Presentation: 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, and Alexey Utkin

UnitTestBot

July 2022

1 Introduction

Many developers find KLEE too complicated for usage and integration into projects. To generate tests, one needs to manually navigate through difficult process of building the project in LLVM IR, running KLEE and parsing KLEE complex output. Moreover, users need to modify their code to introduce symbolic variables that is pretty inconvenient. The article introduces UTBot – tool for simple test generation in human readable format without user involved.

2 Pipeline

2.1 Opening a project

When the user opens a C project in VS Code with the enabled UTBot, the plugin extracts information about the project structure. It uses this information to collect top-level entities and rebuild the project into LLVM IR.

UTBot uses a compilation database format to store information. There are two common ways to generate the `compile_commands.json` file for the project: (a) to get it from CMake by enable `CMAKE_EXPORT_COMPILE_COMMANDS` option in it or (b) to use Bear utility.

The resulting `compile_commands.json` file contains information about the compilation commands for the individual source files. That is not enough to rebuild the project. To build the tests as the executable units, UTBot needs information about the project's linking commands. Our custom `link_commands.json` file uses the same format as the `compile_commands.json` file does, but it also includes missing information about the linkage stage.

UTBot comes with modified CMake and Bear tools that support generation of `link_commands.json` files.

2.2 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. Consider the following function as an example: `int max(int a, int b)`. The UTBot-generated wrapper file for KLEE analysis is the following:

```
int klee_entry__max(int argc, char ** argv, char ** envp) {
  int a;
  klee_make_symbolic(&a, sizeof(a), "a");
  klee_prefer_cex(&a, a >= -10  & a <= 10);
  int b;
  klee_make_symbolic(&b, sizeof(b), "b");
  klee_prefer_cex(&b, b >= -10  & b <= 10);
  int res;
  klee_make_symbolic(&res, sizeof(res), "res");
  int utbot_tmp = max(a, b);
  klee_assume(utbot_tmp == result);
  return 0;
}
```

2.3 Prepare bitcode and run KLEE

To build the project into LLVM IR, UTBot filters the `compile_commands.json` and `link_commands.json` files to find the appointed library or executable and modifies the build commands from `binary` to a `bitcode` format. UTBot runs modified KLEE on this bitcode file.

UTBot introduces several enhancements to KLEE, e.g. "Guided KLEE", "Weakest precondition in symbolic execution" to speed up KLEE and "Lazy initialization", "Floating-point support" to increase code coverage.

2.4 Google Test files generation

As a result of its work KLEE returns test case descriptions for each project function that was previously specified as an entry point. UTBot performs a complex transformation of KLEE output into the Google Test format. UTBot uses this testing framework for its popularity and compatibility with both C and C++. KLEE returns raw byte values for symbolic objects, while UTBot is able to construct the human-like code assigning these bytes to the variables. For instance, UTBot sets structure fields one by one with the assignment operators instead of performing a single `memcpy()` call.

```
TEST(regression, max_test_1)
{
    // Construct input
    int a = 0;
    int b = -1;
    // Expected output
    int expected = 0;
    // Trigger the function
    int actual = max(a, b);
    // Check results
    EXPECT_EQ(expected, actual);
}
```

2.5 Compile and run tests

UTBot allows to compile test files, link them with the rest of the code and run the tests to get results and information about code coverage. When linking generated tests UTBot solves the following problems:

- If the user code contains a C static function, the generated tests binary cannot be linked, because the static functions are available only in the source file they are declared. To handle this problem, UTBot generates wrapper files with the non-static functions that include original files via the `#include` directive. The functions in these wrapper files have the same signatures as the original static functions and call them in their bodies.

  ```
  #include "/home/utbot/example/abs.c"
  int max_main_c(int a, int b) {
      return max(a, b);
  }
  ```

- C++ has some keywords, which are absent in C, for example, `try` or `class`. In C these words can serve as the function/variable names. In the test headers UTBot guards the user function wrappers with a `#define` and `#undef` macro and renames them so that they do not clash with any C++ specific keyword.

3 Conclusion

UTBot is a tool for easy tests generation and integration. UTBot complements KLEE's work on generating complete and applicable test cases with a user-friendly interface. Moreover, UTBot improves some KLEE functionally that also raises user experience.

4 References

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