Research Proposal

Formal Verification and Algorithmic Analysis of Braun Trees in Coq

Filiuta Alexandru s4541316

March 24, 2024

Abstract

This proposal details a rigorous investigation into the formal verification and analysis of Braun Trees using the Coq proof assistant. Braun Trees are vital in computing science, offering efficient solutions for various computational problems, but they have yet to be thoroughly verified in Coq. This research aims to bridge this gap by leveraging Coq's interactive development environment to comprehensively analyze the correctness of Braun Trees alongside 4 fundamental algorithms proposed by Chris Okasaki, Tobias Nipkow, and Thomas Sewell. This endeavor not only aims to strengthen the trust in software systems by enhancing their reliability through formal proofs but also seeks to enrich the body of knowledge around the formal verification process itself.

1 Introduction

In computer science, the complexity of software systems calls for robust verification methods to ensure their reliability and efficiency. Formal verification of algorithms and data structures offers a mathematical guarantee of their correctness. This proposal delves into the formal verification and algorithmic analysis of Braun Trees within the Coq proof assistant, a significant step towards establishing formal correctness proofs for these fundamental structures. Coq is a tool that aids in creating mathematical proofs and checking algorithms. Its foundation allows for precise definitions and proofs.

Braun trees, a type of binary tree designed for efficient implementation of extensible arrays and priority queues, were first introduced by W. Braun and M. Rem in 1983 as an innovative approach to ensure logarithmic time complexity for all array operations[1]. Braun trees are a form of balanced trees (see example in Fig 1). The balancedness is maintained through the following condition: each tree node ensures that the size of its left subtree is either equal to or one more than the size of its right subtree. This balance criterion facilitates operations such as adding or removing elements, element selection, and replacement within logarithmic time bounds. Despite their utility, the formal verification of Braun Trees in Coq remains unexplored territory[2]. This research aims to bridge this gap by employing Coq's powerful formal verification capabilities to rigorously analyze the correctness and efficiency of Braun Trees and Okasaki's associated algorithms.

To carry out this formal verification proposal, we will use techniques both from theoretical computer science and the practice of software engineering. From the plethora of fields of study that are applicable to this proposal, the most notable are: program correctness, pseudocode, logic, functional programming and algorithms and data structures. This document details our efforts to highlight the significance of formal verification in modern software development and advance the domain of program correctness through Coq.

2 Motivation

The proposed research holds significance across various domains, offering substantial benefits to research work, programming industry, and society at large. This research team is highly motivated by the need to

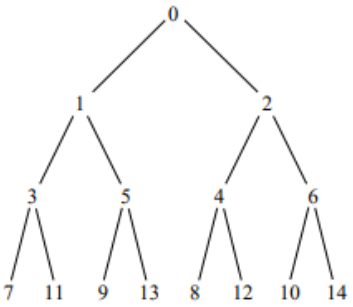

Figure 1: A Braun tree of size 15, with each node labelled by its index

progress formal verification techniques and by the project's potential for advancing the field of program correctness. At its core, formal verification in Coq promises to elevate the reliability and trustworthiness of software systems, mitigating the risks associated with programming errors. As Jay McCarthy stated in their article: "For some programs, proving that they have correct input-output behavior is only part of the story. As Crosby and Wallach observed, incorrect performance characteristics can lead to security vulnerabilities. Indeed, some programs and algorithms are valuable precisely because of their performance characteristics"[3].

From an academic view, this study adds to our knowledge of formal methods and proof making. By extending what Coq can check to include Braun Trees and their algorithms, it makes the range of data structures Coq can handle even broader. This effort will also push forward how we check software, benefiting not just Coq but also other languages. Using these checking methods could improve software quality and security, building trust in the correctness and efficiency of algorithms and helping users feel more confident in the technology.

The proposed research represents an impactful endeavor that addresses fundamental challenges in software correctness and reliability. By leveraging the capabilities of Coq, we aspire to contribute to the scarce state-of-the-art in program verification. Manipulating previous results and expertise in formal verification as well as with the aid of the course book, this research team is well-equipped to undertake this challenging yet rewarding endeavor. Building on the success of previous studies in formal verification using Coq, this research will push the boundaries of formal verification techniques, ensuring correctness, flexibility, and optimization in Braun Trees implementations.

3 State of the Art

One of Coq's most remarkable features is its ability to extract executable programs from verified proofs. This assists the developers with writing proofs in Coq, ensuring their correctness through formal verification, and then automatically translating those proofs into programs in languages such as OCaml. This extracted code inherits the correctness of the original proof, providing a way that connects formal verification and practical deployment. This is better explained in Pierre Letouzey's article, "Extraction in Coq: An Overview"[4]. This ability to extract programs makes Coq not just a theoretical tool but a practical solution for building high-assurance software systems.

Coq uses a unique approach to proof development based on tactics and proof terms. Tactics are essentially instructions that guide Coq in manipulating the current goal(the unproven statement) until it is reduced to a trivial truth. This interactive process empowers developers to reason about their programs step by step, building proofs through a series of well-defined actions. Beta Ziliani describes Coq's interactive proof development as a game. In this metaphor, the proof developer initiates by stating a theorem, and Coq responds by requesting proof strategies. The process involves providing tactics to simplify goals, with Coq adjusting and presenting updated goals. The game continues until all subgoals are resolved. If a tactic is invalid, Coq

raises an objection. This contrasts with non-interactive proof assistants like Twelf. The game-like metaphor incorporates the practicality of Coq's interactive proof development[5].

Beyond tactics, Coq utilizes multiple proof types to structure and guide the reasoning process[6]. These are:

1. Simplification: Reducing complex expressions to simpler forms using known equalities and rules
2. Rewriting: Replacing parts of the goal with equivalent expressions using predefined rewrite rules.
3. Case analysis: Breaking down the goal into different cases based on specific conditions and proving each case separately.
4. Induction: Proving a statement for all natural numbers by proving the base case and an inductive step.
5. Proof within proofs: Defining and using auxiliary lemmas within the main proof to modularize reasoning.

The academic work on Braun Trees is fairly limited. Studies made by Hoogerwoord made use of them in order to simplify the implementation of flexible arrays. By adopting a functional programming approach, Hoogerwoord simplified the derivation of these trees, making the implementation of array operations more straightforward and maintaining the logarithmic time complexity for these operations[7]. This approach not only demystified Braun and Rem's original design but also highlighted the adaptability of Braun Trees to functional programming paradigms, showcasing their potential for a wide range of applications. Okasaki introduced three new and innovative algorithms for Braun Trees, enhancing their efficiency and functionality[8]. Following this, Paulson explored another dimension of Braun Trees by applying them to priority queues, further expanding their applicability in computer science. Filliâtre's work stands out as a notable effort in formally verifying Braun Trees and one of Okasaki's efficient algorithms using the Why3 system. This represents a significant first step toward ensuring the correctness and reliability of Braun Trees through formal methods, setting a precedent for future verifications of data structures and algorithms[2].

Even though there has not been Braun tree structures implementations in Coq, insights from Niels Mündler and Tobias Nipkow on B+-trees verified in Isabelle/HOL shed light on the potential for Coq-based verification[9]. Their methodologies highlight how Coq's capabilities (rich type system, logical reasoning, and refinement techniques) can adapt well to formal verification of complex tree structures, including Braun Trees. The article's approach suggests that, similar to B+-trees, Braun Trees could benefit from Coq's separation logic framework, for ensuring correctness and optimizing memory usage. By drawing parallels between B+-tree verifications in Isabelle/HOL and potential Braun Tree implementations in Coq, we see a pathway for developing efficient, reliable tree-based systems within Coq. Such efforts promise enhanced correctness and functionality for a wide range of applications, from range queries to complex system constructions.

While Coq has proven competent and valuable for formal verification, challenges remain. One key challenge lies in extracting efficient programs from proofs. As stated by Christine Paulin-Mohring and Benjamin Wener in their paper: "The main problem with this methodology is obtaining efficient, realistic programs out of proofs"[10]. The article details several specific issues, such as controlling algorithms from proofs, avoiding unnecessary computations, and introducing fixpoints while maintaining termination guarantees. Other potential future improvements for Coq constitute of increased automation through more sofisticated automated proof tools that reduce the manual effort in building formal proofs, optimizing extraction techniques and wider integration of Coq with other existing development environments in order to promote formal verification. Current active research and development are addressing the aforementioned issues, with promising advancements in areas like automated proof techniques, program extraction optimization, and integration with existing development workflows.

For this section, all the specified references and information have been retrieved from the academic databases SmartCat, Web of Science and Google Scholar, with the aid of the following key terms: "Coq",

"proof assistant", "functional programming", "formal verification", "data structure", "algorithm".

4 Proposal

In this research project, we propose to investigate and explore the formal verification of Braun Trees and the algorithms of Okasaki within the Coq Proof Assistant. We will consider the following three operations proposed: calculating the size of a tree, creating a tree by copying, and converting a list to a tree[8]. As a bonus, we will add to this Tobias Nipkow and Thomas Sewell's proposed algorithm of converting a Braun Tree into a list[2], which can be observed in figure 2.

```
list_fast_rec :: 'a tree list ⇒ 'a list
list_fast_rec ts
= (let us = filter (λt. t ≠ ⟨⟩) ts
   in if us = [] then []
      else map value us @
           list_fast_rec (map left us @ map right us))
where value ⟨l, x, r⟩ = x, left ⟨l, x, r⟩ = l and right ⟨l, x, r⟩ =
r.
```

Figure 2: Recursive function of the proposed Braun Tree to List algorithm

The most relevant prior work in this domain includes the formal verification of data structures and algorithms in proof assistants such as Isabelle/HOL. While prior work has focused on verifying similar structures and algorithms, our research will specifically target the implementation of Braun Trees and the verification of Okasaki's algorithms in Coq, offering a novel perspective on formal verification techniques within a different proof assistant.

Our investigation will not only contribute to the development of formal proof languages but it will also benefit the engineers working in the software development field by minimizing the time wasted on program correctness, as it was stated earlier that "incorrect performance characteristics can lead to security vulnerabilities"[3]. In addition to this, Coq allows for a detailed and interactive verification process, ensuring the correctness and efficiency of the implemented structures and algorithms. Moreover, our research will contribute to the advancement of formal verification techniques in Coq, showcasing its capabilities in verifying intricate data structures like Braun Trees. The expected contributions of this research project include:

1. Implementation of Braun Trees in Coq, providing a formal representation of the data structure within the proof assistant.

2. Formal verification of key characteristics and invariants of Braun Trees, ensuring their correctness and reliability in Coq.

3. Verification of Okasaki's three algorithms in Coq as well as the fourth one proposed by Nipkow and Sewell, demonstrating the efficiency and validity of these algorithms within the Coq environment.

4. Additionally, if time allows, we will explore and compare the formal verification process in Coq with the already existing methods in Isabelle/HOL proof assistant, in order to highlight the strengths and potential improvements.

Throughout this project, we aim to provide a comprehensive and detailed inquiry into formal verification techniques for Braun Trees in Coq, offering valuable insights into the intersection of data structures, algorithms, and formal proof methodologies.

5 Methods

In order to address the objectives outlined in the proposal, this project has been divided into two components, theoretical and practical.

5.1 Theoretical Learnings

The theoretical aspect will involve gaining a comprehensive understanding of Braun Trees, their key characteristics, and the algorithms proposed for proof. This will commence with an in-depth study of the official book "Logical Foundations" and the reference manual available on their website, coq.inria.fr, serving as an entry-point to theorem proving in Coq[6]. In order to self-assess our comprehension, we will actively engage with the author's available exercises. Other existing literature available online will focus on data structures and algorithms with a strong emphasis on different types of trees and the corresponding characteristics. These resources, will provide a solid theoretical background on formal verification techniques and logical systems.

Furthermore, we will consult the articles on verification on trees in Isabelle/HOL[9] to understand the methodologies used in a similar proof assistant. This will aid in translating the theoretical concepts and algorithms into the Coq proof assistant environment.

5.2 Practical Work

The practical aspect will involve implementing Braun Trees in Coq and formally verifying their key characteristics and algorithms. The CoqIDE will be utilized for building. Specifically, the following aspects will be formalized and verified:

1. Braun Trees characteristic structure: Nodes, shape, recursive nature
2. Operations: Tree traversal, insertion and deletion
3. Okasaki's algorithm for calculating the size of a tree
4. Okasaki's algorithm for creating a tree by copying
5. Okasaki's algorithm for converting a list to a tree
6. Nipkow and Sewell's algorithm for converting a tree to a list

Our research paper will be written in LaTeX, using the online editor - Overleaf. The code implementations, proofs, and documentation, will be stored on GitHub in case a backup is needed. This project is feasible with standard computing resources, utilizing a personal computer for development and the CoqIDE for formal verification. The required software, including Coq and CoqIDE, is freely available and easily accessible. On top of this, a collection of Coq scripts and libraries, which can aid beginners, is available on GitHub. A snapshot showcasing the environment's interface can be seen in Figure 3. The research will be conducted individually and regular meetings with the supervisor will be scheduled to discuss progress, address questions, and receive guidance either weekly or fortnightly. By dividing the project into theoretical and practical components and leveraging open-source online resources, this research project is feasible to achieve its objectives.

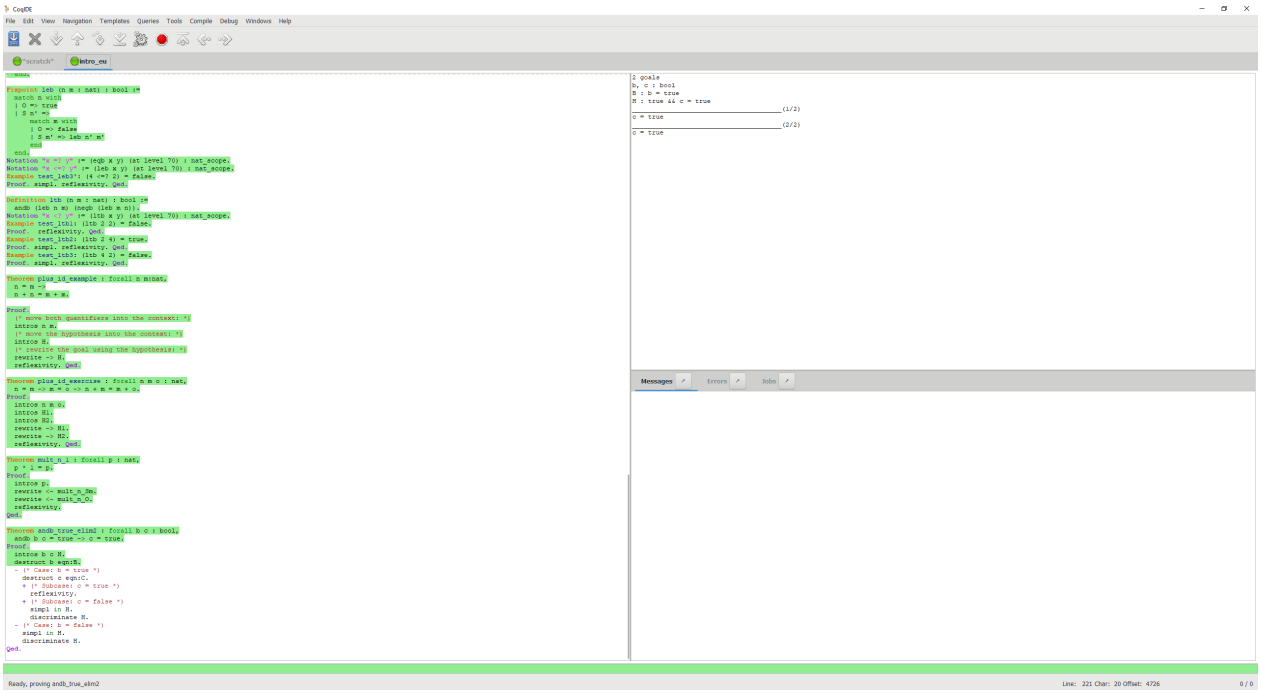

Figure 3: proof by case analysis in CoqIDE

6 Planning

The table below generates an overall summary of the expected spread of work throughout the Bachelor Thesis project. Every segment includes a variety of assignments that need to be finished before the end of the designated week. On top of our assigned milestones, there will be either weekly or fortnightly meetings scheduled with the supervisor in order to address the project's evolution and potential improvements. In case a meeting will be postponed or cancelled, the table will be updated accordingly. With this we will help Talk about the problem of formal proofs Mention it will benefit the evolution of verified formal proofs programming languages etc... In the section proposal: "Transalate the code from Isabelle to Coq. Do it in another formal proof language in order to contribute to the formal proof area.

Deadline	Description	Deliverables and Milestones
Week 0	Inquiring into Coq Proof Assistant Created storage system for backup	Setting up the environment Setting up the GitHub repository
Week 1	Started learning Coq Familiarizing with the software and the ide Researching academic literature	Obtaining theoretical framework Working on Coq book assignments Started reading relevant articles and books in order to get accustomed to the topic work Meeting with the supervisor
Week 2	Fully finished reading the past literature on Coq and Braun Trees Finished learning Coq Basics and Proof by Induction	Starting writing Thesis and first lines of code Gaining necessary knowledge to start the implementation Meeting with the supervisor
Week 3	Thesis writing Practice in CoqIde	Created Thesis Outline Finsihing working on Coq book assignments Meeting with the supervisor
Week 4	Final literature assessment Further improving Coq theoretical and practical skills First lines of code created and stored on GitHub	Code implementation choices made Continuing with the Logical Foundations Starting the coding part of the project Meeting with the supervisor
Week 5	Mastering Coq Creating the data structures in Coq	Completing the Logical Foundations Book and other counterparts Implementing Braun Trees Meeting with the supervisor
Week 6	Completed the Draft of the Research Proposal Formally verifying operations on Braun Trees	Completing the Research Proposal Updating Code and Repository Meeting with the supervisor
Week 7	Formally verifying Okasaki's algorithms Formally verifying Nipkow and Sewell's algorithm	Implementing the 3 formal proofs Implementing the last formal proof Meeting with the supervisor for discussing the code implementation
Week 8	Updating code according to the supervisor feedback Analyze the result and add it to the research paper	Finishing the coding section of the thesis Added findings into the research paper Meeting with the supervisor
Week 9	Send in thesis draft Rehearse for Bachelor Symposium	First Thesis Project version Preparations for the presentation
Week 10	Hold presentation in BSc Symposium Process feedback and update draft accordingly	Thesis presentation completed Incorporate feedback into the thesis Meeting with the supervisor
Week 11	Finalize thesis paper and hand it in together with the code	Thesis project accomplished

Table 1: Project Schedule

References

[1] W. Braun and M. Rem, "A logarithmic implementation of flexible arrays," *Memorandum MR83/4, Eindhoven University of Technology*, 1983.

[2] T. Nipkow and T. Sewell, "Proof pearl: Braun trees," in *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, ser. CPP 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 18–31. [Online]. Available: https://doi.org/10.1145/3372885.3373834

[3] J. McCarthy, B. Fetscher, M. S. New, D. Feltey, and R. B. Findler, "A coq library for internal verification of running-times," *Science of Computer Programming*, vol. 164, pp. 49–65, 2018, special issue of selected papers from FLOPS 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167642317300941

[4] P. Letouzey, "Extraction in coq: An overview," *4th Conference on Computability in Europe, CiE 2008 Athensk*, pp. 359–369, 2008.

[5] B. Ziliani and D. Dreyer, "Interactive typed tactic programming in the coq proof assistant," *Saarlandische Universitats- Und Landesbibliothek*, pp. 1–217, 2015.

[6] B. C. Pierce, A. A. de Amorim, C. Casinghino, M. Gaboardi, M. Greenberg, C. Hriţcu, V. Sjöberg, and B. Yorgey, *Logical Foundations*, ser. Software Foundations, B. C. Pierce, Ed. Electronic textbook, 2023, vol. 1, version 6.5, http://softwarefoundations.cis.upenn.edu.

[7] R. R. Hoogerwoord, "A logarithmic implementation of flexible arrays," in *Mathematics of Program Construction*, R. S. Bird, C. C. Morgan, and J. C. P. Woodcock, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 191–207.

[8] C. Okasaki, "Three algorithms on braun trees," *Journal of Functional Programming*, vol. 7, pp. 661 – 666, 1997. [Online]. Available: https://api.semanticscholar.org/CorpusID:34429104

[9] N. Mündler and T. Nipkow, "A verified implementation of b+-trees in isabelle/hol," in *Theoretical Aspects of Computing – ICTAC 2022*. Cham: Springer International Publishing, 2022, pp. 324–341.

[10] C. Paulin-Mohring and B. Werner, "Synthesis of ml programs in the system coq," *Journal of Symbolic Computation*, vol. 15, no. 5, pp. 607–640, 1993.