Benedek Mezei, 22-926-869

Model-checking the libcds library

Summary

Writing concurrent code is notoriously difficult, with many bugs occuring due to improper synchronization. Adding insult to the injury, modern platforms employ weak memory consistency models, i.e., the instructions of each thread might be reordered by the compiler and/or the CPU.

To make writing concurrent code less error-prone, developers often use concurrent libraries. These libraries implement standard concurrent data structures in a thread-safe manner, and aim to achieve optimized performance by incorporating fine-grained synchronization, elaborate memory reclamation algorithms, and contention reduction mechanisms. However, many of these libraries are not formally checked for correctness, so it's possible that there are executions that lead to undesired behavior.

To combat this, we aim to formally reason about the correctness of one such library (namely, libcds) by using the GenMC model checker. GenMC explores all possible states of a given program (up to a bound) to find states where an invariant is violated. To reduce the workload, GenMC avoids exploring states that only differ in the order in which independent instructions are executed, and is also equipped with various optimizations designed to aid in the verification of concurrent data structures.

In terms of development, this thesis involves work in three different areas: (a) providing a scaffolding for the library environment, (2) fixing any runtime issues that arise in GenMC (due to libcds being in C++ and GenMC being largely untested for C++), and (3) adding or modifying GenMC's user API so that all concurrency primitives of libcds are supported.

Core goals

- Familiarize with libcds and GenMC
- Create an environment for libcds so that GenMC can verify it:
  1) Provide custom APIs for any OS/compiler directives the library relies on using either stub methods or the provided GenMC API
  2) Ensure that GenMC's interpreter can handle the library code and fix any issues that arise
  3) In case the library relies on a concurrency API not properly supported by GenMC (e.g., hazard pointers), extend GenMC's API and add the corresponding support
- Prove functional correctness of the library (relaxed linearizability)

Extended goals

- Minimize the number of explored executions by using code annotations already provided by GenMC
- To further enhance scalability, add additional annotations like explicit linearization points
- Provide an extensive evaluation of GenMC (including the added annotations) on libcds