CCIT Report #866 August 2014

On Correctness of Data Structures under Reads-Write Concurrency [*]

Kfir Lev-Ari
EE Department
Technion,
Haifa, Israel
kfirla@campus.technion.ac.il

Gregory Chockler
CS Department
Royal Holloway,
University of London, UK
gregory.chockler@rhul.ac.uk

Idit Keidar
EE Department
Technion,
Haifa, Israel
idish@ee.technion.ac.il

Abstract

We study the correctness of shared data structures under reads-write concurrency. A popular approach to ensuring correctness of read-only operations in the presence of concurrent update, is read-set validation, which checks that all read variables have not changed since they were first read. In practice, this approach is often too conservative, which adversely affects performance. In this paper, we introduce a new framework for reasoning about correctness of data structures under reads-write concurrency, which replaces validation of the entire read-set with more general criteria. Namely, instead of verifying that all read shared variables still hold the values read from them, we verify abstract conditions over the shared variables, which we call *base conditions*. We show that reading values that satisfy some base condition at every point in time implies correctness of read-only operations executing in parallel with updates. Somewhat surprisingly, the resulting correctness guarantee is not equivalent to linearizability, rather, it can express a range of conditions. Here we focus on two new criteria: *validity* and *regularity*. Roughly speaking, the former requires that a read-only operation never reaches a state unreachable in a sequential execution; the latter generalizes Lamport's notion of regularity for arbitrary data structures, and is weaker than linearizability. We further extend our framework to capture also linearizability and sequential consistency. We illustrate how our framework can be applied for reasoning about correctness of a variety of implementations of data structures such as linked lists.

1 Introduction

Motivation Concurrency is an essential aspect of computing nowadays. As part of the paradigm shift towards concurrency, we face a vast amount of legacy sequential code that needs to be parallelized. A key challenge for parallelization is verifying the correctness of the new or transformed code. There is a fundamental tradeoff between generality and performance in state-of-the-art approaches to correct parallelization. General purpose methodologies, such as transactional memory [14, 25] and coarse-grained locking, which do not take into account the inner workings of a specific data structure, are out-performed by hand-tailored fine-grained solutions [21]. Yet the latter are notoriously difficult to develop and verify. In this work, we take a step towards mitigating this tradeoff.

It has been observed by many that correctly implementing concurrent modifications of a data structure is extremely hard, and moreover, contention among writers can severely hamper performance [23]. It is therefore not surprising that many approaches do not allow write-write concurrency; these include the *read-copy-update (RCU)* approach [20], flat-combining [13], coarse-grained readers-writer locking [9], and pessimistic software lock-elision [1]. It has been shown that such methodologies can perform better than ones that allow write-write concurrency, both when there are very few updates relative to queries [20] and when writes contend heavily [13]. We focus here on solutions that allow only read-read and read-write concurrency.

[*]This work was partially supported by the Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI), by the Israeli Ministry of Science, by a Royal Society International Exchanges Grant, and by the Randy L. and Melvin R. Berlin Fellowship in the Cyber Security Research Program.