# Lock-Free Linked Lists

An Analysis of the Fomitchev-Ruppert Implementation

ANDREW GELTZ, University of Central Florida, USA

The linked list data structure is an essential building block for many other complex data structures including many lock-free hash table and skip-list implementations. In its own right, the linked list is useful data structure because it maintains order that is common in array structures while still having the ability to be dynamic with size and insertions. The implementation covered in this paper was proposed by Mikhail Fomitchev and Eric Ruppert from York University.

CCS Concepts: • **Theory of computation** → **Data structures design and analysis**; • **Computer systems organization** → **Multiple instruction, single data**; • **Software and its engineering** → *Concurrency control*;

Additional Key Words and Phrases: Linked-List, Concurrent, Parallel, Efficiency

**ACM Reference Format:**
Andrew Geltz. 2018. Lock-Free Linked Lists: An Analysis of the Fomitchev-Ruppert Implementation. 1, 1 (March 2018), 3 pages.
https://doi.org/0000001.0000001

## 1 INTRODUCTION

This paper is intended to catalog the implementation and performance of a lock-free linked list as suggested by Fomitchev and Ruppert. An analysis of the performance impact of the changes over several different linked list algorithms is provided as a means of comparison. The suggested implementation has an added layer of complexity with the inclusion of a "backlink" [Fomitchev and Ruppert 2004, pg 52] in order to eliminate the need for re-traversal when failed Compare and Swap (CAS) calls are made. Being able to recover from these situations should increase the performance in environments that experience higher contention on nodes in the list.

## 2 EXPLANATION OF ALGORITHMS

### 2.1 Linked List Requirements

For the purposes of this paper, a linked list algorithm is a list of nodes that have some information and a one-way reference to the next node in the list. To ensure the same functionality for all algorithms the nodes will be organized using a key value and sorted in ascending order. The algorithms must support the following functions:

- Add - Inserts a node into the list in its sorted position, has no effect if a node with that key is already in the list
- Remove - Deletes a node from the list, has no effect if specified node is not in the list
- Contains - Returns a boolean value that indicates whether the specified node is in the list

Author's address: Andrew Geltz, University of Central Florida, 4000 Central Florida Blvd, Orlando, FL, 32816, USA, andrew.geltz@knights.ucf.edu.

### 2.2 Single Thread Version

#### *2.2.1 Implementation.*

### 2.3 Locked Based Versions

#### *2.3.1 Implementation.*

### 2.4 Harris Multi Thread Version

The Harris implementation of the linked list data structure [Harris 2001], is similar to the algorithm discussed in class. It uses CAS to ensure the list is updated atomically.

#### *2.4.1 Implementation.*

### 2.5 Fomitchev-Ruppert Version

The Fomitchev-Ruppert implementation [Fomitchev and Ruppert 2004] is similar to the Harris implementation in structure but differs in a few parts. In order to implement a "backlink" to the previous node and prevent issues that would occur from using that link naively, the data structure for the node had to be manipulated. The backlink is simply a pointer to the successor of the current node but it is only linked during the removal of the current node to reduce the overhead of other operations as well as limit the number of memory locations that need to manipulated when changing the structure of the list, such as adding or removing nodes.

The main change to the node structure was the addition of the backlink pointer. Before a node is marked for deletion, the backlink is atomically updated to point to the node's predecessor. If another process' CAS fails because the node is marked, it can follow the backlink to predecessor and attempt to recover without having to re-traverse the linked list. The implementation of backlinks does not guarantee performance as long chains of backlinks can occur when many removes are done near one another and that decreases the performance of other operations. Fomitchev and Ruppert suggest using a second flag bit in the node's next pointer. This extra flag bit would be used to indicate that the next node was in the process of being removed. It acts as a warning system for other remove operations to not remove the predecessor to the node being removed to prevent backlink chains.

#### *2.5.1 Implementation.*

#### *2.5.2 Correctness.*

## 3 PERFORMANCE EVALUATION

### 3.1 Testing Procedures

### 3.2 Testing Results

### 3.3 Analysis of Results

## ACKNOWLEDGMENTS

The author would like to thank Dr. Damian Dechev of University of Central Florida. His instruction on Parallel Processing was invaluable in the research and implementation of this data structure.

## REFERENCES

Mikhail Fomitchev and Eric Ruppert. 2004. Lock-free Linked Lists and Skip Lists. In *Proceedings of the Twenty-third Annual ACM Symposium on Principles of Distributed Computing (PODC '04)*. ACM, New York, NY, USA, 50–59. https://doi.org/10.1145/1011767.1011776

Timothy L. Harris. 2001. A Pragmatic Implementation of Non-blocking Linked-Lists. In *Proceedings of the 15th International Conference on Distributed Computing (DISC '01)*. Springer-Verlag, London, UK, UK, 300–314. http://dl.acm.org/citation.cfm?id=645958.676105