SCFENCE: Automatic Inference of Memory Order Parameters to Obtain SC Behaviors under C/C++11

Peizhao Ou and Brian Demsky

University of California, Irvine

{peizhaoo,bdemsky}@uci.edu

Abstract

1. Introduction

2. Motivating Example

3. Technical

3.1 Inference Rules

Previous work takes advantage of model-checking approach to check whether a specific C/C++11 trace is sequentially consistent. By building up edges (*sb*, *rf* and *sc* by implication rules) between atomic operations, it judges whether the trace is SC by whether there exists a cycle in that graph. Besides, when it finds a non-SC trace, it has a sorting algorithm that generates an SC-like trace and exposes which reads-from edge that causes a cycle.

Under the C/C++11 memory model, inferring the ordering parameters to obtain SC behaviors is essentially a searching problem. In the absence of consume operations, memory order parameters for atomic operations can be only one of the following: *memory_order_relaxed*, *memory_order_release*, *memory_order_acquire*, *memory_order_acq_rel* and *memory_order_seq_cst*. By enumerating all possible memory order parameters, we can guarantee that we can find out all the possible inference of parameters that ensure SC behaviors for a specific test case. However, this naive approach obviously leads to an exponential searching space.

Actually, when we have a non-SC execution, we have some knowledge availabe reflecting where the problem may lie. Consider we start from the case where all memory order parameters are *memory_order_relaxed*. Whenever the model-checking approach finds out a cycle in a specific execution, we have to infer some stronger memory orders to eliminate the cycle. What causes the cycle to happen leads to the non-SC trace. We propose a search-based approach combined with cycle patterns and their fixes to reduce searching space.

In Figure 1 , we show a number of universal patterns that can exist in cycles. We explain how we should fix those cycle patterns respectivelys the following.

Circular *sb* $\cup$ *rf*: If a cycle is composed of edges which are the union of *sb* & *rf*, a universal fix is to make all but one of the atomic operation *happen-before* the next atomic operation in that cycle. It is worth noting that imposing *happens-before* to ajacent nodes is not limited to imposing release/acquire pairs to store/load operations involed in the cycle. Instead, any possible paths composed with the union of *sb* $\cup$ *rf* between the two nodes can be strengthened.

Old value read I: When we sort the trace into an SC-like ordering, we may encounter the case where a load reads from some old store and those three operations can be connected via at least one path composed on the union of *sb* $\cup$ *rf*. A universal fix for this pattern is to impose *happens-before* between the two stores and *happens-before* between the recent store and the load at the same time.

Old value read II: Similar to the previous pattern, we see a load read from an old store. However, the difference is that we do not have a union of *sb* $\cup$ *rf* between the two stores and between the recent store and the load at the same time. To fix this problem, we need to impose the following: 1) the old store *modification-order* before the recent store (weaker than *sc* or *hb*); 2) the recent store *sc* before the load or the *happens-before* the load.

Future value read: Unlike reading an old value in the trace, another possible pattern is reading a future value. To fix this pattern, we can do the following: 1) impose *happens-before* from the store to the load; 2) impose *sc* from the load to the store.

Figure 2 shows the core searching algorithm for all possible parameters.

4. Evaluation

5. Related Work

SC [1]

6. Conclusion

References

[1] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, 28(9):690–691, Sept. 1979.

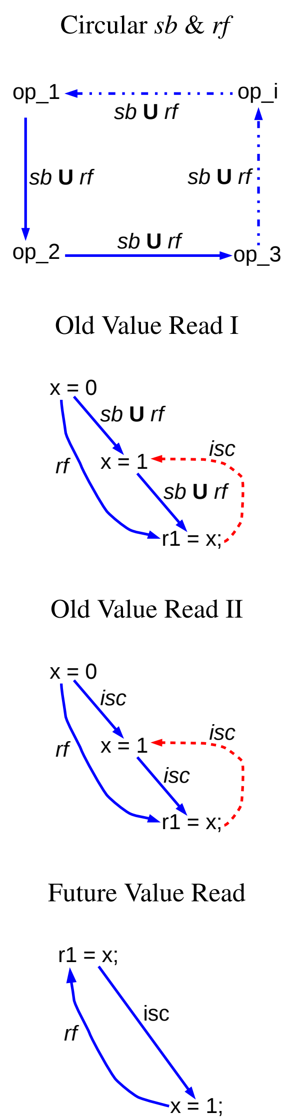

Figure 1. Cycle Patterns for Non-SC Behaviors

function INFERPARAMS
    candidates := {}
    candidate $c1$:= replace all wildcards with *relaxed*
    candidates += $c1$
    results := {}
    while candidates is not empty **do**
        Candidate c := candidates.pop()
        Model-check with c and yield a cycle l
        if l == NULL **then**
            results += c
        else
            STRENGTHENPARAM(l, c, candidates)
        end if
    end while
    return results
end function
procedure STRENGTHENPARAM(c, p, candidates)
    while $\exists$ a pattern p in cycle c **do**
        possible_fixes := strengthen c by pattern p
        candidates += possible_fixes
    end while
end procedure

Figure 2. Algorithm for Searching All Possible Parameters