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 estabishing 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.

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.

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. In order to guarantee completeness, we propose a search-based approach combined with patterns to reduce searching space. In Figure 1 , we show a number of common patterns that can exist in cycles. We explain what the weakest orders we should impose on operations to eliminate the corresponding cycle as the following.

Text running away

Old value read I:

Old value read II:

Reading future value:

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

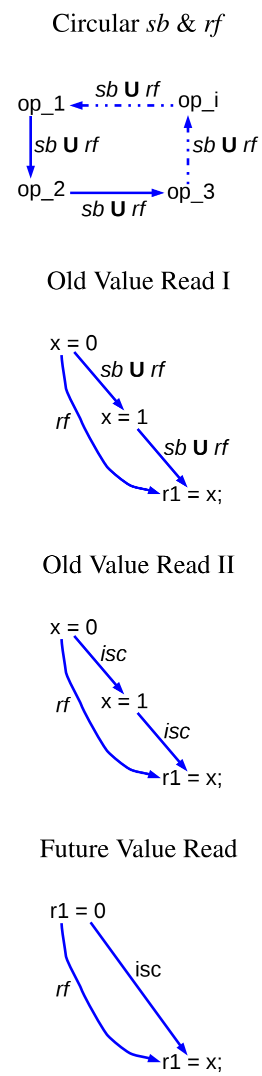

Figure 1. Cycle Patterns for Non-SC Behaviors

4. Evaluation

5. Related Work

SC [1]

```
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(cycle, candidate, candidates)
    if ∃ a pattern p in c then
        Candidate new := strengthen c by pattern p
        candidates += new
    else
        for all wildcard w in c do
            if w can be strengthened then
                Candidate new := strengthen w in c
                candidates += new
            end if
        end for
    end if
end procedure
```

Figure 2. Algorithm for Searching All Possible Parameters

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.