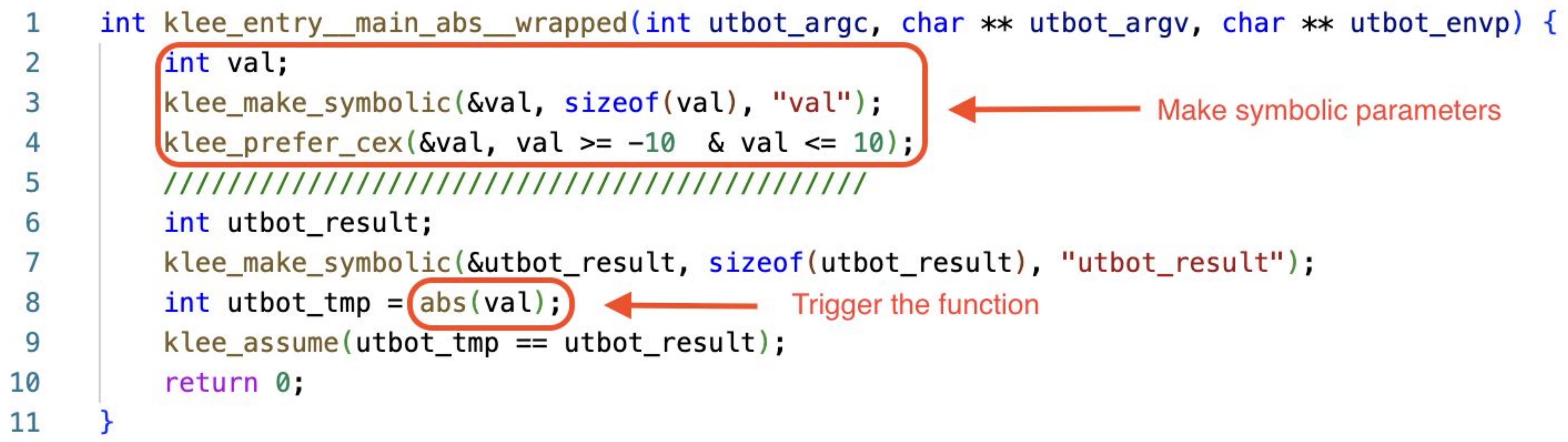
How UTBot works

Prepare a project

Project Type	**Project preparation**
CMake project	UTBot runs **patched CMake** to generate both `compile_commands.json` and `link_commands.json`
Make project	UTBot runs **patched Bear** to generate both `compile_commands.json` and `link_commands.json`
Other	UTBot asks user to run `bear $BUILD_COMMAND`

- **`compile_commands.json`** — **compilation database**, compilation commands for libraries and/or executables
- **`link_commands.json`** — **linkage database**, linking commands for libraries and executables **(specific to UTBot)**

Prepare KLEE run

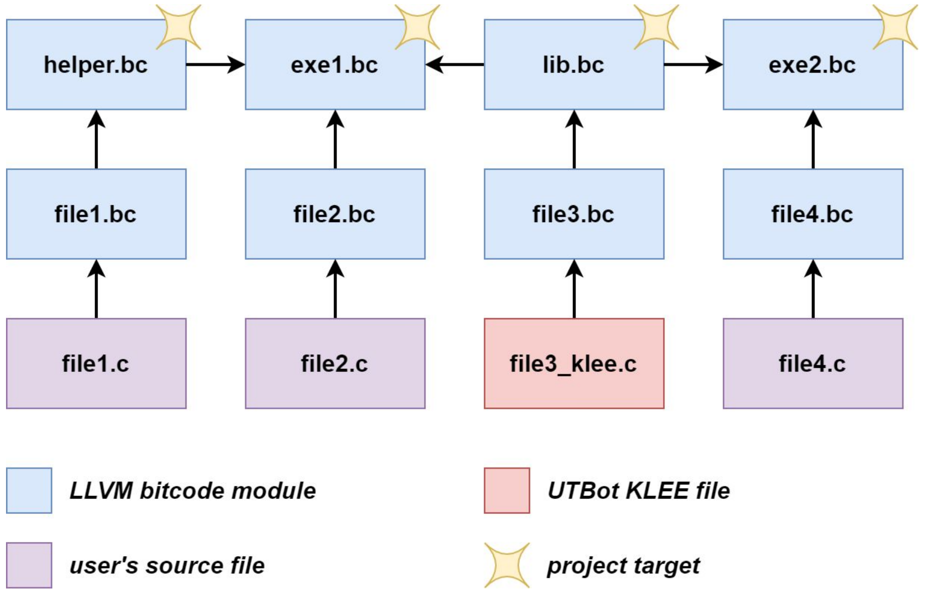


```c
int klee_entry__main_abs__wrapped(int utbot_argc, char ** utbot_argv, char ** utbot_envp) {
    int val;
    klee_make_symbolic(&val, sizeof(val), "val");
    klee_prefer_cex(&val, val >= -10  & val <= 10);
    ////////////////////////////////////////////
    int utbot_result;
    klee_make_symbolic(&utbot_result, sizeof(utbot_result), "utbot_result");
    int utbot_tmp = abs(val);
    klee_assume(utbot_tmp == utbot_result);
    return 0;
}
```

Prepare bitcode and run KLEE

KLEE patches

- Speed
  - **Pruning The Recursive States***
  - Weakest Precondition in Symbolic Execution
- Code coverage
  - Floating-point Support
  - **Complex Test Input Generation***
  - **Detection of Undefined Behavior***

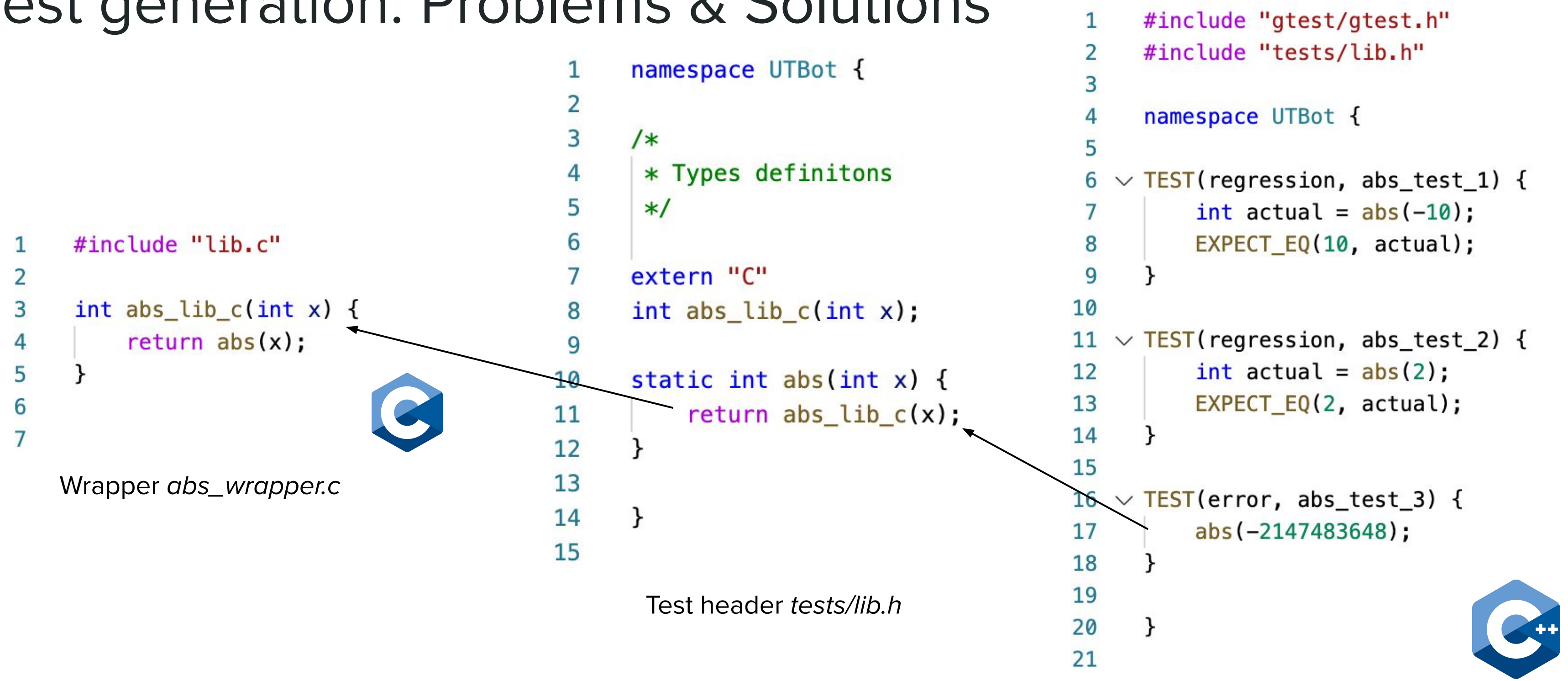


* presented at KLEE Workshop 2022

Google Test generation

```
TEST(regression, abs_test_1) {
    // Construct input
    int val = 0;
    // Expected output
    int expected = 0;
    // Trigger the function
    int actual = abs(val);
    // Check results
    EXPECT_EQ(expected, actual);
}
```

Test generation: Problems & Solutions



```c
#include "lib.c"

int abs_lib_c(int x) {
    return abs(x);
}
```

Wrapper *abs_wrapper.c*

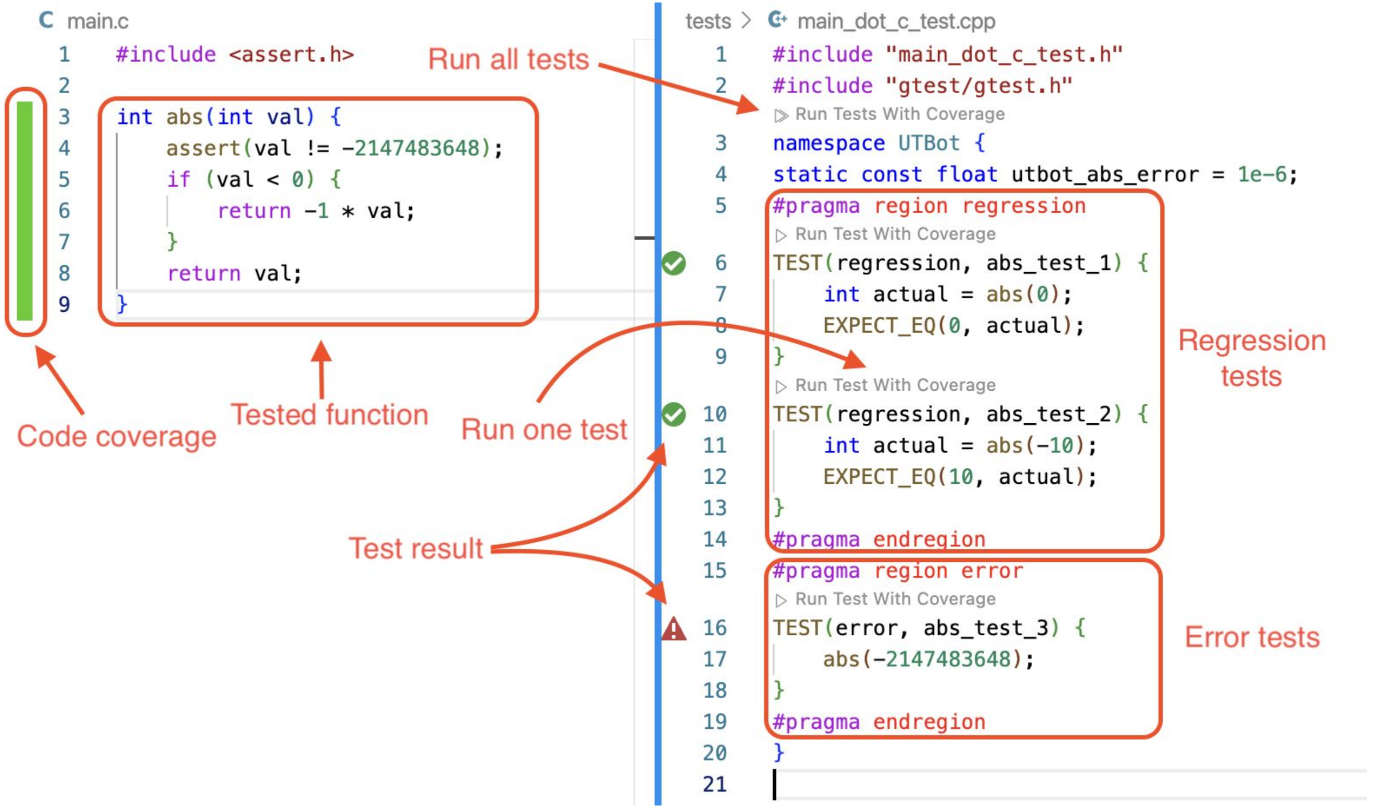
```cpp
namespace UTBot {

/*
 * Types definitons
 */

extern "C"
int abs_lib_c(int x);

static int abs(int x) {
    return abs_lib_c(x);
}

}
```

Test header *tests/lib.h*

```cpp
#include "gtest/gtest.h"
#include "tests/lib.h"

namespace UTBot {

TEST(regression, abs_test_1) {
    int actual = abs(-10);
    EXPECT_EQ(10, actual);
}

TEST(regression, abs_test_2) {
    int actual = abs(2);
    EXPECT_EQ(2, actual);
}

TEST(error, abs_test_3) {
    abs(-2147483648);
}

}
```

Test file *tests/lib.cpp*

Compile and run tests


main.c

```c
#include <assert.h>

int abs(int val) {
    assert(val != -2147483648);
    if (val < 0) {
        return -1 * val;
    }
    return val;
}
```

tests > main_dot_c_test.cpp

```cpp
#include "main_dot_c_test.h"
#include "gtest/gtest.h"
▷ Run Tests With Coverage
namespace UTBot {
static const float utbot_abs_error = 1e-6;
#pragma region regression
▷ Run Test With Coverage
TEST(regression, abs_test_1) {
    int actual = abs(0);
    EXPECT_EQ(0, actual);
}
▷ Run Test With Coverage
TEST(regression, abs_test_2) {
    int actual = abs(-10);
    EXPECT_EQ(10, actual);
}
#pragma endregion
#pragma region error
▷ Run Test With Coverage
TEST(error, abs_test_3) {
    abs(-2147483648);
}
#pragma endregion
}
```

Conclusion

Results

- Built UTBot for C — tool for auto tests generation
- Complemented KLEE with a user-friendly interface
- Improved KLEE functionality: speed & coverage patches

Future work

- Full C++ support
- CLion integration
- CI integration

References

- UTBot (UnitTestBot) project website
  https://www.utbot.org
- UTBot C/C++ project page on GitHub
  https://github.com/UnitTestBot/UTBotCpp
- UTBot C/C++ documentation
  https://www.utbot.org/docs/cpp/general/home