Consistent Execution Construction from C/C++ Concurrent Programs

CS4560 - Parallel and Concurrent Programming

February, 2026

Project description

In this project, you will construct SC execution from a C11 concurrent program. [1]

Background Information

C11 concurrency

C/C++ defines a relaxed memory concurrency model known as the C11 concurrency model [1]. C11 has various kinds of accesses that affect shared memory concurrency. To begin with, it provides plain or non-atomic load and store accesses. In addition, C11 also has atomic accesses of four kinds: load, store, atomic update (RMW) such as compare-and-swap and atomic increment, and memory fence. Each atomic access is attached with a memory order from – relaxed, acquire, release, acquire-release, and sequentially-consistent.

Sequential Consistency

An execution is sequentially consistent if it follows an interleaving execution. Following the axiomatic model, in a sequential consistent execution forbids any cycle consisting of program-order (po), read-from (rf), modification-order a.k.a coherence-order (mo or co), and from-read (fr) relation edges. More formally, axiom for SC consistency is: ($\mathsf{po} \cup \mathsf{rf} \cup \mathsf{mo} \cup \mathsf{fr}$) are acyclic.

Note that we consider atomic update as a single event)

Execution

An execution consists of a set of events resulting from shared memory accesses or fences and relations between these events. Further details are in [1, 2, 3].

c11tester

Given a C11 program the c11tester tool [2, 3] may execute the program and generate execution traces with events and relations as discussed above following the memory orders of the accesses.

Note: *feel free to use any other tool if you like.*

Projects

In this project, you will construct SC consistent executions only – that is forbid any ($\mathsf{po} \cup \mathsf{rf} \cup \mathsf{mo} \cup \mathsf{fr}$) cycle in the execution.

[1] The project description is subject to small changes and updates. Please get in touch with the TAs and the teachers if you have any questions.

Roadmap for the project:

The project involves the following steps:

- Set up the c11tester tool. It is available at `https://brightspace.tudelft.nl/d2l/le/content/680657/viewContent/4088449/View`.
- Understand the internals of the c11tester execution construction steps.
- Develop the algorithms for SC consistent execution construction.
- Implement the algorithm inside c11tester.
- Evaluate on the c11tester benchmarks*.

The above-mentioned steps assume the c11tester tool. You may use any other tool.

Note

- You may use c11tester inside the vagrant box.
- The evaluation on the 'cdschecker-benchmarks' suffice. You may generate a larger trace by changing the input.
- Some applications (e.g. firefox) require significantly more computation and memory. You may skip these.

Restriction Do not change the memory orders of the accesses to the SC memory order.

References

[1] Mark Batty, Scott Owens, Susmit Sarkar, Peter Sewell, and Tjark Weber. Mathematizing C++ concurrency. In *POPL'11*, pages 55–66. ACM, 2011.

[2] Weiyu Luo and Brian Demsky. *C11Tester: A Race Detector for C/C++ Atomics*, page 630–646. 2021.

[3] Weiyu Luo and Brian Demsky. C11tester artifact.