Rheinland-Pfälzische
Technische Universität
Kaiserslautern
Landau

Verified Functional Data Structures: Priority Queues in Liquid Haskell

Master's Thesis

by

MohammadMehran Shahidi

August 7, 2025

University of Kaiserslautern-Landau
Department of Computer Science
67663 Kaiserslautern
Germany

Examiner: Prof. Dr. Ralf Hinze
Michael Youssef, M.Sc.

Declaration of Independent Work

I hereby declare that I have written the work I am submitting, titled "Verified Functional Data Structures: Priority Queues in Liquid Haskell", independently. I have fully disclosed all sources and aids used, and I have clearly marked all parts of the work — including tables and figures — that are taken from other works or the internet, whether quoted directly or paraphrased, as borrowed content, indicating the source.

Kaiserslautern, den 7.8.2025

MohammadMehran Shahidi

Abstract

Formal program verification is a powerful approach to ensuring the correctness of software systems. However, traditional verification methods are often tedious, requiring significant manual effort and specialized tools or languages [RKJ08].

This thesis explores `LiquidHaskell`, a refinement type system for Haskell that integrates SMT (Satisfiability Modulo Theories) solvers to enable automated verification of program properties [Vaz+18]. We demonstrate how Liquid Haskell can be used to verify correctness of priority queue implementations in Haskell. By combining type specifications with Haskell's expressive language features, we show that `LiquidHaskell` allows for concise and automated verification with minimal annotation overhead.

Contents

1. Introduction **1**
1.1. Motivation . . . 1
1.2. Problem Statement . . . 1
1.3. Goals and Contributions . . . 2
1.4. Structure of the Thesis . . . 2

2. Background and Related Work **3**
2.1. Functional Data Structures . . . 3
2.2. Priority Queues . . . 3
2.3. Program Verification Techniques . . . 3
2.4. Related Work . . . 3

3. Priority Queue Implementations **5**
3.1. Specification of Priority Queue Properties . . . 5
3.2. Leftist Heap Implementation . . . 5
3.3. Binary Heap Implementation . . . 5
3.4. Skew/Binomial/Pairing Heap (optional) . . . 5
3.5. Comparison of Implementations . . . 5

4. Liquid Haskell **7**
4.1. Overview of Liquid Haskell . . . 7
4.2. Specification Language . . . 7
4.3. Verification Workflow . . . 7
4.4. Strengths and Limitations . . . 7
4.5. Related Work . . . 7

5. Verification in Liquid Haskell **9**
5.1. Encoding Invariants . . . 9
5.2. Use of Measures and Predicates . . . 9
5.3. Example Proofs . . . 9
5.4. Dealing with Termination and Recursion . . . 9
5.5. Challenges and Workarounds . . . 9

List of Figures **11**

A. My Code **13**

B. My Ideas **15**

Bibliography **17**

1. Introduction

1.1. Motivation

Data structures are fundamental in computer science, providing efficient ways to organize, store, and manipulate data. However, correctness of these structures is vital, especially in safety-critical systems such as aviation, finance, or healthcare, where software bugs can lead to catastrophic consequences. Traditional testing techniques are often insufficient to cover all execution paths or edge cases, particularly for complex data invariants.

Priority queues are one such data structure used widely in scheduling, pathfinding algorithms (e.g., Dijkstra's), and operating systems. Their correctness is essential to ensure minimal elements are accessed as expected, and operations like insertion, deletion, and merging preserve the heap property [Oka98].

Formal verification provides a promising avenue for ensuring correctness, but mainstream adoption is hindered by the complexity of existing tools. This thesis explores an approach that brings verification closer to the developer: integrating verification directly into the Haskell programming language via `LiquidHaskell`. By embedding logical specifications into types, developers can catch invariant violations at compile time—without leaving their programming environment [RKJ08].

1.2. Problem Statement

Program verification is the process of proving that a program adheres to its intended specifications. For example, verifying that the result of a `splitMin` operation on a priority queue indeed removes the minimum element and preserves the heap invariant.

While powerful tools like Coq, Agda, and Dafny enable formal proofs, they often require switching to a new language or proof assistant environment, a deep understanding of dependent types or interactive theorem proving, and significant annotation and proof overhead. These barriers limit adoption in day-to-day software development.

`LiquidHaskell` offers an alternative: a lightweight refinement type system that integrates seamlessly into Haskell [Vaz+18]. It leverages SMT solvers to check properties like invariants, preconditions, and postconditions automatically, thus reducing manual proof effort.

1.3. Goals and Contributions

This thesis aims to bridge the gap between practical programming and formal verification by demonstrating how Liquid Haskell can be used to verify the correctness of priority queue implementations.

The key contributions are as follows. First, we implement multiple functional priority queue variants (e.g., Leftist Heap, Binary Heap) in Haskell. Second, we encode structural and behavioral invariants using refinement types in Liquid Haskell. Third, we demonstrate verification of correctness properties directly in Haskell with minimal annotation. Finally, we evaluate the ease, limitations, and effort required in this approach compared to traditional theorem provers.

This integrated approach allows both implementation and verification to happen in the same language and tooling ecosystem, making verified software development more accessible.

1.4. Structure of the Thesis

The rest of this thesis is structured as follows:

- **Chapter 2** provides background on functional data structures, priority queues, and program verification techniques, along with related work.
- **Chapter 3** describes the design and implementation of different priority queue variants in Haskell.
- **Chapter 4** introduces Liquid Haskell, its syntax, verification pipeline, and its strengths and limitations.
- **Chapter 5** demonstrates the verification of priority queue operations using Liquid Haskell, including encoding invariants, use of refinement types, and example proofs.
- **Appendices** contain the complete verified code and additional implementation insights.

2. Background and Related Work

2.1. Functional Data Structures

Overview of functional programming principles and persistent data structures.

2.2. Priority Queues

Priority queues are multiset with an associated priority for each element, allowing efficient retrieval of the highest (or lowest) priority element. To avoid confusion with the term FIFO queues, we will refer to them as "priority heaps" throughout this thesis. Definition, applications, and various implementations (e.g., heaps, binomial queues).

2.3. Program Verification Techniques

Overview of Hoare logic, model checking, interactive theorem proving, etc.

2.4. Related Work

Comparison with Coq, Agda, Dafny, or other tools verifying similar structures.

3. Priority Queue Implementations

3.1. Specification of Priority Queue Properties

Priority queues or heaps are data structures that provide efficient access to the high priority element (often minimum element). that support the following operations:

- **Insert**: Add an element with a given priority.
- **Find-Min**: Retrieve the element with the highest priority (lowest value).
- **Delete-Min**: Remove the element with the highest priority.
- **Merge**: Combine two priority queues into one.

For the sake of simplicity, we will focus on the **min-heap** variant, where the element with the lowest value has the highest priority.

3.2. Leftist Heap Implementation

Haskell code, explanation, and verification.

3.3. Binary Heap Implementation

Code structure, key invariants, and proof techniques.

3.4. Skew/Binomial/Pairing Heap (optional)

More advanced structure and proof techniques (if time allows).

3.5. Comparison of Implementations

Performance, expressiveness, and verification effort.

4. Liquid Haskell

4.1. Overview of Liquid Haskell

Setup, syntax, and capabilities.

4.2. Specification Language

How to write and interpret refinement types.

4.3. Verification Workflow

From writing Haskell code to getting verified guarantees via Liquid Haskell.

4.4. Strengths and Limitations

Where it excels and where it struggles (e.g., termination proofs, higher-order functions).

4.5. Related Work

Comparison with Coq, Agda, Dafny, or other tools verifying similar structures.

5. Verification in Liquid Haskell

5.1. Encoding Invariants

How structural and behavioral invariants are expressed in refinement types.

5.2. Use of Measures and Predicates

Defining custom properties over data.

5.3. Example Proofs

Walkthroughs of insert and deleteMin correctness.

5.4. Dealing with Termination and Recursion

How Liquid Haskell checks termination.

5.5. Challenges and Workarounds

What was hard to prove and how you solved it.

List of Figures

A. My Code

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

B. My Ideas

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Bibliography

[Oka98] Chris Okasaki. *Purely Functional Data Structures*. Cambridge: Cambridge University Press, 1998. ISBN: 978-0-521-63124-2. DOI: 10.1017/CBO9780511530104. URL: https://www.cambridge.org/core/books/purely-functional-data-structures/0409255DA1B48FA731859AC72E34D49 (visited on 04/06/2025).

[RKJ08] Patrick M. Rondon, Ming Kawaguci, and Ranjit Jhala. "Liquid Types". In: *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '08. New York, NY, USA: Association for Computing Machinery, June 7, 2008, pp. 159–169. ISBN: 978-1-59593-860-2. DOI: 10.1145/1375581.1375602. URL: https://dl.acm.org/doi/10.1145/1375581.1375602 (visited on 03/03/2025).

[Vaz+18] Niki Vazou, Anish Tondwalkar, Vikraman Choudhury, Ryan G. Scott, Ryan R. Newton, Philip Wadler, and Ranjit Jhala. "Refinement Reflection: Complete Verification with SMT". In: *Proceedings of the ACM on Programming Languages* 2 (POPL Jan. 2018), pp. 1–31. ISSN: 2475-1421. DOI: 10.1145/3158141. URL: https://dl.acm.org/doi/10.1145/3158141 (visited on 12/15/2024).