Brief Announcement: Obstruction-Free Step Complexity: Lock-free DCAS as an Example

Faith Ellen Fich[2⋆], Victor Luchangco[1], Mark Moir[1], and Nir Shavit[1]

[1] Sun Microsystems Laboratories
[2] University of Toronto

We propose *obstruction-free step complexity*, a new complexity measure for nonblocking algorithms. We believe that this measure provides a more pragmatic quantification of nonblocking algorithms than previous measures, providing better guidance for designers of practical nonblocking algorithms.

In our opinion, the main shortcoming of existing complexity measures for nonblocking algorithms is that they are targeted towards worst-case behavior in worst-case scenarios, and say little about behavior in more common cases. This is true for the *sensitivity* measure of Attiya and Dagan [1], and the *d-local step complexity* of Afek et al. [2]. These measures are directed at evaluating the behavior of algorithms under contention, i.e., when concurrent operations actively interfere with each other's progress. However, in practice, a well-designed system manages contention so that it does not impact performance too greatly. Thus, these previous measures do not evaluate the behaviour that is likely to be observed.

For *any* nonblocking algorithm, be it wait-free, lock-free or obstruction-free, the *obstruction-free step complexity* of the algorithm is the maximum over all reachable states of the number of steps required for any operation to complete if the process executing the operation runs alone from that state.[3]

Obstruction-free step complexity is targeted towards a more pragmatic evaluation of the performance of nonblocking algorithms in real-world applications, and is based on the assumption, evidenced by numerous technical papers, that performance in uncontended cases is more important in many practical settings than worst-case performance under contention. Moreover, by far the most common approach to dealing with contention is backoff, in which contended cases are essentially turned into uncontended ones by delaying contending operations; other contention management approaches behave similarly. This suggests that even for applications and systems in which contention is significant, it is important to design algorithms that perform well while there is no contention. In particular, if operations complete quickly as soon as contention subsides, then contention management techniques such as backoff will be more effective, and

⋆ The research of Faith Ellen Fich was financially supported by the Natural Sciences and Engineering Research Council of Canada and the Scalable Synchronization Research Group of Sun Microsystems, Inc.

[3] Algorithms that can take an unbounded number of steps even in the absence of contention have unbounded obstruction-free step complexity.

will produce progress more quickly. This is the motivation behind our proposed complexity measure.

Because the obstruction-free step complexity of an algorithm bounds the number of steps that a process takes when running alone from an *arbitrary* reachable state, even one in which other processes may be in the midst of executing the algorithm, it evaluates how amenable an algorithm is to contention management strategies such as backoff.

It is important to distinguish obstruction-free step complexity from an alternative measure, which we call *completely contention-free step complexity*: the maximum number of steps any process takes if it runs alone from a "quiescent" state (i.e., a state in which no process is in the midst of executing the algorithm). This measure provides no insight into how amenable the algorithm is to contention management. In particular, a lock-based algorithm may have very low completely contention-free complexity, but a process that stops while holding a lock can prevent all the other processes from making progress indefinitely, and no contention management can help in this case.

Taking an approach similar to that of [1, 2], we use obstruction-free step complexity to evaluate nonblocking implementations of multilocation synchronization operations from unary ones. One difficulty in designing such algorithms is deciding what an operation should do when it discovers that another operation is already accessing a location it wants to access. One option is to "abort" the competing operation, causing the aborted operation to retry; another is to "help" that operation complete. Aborting operations excessively can compromise lock-freedom, as multiple operations can repeatedly abort each other. On the other hand, if an operation always helps the competing operation, long chains can form such that one operation must recursively help all the operations in the chain, even if none of them are actively interfering.

In particular, we consider implementations of double-compare-and-swap from compare-and-swap, and present the first algorithm with *constant* obstruction-free step complexity. All previous lock-free algorithms use the recursive-helping technique and therefore do not have constant obstruction-free step complexity. To achieve constant obstruction-free step complexity, our algorithm introduces a novel approach, which carefully helps only enough to ensure lock-freedom, while avoiding long helping chains. Note that our result does not contradict the separation result of [1], which considered the worst-case step complexity of *wait-free* algorithms over *all* executions: our algorithm is only lock-free, and the obstruction-free step complexity measure focuses only on the uncontended case.

References

1. Attiya, H., Dagan, E.: Improved implementations of binary universal operations. J. ACM **48** (2001) 1013–1037
2. Afek, Y., Merritt, M., Taubenfeld, G., Touitou, D.: Disentangling multi-object oeprations. In: Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing. (1997) 111–120