

Brief Announcement: Dynamic-Sized Lock-Free Data Structures*

Maurice Herlihy
Computer Science Department
Brown University, Box 1910
Providence, RI 02912

Victor Luchangco Paul Martin Mark Moir
Sun Microsystem Laboratories
1 Network Drive, UBUR02-311
Burlington, MA 01803

Almost all previous dynamic-sized lock-free data structures are either unable to free memory to the memory allocator when it is no longer required, or require special system or hardware support. In the only exception we are aware of, a single thread failure can prevent further memory reclamation (see full paper for reference).

We recently posed a problem — the Repeat Offenders Problem (ROP) — and presented one solution and its correctness proof [3]. Solutions to this problem can be used to design dynamic-sized lock-free implementations of shared data structures that overcome all of the problems mentioned above.

In the full paper [2], we present two results. The first is a general methodology, based on any ROP solution, for transforming dynamic-sized lock-free data structure implementations that depend on garbage collection (GC) for memory management into equivalent ones that do not require GC. This methodology is based on reference counts, and therefore entails space and time overhead required to maintain reference counts. (This methodology improves on the one presented in [1] by removing the dependence on double compare-and-swap (DCAS).)

ROP can also be applied directly to achieve more efficient (both in space and time) implementations of dynamic-sized lock-free data structures. In the full paper, we give an example to demonstrate this approach. Specifically, we show how to modify the widely-used lock-free FIFO queue implementation of Michael and Scott so that it can free memory to the memory allocator when it is no longer required. We also present the results of performance experiments

we have conducted on two multiprocessor machines; these results demonstrate that the overhead of this additional functionality is negligible under low contention, and low in all cases.

Maged Michael has concurrently and independently developed an approach that is very similar to ours [4]. Michael's approach has the advantage that it does not require compare-and-swap (CAS), as ours does. However, most practical lock-free dynamic data structure implementations depend on strong synchronization primitives anyway, so this advantage will be irrelevant in most cases. Furthermore, our ROP algorithm makes stronger guarantees about freeing memory than Michael's does; we suspect that our stronger property cannot be achieved efficiently using only reads and writes. See [3] for a discussion of the differences between our approaches.

References

- [1] D. Detlefs, P. Martin, M. Moir, and G. Steele. Lock-free reference counting. In *Proceedings of the 20th Annual ACM Symposium on Principles of Distributed Computing*, pages 190–199, 2001.
- [2] M. Herlihy, V. Luchangco, P. Martin, and M. Moir. Dynamic-sized lock-free data structures. Technical Report TR-2002-110, Sun Microsystems Laboratories, 2002.
- [3] M. Herlihy, V. Luchangco, and M. Moir. The repeat offender problem: A mechanism for supporting lock-free dynamic-sized data structures. Technical Report TR-2002-112, Sun Microsystems Laboratories, 2002.
- [4] M. Michael. Safe memory reclamation for dynamic lock-free objects using atomic reads and writes. In *Proceedings of the 21st Annual ACM Symposium on the Principles of Distributed Computing*, 2002.

*Contact email: Mark.Moir@sun.com