

# 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 available 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 respectively 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 adjacent nodes is not limited to imposing release/acquire pairs to store/load operations involved 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:**

**Old value read II:**

**Future value read:**

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

## 4. Evaluation

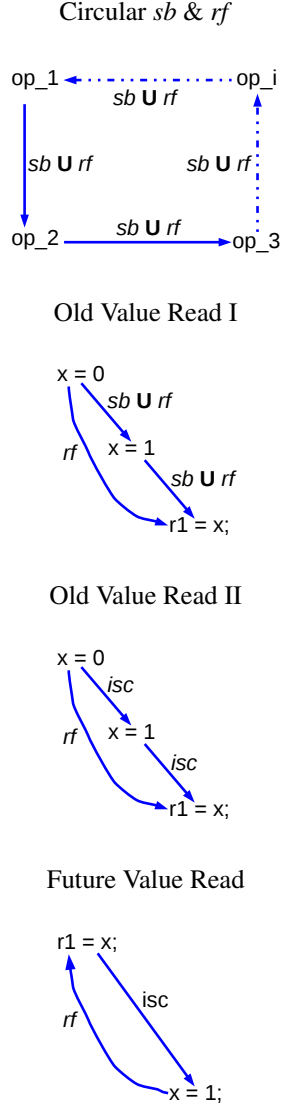
## 5. Related Work

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

```

1: function INFERPARAMS
2:   candidates := {}
3:   candidate c1 := replace all wildcards with relaxed
4:   candidates += c1
5:   results := {}
6:   while candidates is not empty do
7:     Candidate c := candidates.pop()
8:     Model-check with c and yield a cycle l
9:     if l == NULL then
10:      results += c
11:    else
12:      STRENGTHENPARAM(l, c, candidates)
13:    end if
14:  end while
15:  return results
16: end function
17: procedure STRENGTHENPARAM(c, p, candidates)
18:   while ∃ a pattern p in cycle c do
19:     possible_fixes := strengthen c by pattern p
20:     candidates += possible_fixes
21:   end while
22: end procedure

```

**Figure 2.** Algorithm for Searching All Possible Parameters