Introduction. Input generation is an important testing approach to find inputs reaching interesting, e.g., buggy, program locations. The two main input generation techniques are blackbox and whitebox testings. Blackbox techniques such as fuzzing [3] generate random inputs to test programs whose source code might not be available. Fuzzing techniques are typically fast, but their randomly-generated inputs often fail to expose hard-to-reach, buggy locations. In contrast, whitebox techniques such as symbolic execution [1, 2] analyze the program structure to explicitly generate inputs to reach these difficult program locations.

In this project, we will study CIVL [4], a mature and powerful symbolic execution tool for C programs. CIVL supports normal, sequential C programs and parallel/concurrent C programs (e.g., using parallel models such as MPI, CUDA, etc). At high level, the CIVL framework consists of three parts: (i) a core programming language, CIVL-C, which is a C-like language extended with primitives to represent concurrency features, (ii) a checker that uses symbolic execution to verify a number of safety properties of C programs, and (iii) a number of translators to convert commonly-used concurrency languages/API’s to CIVL-C (e.g., MPI, OpenMP, PThreads, CUDA).

Proposed Works. We will perform three tasks for this project. First, we will describe the underlying technical approach used in CIVL. We will read at least 3 research papers describing the main ideas used in the tool (e.g., the CIVL-C language, the symbolic execution engine, and translators). Next, we will show how CIVL works with detailed examples. We will demonstrate the capabilities of CIVL through a wide-variety of interesting and challenging example programs (e.g., those from the CIVL’s own benchmark suite). We will also demonstrate the limitations of CIVL through example programs. Third, we will use CIVL to test a real-world program. We will search GitHub for a popular, medium-sized C program and apply CIVL over it to check for program assertion violations and common errors such as NULL-pointer dereferencing and division by zero.

Timeline. We have approximately 9 weeks to work on this project. We will use 2 weeks to find and read 3 research papers to understand CIVL. We will spend the next 2 weeks to find examples to demonstrate how CIVL works. For the next 3 weeks, we will apply CIVL to a real-world example. To help prepare for the presentation and final report, we will documenting all our findings and examples for each of these tasks. We use the final 2 weeks to complete our writing and prepare for the presentation.

References

