Science of Computer Programming 102 (2015) 57–75

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Mutual exclusion by four shared bits with not more than quadratic complexity

Wim H. Hesselink

Dept. of Computing Science, University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands

A R T I C L E   I N F O

Article history:
Received 19 February 2014
Received in revised form 4 November 2014
Accepted 5 January 2015
Available online 14 January 2015

Keywords:
Concurrent algorithm
Mutual exclusion
Fault tolerance
Time complexity
Verification

A B S T R A C T

For years, the mutual exclusion algorithm of Lycklama and Hadzilacos (1991) [21] was the optimal mutual exclusion algorithm with the first-come-first-served property, with a minimal number of (non-atomic) communication variables (5 bits per thread). Recently, Aravind published an improvement of it, which uses 4 bits per thread and has simplified waiting conditions. This algorithm is extended here with fault tolerance, and it is verified by assertional methods, using the proof assistant PVS. Progress is proved by means of UNITY logic. The paper proposes a new measure of concurrent time complexity, and proves that the concurrent complexity for throughput of the present algorithm is not more than quadratic in the number of threads.

1. Introduction

Due to the advent of multiprocessors and multicore architectures, the practical relevance of concurrent algorithms is increasing dramatically. This raises the interest in the correctness of them because, as is well known, such algorithms can unexpectedly misbehave due to subtle bugs or race conditions.

Testing and model checking are important methods to find errors, but, for concurrent algorithms, they are often not able to ensure correctness. Formal verification of a concurrent algorithm usually requires many case distinctions. Therefore, even carefully drafted man-made proofs are hardly convincing. In recent years, the advance of mechanical theorem provers like ACL2, Coq, HOL, Isabelle, PVS has made it possible to prove concurrent algorithms exhaustively, and in such a way that the proof script can be inspected and replayed to verify the correctness claims of the verifier. Effective use of a prover for these purposes requires a good understanding of the methods of concurrency verification.

The present paper illustrates these possibilities by presenting a computer assisted verification of a recent improvement by Alex Aravind [5] of the mutual exclusion algorithm of Lycklama and Hadzilacos [21]. For many years, the algorithm of [21] was the “best possible one”, in the sense that it provides mutual exclusion with the first-come first-served property by means of a minimum number of communication variables: 5 bits per thread which need not be atomic.

Aravind’s improvement reduces the number of communication bits to 4, simplifies the waiting conditions, and retains the other properties of the algorithm. Moreover, the algorithm is fault tolerant in the sense that a thread may fail at any time. A failed thread goes immediately to its noncritical section, but its communication variables may get arbitrary values. After some period of time it resets them, and it may try and reenter the protocol.

The algorithm and its proof work under the assumption of sequentially consistent memory. If one wants to apply it on current hardware, memory fences are needed to prevent the hardware from reordering loads before stores [7, Section 4].

E-mail address: w.h.hesselink@rug.nl.

http://dx.doi.org/10.1016/j.scico.2015.01.001
0167-6423/© 2015 Elsevier B.V. All rights reserved.

Simplified versions of the algorithms of Lycklama–Hadzilacos and Aravind were tested with appropriate memory fences in [7, Section 17].

In the present paper, the algorithm is verified completely, including nonatomicity and fault tolerance. The verification is done entirely with assertional methods, i.e., in terms of states, the next state relation, and the forward steps done under weak fairness. The safety properties are verified by invariants and history variables. Progress of the algorithm is verified with a bounded version of UNITY logic [9,12]. In this way, we obtain explicit bounds for throughput and individual delay.

Contributions:

- Addition of fault tolerance and fault recovery.
- Verification of safety and progress.
- Explicit bounds for throughput and individual delay in terms of rounds.

1.1. Overview

Section 1.2 presents the mutual exclusion problem (MX) and the first-come-first-served property (FCFS). Section 1.3 introduces the solution of Lycklama and Hadzilacos, as improved by Aravind. In Section 1.4, we explain our time complexity for concurrent algorithms. Section 1.5 sketches the approach to verification.

Section 2 presents the algorithm with 4 shared bits. It first explains how Lycklama and Hadzilacos [21] have separated the concerns for MX and FCFS. It then presents Aravind's algorithm along these lines.

Section 3 presents the formal model for concurrent algorithms with shared memory, introduces UNITY and our bounded version of it, and then discusses atomicity and nonatomic shared variables.

In Section 4, we decorate the algorithm as presented in Section 2 with history variables and environment steps in such a way that it forms a blueprint for a transition system amenable to formal verification. We prove that this system satisfies MX and FCFS, and absence of immediate deadlock.

Progress is treated in Section 5. Here, the fault tolerance complicates matters considerably, because a frequently failing thread can obstruct the progress of nonfailing threads completely. We conclude in Section 6.

1.2. Mutual exclusion

The problem that concurrent processes may need exclusive access to some shared resource was first proposed and solved in [10]. The problem came to be known as mutual exclusion in [11]. Numerous solutions to this problem have been proposed, e.g., see the surveys [2,7,25,27].

An early and elegant solution is Lamport's bakery algorithm [18], which has three additional properties: it has the first-come-first-served property (FCFS), the shared variables used need not be atomic, and it is fault tolerant in a certain sense. The second point means that the algorithm does not assume mutual exclusion on read and write operations on shared variables. On the other hand, these shared variables hold integer values that can become arbitrarily large.

In the past, devising busy-waiting solutions to the mutual exclusion problem was primarily an academic exercise, because busy waiting is inefficient on a single processor. The advent of multiprocessors and multicore architectures, however, has spurred renewed interest in such algorithms [3, p. 133].

Similarly, algorithms for nonatomic shared variables are becoming practically relevant, because several recent systems such as smart-phones, multi-mode handsets, multiprocessor systems, network processors, graphics chips, and other high performance electronic devices use multiport memories, and such memories allow nonatomic accesses through multiple ports [17,26,28].

Mutual exclusion without FCFS does not require much shared memory. Indeed, the algorithm of Burns–Lamport [8,19] establishes mutual exclusion with only one nonatomic shared Boolean variable per thread.

Lamport's bakery algorithm for N threads gives mutual exclusion with FCFS, using only N nonatomic shared integer variables, but these variables cannot be bounded. The algorithm of [6] also gives mutual exclusion with FCFS. It uses N nonatomic variables with values $\leq N$, and $3N + 2$ nonatomic shared bits.

In terms of shared-space complexity, however, the best mutual exclusion algorithm with the FCFS property, that is known to us, is the one of Lycklama and Hadzilacos [21], or rather the recent simplification by Aravind [5].

1.3. Mutual exclusion by Lycklama–Hadzilacos–Aravind

The mutual exclusion algorithm of Lycklama and Hadzilacos [21] establishes mutual exclusion with the FCFS property for an arbitrary number of threads. Per thread, it uses five shared Boolean variables, which need not be atomic.

The presentation of the algorithm splits it in two parts: an inner algorithm to establish mutual exclusion and an outer algorithm to guarantee the FCFS property. The inner algorithm is the mutual exclusion algorithm of Burns [8] and Lamport [19], which uses one shared bit per thread. The outer algorithm uses four shared bits per thread. The combined algorithm thus uses five bits per thread.