Efficient Checking of Thread Refinement

Daniel Poetzl
University of Oxford

Motivation

Compilers must guarantee observational refinement for optimized threads. An optimized thread T' is a refinement of the original thread T if for all possible thread T_1, …, T_n, the set of final states reachable by T' ‖ T_1 ‖ … ‖ T_n is a subset of the set of final states reachable by T ‖ T_1 ‖ … ‖ T_n. We assume the "SC for DRF" model, i.e. programs behave sequentially consistent (SC) if their SC executions are free of data races, and programs containing data races have undefined semantics.

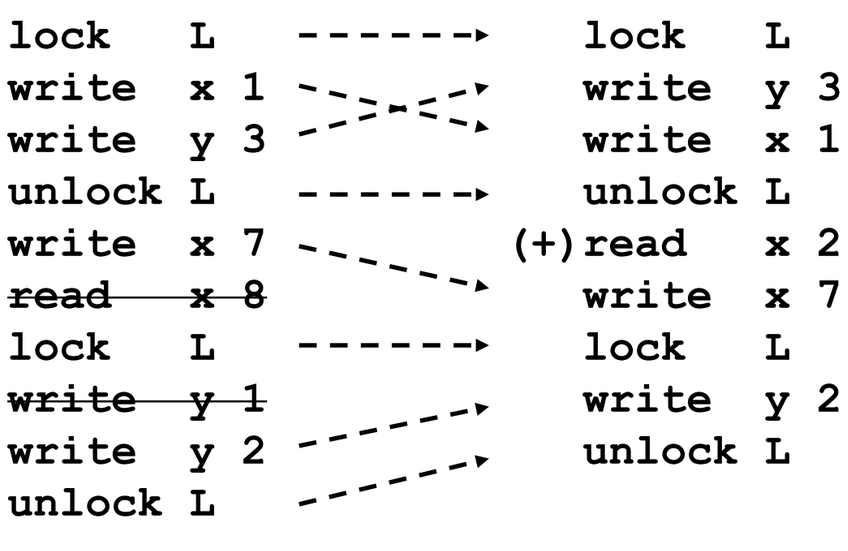
Our goal:

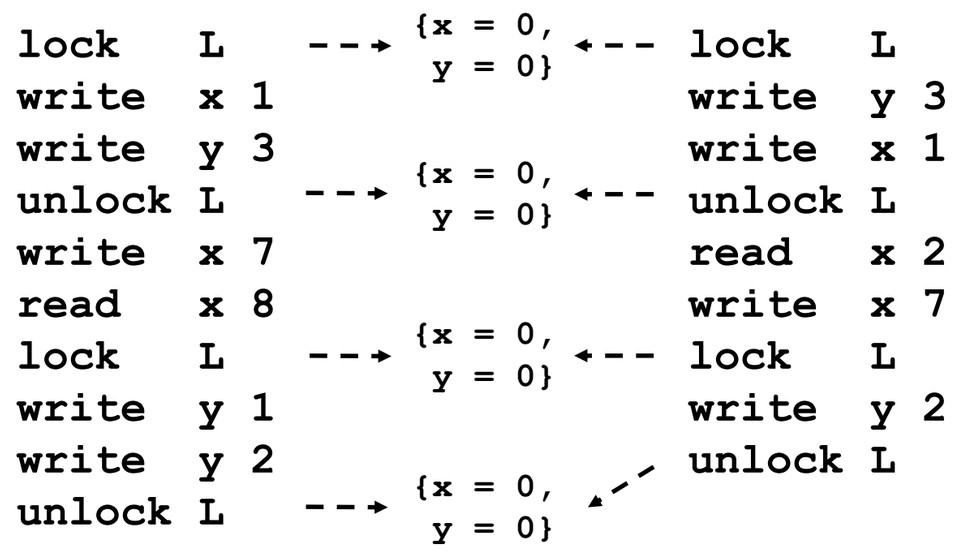
- *Formal criterion* for when a thread T' is a refinement of a thread T

Requirements:

- *Precision:* should validate all existing compiler optimizations and support potentially new compiler optimizations
- *Efficiency:* should support the implementation of efficient procedures for refinement checking

Specifying Refinement: Events vs. States





Current theories specify the allowed optimizations in terms of which reorderings, eliminations, and introductions of memory accesses are allowed on thread execution traces (e.g. [1], [2]). If all execution traces of the optimized thread T' can be transformed to an execution trace of the original thread T via a sequence of such allowed trace transformations, T' is considered a refinement of T.

Our specification approach:

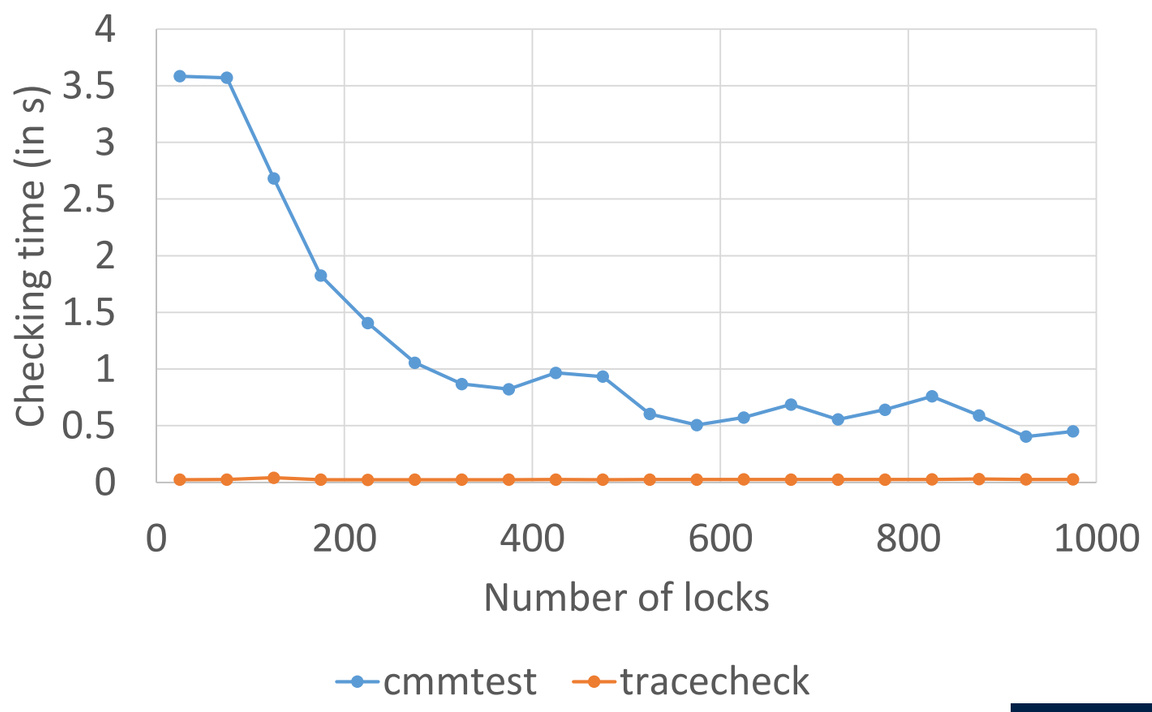
- Require that T' and T are in the same state at corresponding lock operations
- Require that the memory locations accessed by T' in a segment between two lock operations form a subset of the memory locations accessed by T in the corresponding segment

Application: Compiler Testing

Compiler testing method:

1. generate random C program (e.g. with csmith)
2. collect traces of optimized and unoptimized program
3. check traces for refinement
   - if trace of optimized program is not a refinement of trace of unoptimized program => compiler bug found

Morisset et al. [1] implemented this approach in the tool *cmmtest*, with an event-based trace checking method. Our tool *tracecheck* can check traces several orders of magnitude faster than cmmtest. The time taken by cmmtest varies with the number of locks in a trace, whereas tracecheck is insensitive to the number of locks.



References:
[1] R. Morisset, P. Pawan, F. Zappa Nardelli. Compiler Testing via a Theory of Sound Optimisations in the C11/C++11 Memory Model. PLDI '13.
[2] J. Sevcik. Safe Optimisations for Shared-Memory Concurrent Programs. PLDI '11.

UNIVERSITY OF OXFORD