Adding a Rust frontend to GenMC

Thesis Description

Student: Arbenit Kamberi, 22-932-776

Supervisor: Prof. Dr. Michalis Kokologiannakis

Summary:

GenMC is a stateless model checker that can verify safety properties of concurrent C/C++ programs under a wide range of weak memory consistency models. It works at the level of LLVM Intermediate Representation (LLVM-IR): GenMC invokes clang to compile a program to LLVM-IR, and then repeatedly executes the resulting code, until all program behaviors have been explored.

To be able to execute the program under test, GenMC expects that the produced LLVM-IR output uses a concurrency API (e.g., for thread creation) that it can intercept, and ensures that this is the case by providing custom headers for standard C libraries such as pthread.h and stdatomic.h. Unfortunately, however, expecting a particular API means that other languages that also compile to LLVM-IR (e.g., Rust) are not immediately supported, as the respective compilers do not produce GenMC-compliant LLVM-IR out of the box.

In this thesis, we will extend GenMC so that it can verify Rust programs. Specifically, we will provide custom implementations of commonly used Rust concurrency primitives so that the resulting LLVM-IR code has a structure consistent with what GenMC expects (as done in C/C++), and evaluate the resulting frontend on a variety of benchmarks.

Core Goals:

- Familiarize with basic concepts:
    - Understand GenMC's architecture
    - Examine the produced LLVM-IR
    - Learn Rust and study Rust's standard concurrency APIs
- Verify concurrent Rust programs with only loads and stores. Compare results with the same programs written in C and see what already works.
- Provide custom implementations of as many Rust APIs as GenMC's LLVM-IR API can handle.
- Test and validate these custom implementations. Ensure they produce code that GenMC's runtime supports, and extend the runtime otherwise.

Extension Goals:

- User friendliness: users shouldn't include these custom implementations into their code themselves; if possible, this should be done automatically by GenMC. GenMC should also automatically choose the appropriate compiler frontend based on the program to be verified (C/C++ or Rust).
- Compatibility with existing transformations: port existing data-structure implementations in Rust and verify them using GenMC. Check whether the LLVM-IR optimizations that GenMC performs still apply, and repair or extend them in cases where they do not.
- Test suite: provide a comprehensive test suite covering all supported features.
- Case Study: verify an open-source Rust library, such as a library for concurrent data structures.
- Merge the resulting code into GenMC's codebase.