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

Generating C/C++ tests with KLEE is hard. UTBot does the work!

1. 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
link_commands.json — linking commands for libraries and executables (specific to UTBot)

2. Prepare KLEE run

During the analysis phase, UTBot uses information about the declared top-level functions and types collected while opening the project. UTBot wraps the functions for KLEE.

```
int klee_entry__main_abs__wrapped(int _argc,
                                  char ** _argv, char ** _envp) {
    int val;
    klee_make_symbolic(&val, sizeof(val), "val");
    klee_prefer_cex(&val, val >= -10  & val <= 10);
    int _result;
    klee_make_symbolic(&_result, sizeof(_result), "_result");
    int _tmp = abs(val);
    klee_assume(_tmp == _result);
    return 0;
}
```

Make symbolic parameter

Trigger the function

3. Google Test generation

KLEE returns test case descriptions for a function that was previously specified as an entry point. UTBot performs the transformation of KLEE output into the Google Test format.

```
▷ Run Test With Coverage
TEST(regression, strToTime_test_1) {
    char s_buffer[] = "00:00";
    const char * s = s_buffer;
    struct Time expected = {
        .hour = 0U,
        .minutes = 0U
    };
    struct Time actual = strToTime(s);
    EXPECT_EQ(expected.hour, actual.hour);
    EXPECT_EQ(expected.minutes, actual.minutes);
}
▷ Run Test With Coverage
TEST(error, strToTime_test_2) {
    char s_buffer[] = "90:00";
    const char * s = s_buffer;
    strToTime(s);
    FAIL() << "Function was supposed to fail";
}
```

Construct expected output

Construct parameters

Trigger the function

4. Compile and run tests

There are plugins for VS Code and CLion, GitHub Actions.

```
#include <assert.h>

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

Tested function

Run all tests

```
#include "main_dot_c_test.h"
#include "gtest/gtest.h"
▷ Run Tests With Coverage
namespace UTBot {
▷ 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);
}
▷ Run Test With Coverage
TEST(error, abs_test_3) {
    abs(-2147483648);
}
}
```

Regression

Error

Coverage

Run single test

Test result

UTBot allows compiling test files to link them with the rest of the code and to run the tests to get results and information about code coverage. UTBot can also be used as a zero false positive static analyzer by generating a SARIF report.

Enhancements in KLEE

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 the 3rd International KLEE Workshop, 2022**

References

https://www.utbot.org
https://github.com/UnitTestBot/UTBotCpp
https://github.com/UnitTestBot/klee

3rd International KLEE Workshop on Symbolic Execution 15–16 September 2022 • London, UK