Research Proposal: Modularity in Proof Assistants

Yee-Jian Tan

January 31, 2025

1 Introduction

Software underpins our world, and in 2018, the digital sector contributed £149bn to the UK economy. Software also comes with errors, sometimes with catastrophic implications. In cases where faults have catastrophic results, or the software must withstand a determined attacker, the guarantees that testing provides might not be strong enough – we possibly want to verify that a program behaves exactly as described. Proof assistants allow us to develop such mathematically sound proofs and show that all inputs lead to the desired behaviour. However, despite the stronger guarantee than testing, they are also unaffordable for the vast majority of applications.

Motivation One of the main hindrances is lacking modularity: Even small changes and extensions entail non-trivial code modifications. For example, proof assistants are commonly used to ensure the safety of domain-specific programming languages, i.e., programming languages tailored to a specific problem, by providing proofs about their metatheory, custom compilers, and correctness proofs for these custom compilers. Small modifications of the original language (e.g., extensions or partial use) require setting up a completely new code base. Therefore, despite their prevalent role in research on programming languages and compilers, the size of projects scales only slowly: central notions of theorem provers prevent modularity and hence setting up realistic libraries. Limited modularity moreover means that the automatic development of proofs is tough; the search space is simply too big.

2 Proposal

Research Hypothesis This PhD project would build on the work of Forster and Stark 2020 and investigate a new form of modularity for interactive definitions and proofs via open recursion and tailored tool support in the Coq proof assistant. If successful, this would allow extending code and proofs in hindsight.

This would not only make existing developments much easier to maintain but also allow (as of now impossible) libraries of reusable language components.

As a second step, we propose a goal of automatically transforming a traditional development into a modular development – hence profiting from the big amount of verified code that already exists. The proposed research would not only make existing developments much easier to maintain but also allow libraries of reusable language components and simplify the automatic deduction of proofs. If successful, this proposal might be applicable to several proof assistants, such as Coq or Lean.

This PhD would take place with Kathrin Stark (Assistant Professor, Heriot-Watt University) and Yannick Forster (tenured researcher, Inria, Cambium Team, Paris).

Literature Review Other than the motivating work of this PhD, Forster and Stark 2020, several other works also attempt to address the modularity problem. Schwaab and Siek 2013 adopted a similar approach in Agda but did not go as far in proofs and their user support. Delaware et al. 2013 encode inductive data types in Coq as Church Encodings. More recently, Jin, Amin, and Zhang 2023 integrated Family Polymorphism into Coq to this end, extending the language and the base theory of Coq.

These approaches work well theoretically, but are not practical for the users of proof assistants because it requires them to either write in an encoding that could be hard to decipher, or work with a new type theory which could be incompatible with the users' previous proofs or even their metatheory.

Proposed Methodology This project aims to utilize metaprogramming to implement modularity. In particular, we will use the MetaCoq metaprogramming framework to generate boilerplate code in Coq that translates user code into a modular representation.

Two central parts of the MetaCoq project are a formalisation of Coq's type-theory in Coq (Sozeau, Boulier, et al. 2020) and, on top of that, verified implementations of a type checker (Sozeau, Forster, et al. 2023). The formalisation of type theory faithfully captures the typing rules of Coq in an inductive predicate, and is parameterised in a guard checker function which is required to fulfill some basic properties such as being stable under reduction and substitution. Based on this formalisation, crucial properties such as subject reduction (types are preserved by reduction) and canonicity (normal forms of inductive types start with a constructor) are proved (Sozeau, Forster, et al. 2023). In particular, the verified type checker implemented in MetaCoq and the specification for the type theory of Coq can be used to verify the correctness modularity tool that this PhD aims to develop, guaranteeing correctness on top of its usability.

Since my two previous research projects/internships, summing up to about a year, were done using the MetaCoq framework, I am reasonably familiar with the tool. Relevant experiences are described in the following section.

Planned timeline of the PhD The PhD will start with an extensive literature review of the current projects that work on modularity, and create a benchmark to measure the performance and usability of our upcoming implementation. We aim to set up a benchmark suite that (a) is broad in that it is independent of the proof assistant, (b) has a realistic scope, covering important language features (data types, constructors, recursive functions, induction principles, dependent predicates, dependent types) and different levels of modularity (e.g., changing the type of modular functions, dependency and exclusivity of different features), and (c) clearly defines evaluation criteria.

The bulk of the PhD will be working on implementing via metaprogramming a modular syntax in Coq, using the MetaCoq metaprogramming framework. We develop (a) the principles and (b) the tool support for a basic notion of modularity in the Coq proof assistant. We aim to base our approach on open types, an approach where an inductive type is separated into its semantic components, defined as a separate set of inductive types parameterised by its final closed type. The types from the various modules will be combined into a single traditional inductive type only in the last step of the proof.

Tool-wise, we aim to use a mix of code generation and metaprogramming to reduce the otherwise inevitable boilerplate. A successful implementation should require minimal or no extra effort from the user to use, yet achieving the goals of modularity by giving the heavy lifting to the metaprogramming. Additionally, the correctness of the tool will also be verified mechanically using the verified type checker of MetaCoq.

3 Previous Experience

I am a second-year master's student in the Parisian Masters of Research in Computer Science (MPRI) in École Polytechnique, France, and I have previously completed two bachelor's degrees in Mathematics and Computer Science in the National University of Singapore. In my training, I have taken relevant courses about Dependent Types, Proof Assistants, Functional Programming, and Program Semantics, as well as two summers of research internships on metaprogramming in Coq, which are directly relevant to the PhD topic.

Research Experience Between May and August 2024, I completed my four-month Master's Year 1 internship with Yannick Forster at the Cambium team in INRIA Paris, working on the project "Towards Formalizing the Guard Checker of Coq", which was awarded an Honourable Mention (*Mention de Félicitations*) from École Polytechnique for Master's Year 1 internships. The guard checker of Coq is the part of Coq's kernel that checks for the termination of recursive functions in Coq, which is crucial for Coq's soundness as a proof assistant. In this project, I implemented the guard checker of Coq in the Coq proof assistant and documented its features and behaviours in my report. I presented the project publicly twice, once as an invited speaker at the Workshop on the Guard Condition of Coq, held on 3 June by the RECIPROG project in Nantes, France;

as well as during the Coq Workshop 2024, affiliated with ITP 2024 in Tbilisi, Georgia.

In 2022, I worked on formalizing the Modules system of Coq in MetaCoq during my undergraduate internship with Nicolas Tabareau. Unthe the supervision of co-supervisors Martin Henz and Yue Yang, I wrote my bachelor's thesis on the same topic, which received an A grade from the National University of Singapore, thus completing a dual bachelor's degree in Computer Science and Mathematics with Highest Distinction.

These research internships both use the MetaCoq metaprogramming framework for Coq, which will be the tool to implement the first project of modularisation as proposed, which I already have familiarity with. This will facilitate my onboarding for the project and enable a smoother commencement of the PhD project.

Technical Skills/Theoretical Background Among the courses I studied in Master's Year 1, INF551 Computational Logic, taught by Samuel Mimram, was the most relevant to my research interests. I was awarded full marks (20/20) in this course, where I implemented, for the final project, a dependent type-checker / proof-assistant supporting Dependent Function types, Equality types, and Natural Number types. I have also completed courses on Proof Assistants (2.7.2), Foundations of Proof Systems (2.7.1) in my Master's Year 2, as well as the Category Theory and Proof Theory courses in the Master's in Mathematical Logic and Foundations of Computer Science (LMFI). I am currently taking the courses on Models of Programming Languages (2.2), and Functional Programming (2.4). These courses provide the theoretical foundations needed to carry out the research on modularity in proof assistants.

References

Delaware, Benjamin et al. (2013). "Modular monadic meta-theory". In: *ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25 - 27, 2013*. Ed. by Greg Morrisett and Tarmo Uustalu. ACM, pp. 319–330. DOI: 10.1145/2500365.2500587. URL: https://doi.org/10.1145/2500365.2500587.

Forster, Yannick and Kathrin Stark (2020). "Coq à la carte: a practical approach to modular syntax with binders". In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pp. 186–200.

Jin, Ende, Nada Amin, and Yizhou Zhang (2023). "Extensible Metatheory Mechanization via Family Polymorphism". In: *Proc. ACM Program. Lang.* 7.PLDI, pp. 1608–1632. DOI: 10.1145/3591286. URL: https://doi.org/10.1145/3591286.

Schwaab, Christopher and Jeremy G. Siek (2013). "Modular type-safety proofs in Agda". In: *Proceedings of the 7th Workshop on Programming languages meets program verification, PLPV 2013, Rome, Italy, January 22, 2013*. Ed.

by Matthew Might et al. ACM, pp. 3–12. DOI: 10.1145/2428116.2428120. URL: https://doi.org/10.1145/2428116.2428120.

Sozeau, Matthieu, Simon Boulier, et al. (2020). "Coq Coq correct! verification of type checking and erasure for Coq, in Coq". In: *Proc. ACM Program. Lang.* 4.POPL, 8:1–8:28. DOI: 10.1145/3371076. URL: https://doi.org/10.1145/3371076.

Sozeau, Matthieu, Yannick Forster, et al. (Apr. 2023). "Correct and Complete Type Checking and Certified Erasure for Coq, in Coq". working paper or preprint. URL: https://inria.hal.science/hal-04077552.