Optimizing GenMC's consistency-checking routines

Practical Work Description

Student: Arbenit Kamberi, 22-932-776

Supervisor: Prof. Dr. Michalis Kokologiannakis

Description:

GenMC[1] is a stateless model checker that can verify C/C++ and Rust programs under weak memory models. It works by enumerating possible executions, represented as graphs, while skipping equivalent executions. The consistency of an execution graph under a given weak memory model is checked to determine whether an exploration step should be taken. These measures ensure GenMC is optimal in the number of executions it explores.

Checking graph consistency boils down to cycle detection in graphs that are incrementally expanded over time, using a specialized algorithm based on DFS. The code implementing it is generated using a separate tool named Kater[2], allowing for quick changes through meta-programming to fit with GenMC's requirements without having to touch the algorithm.

Checking the consistency of execution graphs has been identified as a major performance bottleneck in GenMC. For this reason, several optimizations need to be implemented for this algorithm to improve GenMC's general performance. The supervisor has already identified a list of such possible optimizations. The task for the student is to optimize the implementation of this algorithm.

[1]Michalis Kokologiannakis and Viktor Vafeiadis, "GenMC: A Model Checker for Weak Memory Models", CAV 2021, pp 427-440, DOI: https://doi.org/10.1007/978-3-030-81685-8_20

[2]Michalis Kokologiannakis, Ori Lahav, and Viktor Vafeiadis, "Kater: Automating Weak Memory Model Metatheory and Consistency Checking", Pro. ACM Program. Lang. 7, POPL, Article 19, DOI: https://doi.org/10.1145/3571212

Core Goals:

- Familiarization with:
  - GenMC's[1] core modules.
  - How GenMC builds execution graphs and their structures over time.
  - What defines a "consistent" graph.
  - Structure of the Kater[2] tool.
  - The algorithm for determining consistency of a graph.
- Implement a list of possible optimizations previously identified by the supervisor, examples include:
  - Not clearing the "visited" array after every run.
  - Refactor and clean up code-printing infrastructure
  - Replace recursion with a stack to avoid stack-overflow issues
- Evaluate the runtime of the new implementation against the existing one.

Extension Goals:

- Further optimizations like:
  - Simplifications in certain acyclic DFS calculations (statically determined by Kater).
  - For inclusion checks ($r_1 \subseteq r_2$), do DFS on the RHS lazily (only if accepting runs on the LHS exist)
- Identify, implement and evaluate other possible optimizations.
- Merge the end-result into the existing codebase. Adhering to all coding-standards set by the lab.

[1]Michalis Kokologiannakis and Viktor Vafeiadis, "GenMC: A Model Checker for Weak Memory Models", CAV 2021, pp 427-440, DOI: https://doi.org/10.1007/978-3-030-81685-8_20

[2]Michalis Kokologiannakis, Ori Lahav, and Viktor Vafeiadis, "Kater: Automating Weak Memory Model Metatheory and Consistency Checking", Pro. ACM Program. Lang. 7, POPL, Article 19, DOI: https://doi.org/10.1145/3571212