doi:10.1017/S095679682510004X

PhD Abstracts

GRAHAM HUTTON
University of Nottingham, Nottingham, UK
(*e-mail:* graham.hutton@nottingham.ac.uk)

Many students complete PhDs in functional programming each year. As a service to the community, twice per year the *Journal of Functional Programming* publishes the abstracts from PhD dissertations completed during the previous year.

The abstracts are made freely available on the JFP website, i.e. not behind any paywall. They do not require any transfer of copyright, merely a license from the author. A dissertation is eligible for inclusion if parts of it have or could have appeared in JFP, that is, if it is in the general area of functional programming. The abstracts are not reviewed.

We are delighted to publish twelve abstracts in this round and hope that JFP readers will find many interesting dissertations in this collection that they may not otherwise have seen. If a student or advisor would like to submit a dissertation abstract for publication in this series, please contact the series editor for further details.

Graham Hutton
PhD Abstract Editor

Verification of Smart Contracts using the Interactive Theorem Prover Agda

FAHAD ALHABARDI
Swansea University, Swansea, UK

Date: July 2024; Advisor: Anton Setzer
URL: https://cronfa.swan.ac.uk/Record/cronfa67283/

The goal of this thesis is to verify smart contracts in Blockchain. In particular, we focus on smart contracts in Bitcoin and Solidity. In order to specify the correctness of smart contracts, we use weakest preconditions. For this, we develop a model of smart contracts in the interactive theorem prover and dependent type programming language Agda and prove the correctness of smart contracts in it. In the context of Bitcoin, our verification of Bitcoin scripts consists of non-conditional and conditional scripts. For Solidity, we refer to programs using object-oriented features of Solidity, such as calling of other contracts, full recursion, and the use of gas in order to guarantee termination while having a Turing-complete language. We have developed a simulator for Solidity-style smart contracts. As a main example, we executed a reentrancy attack in our model. We have verified smart contracts in Bitcoin and Solidity using weakest precondition in Agda.

Furthermore, Agda, combined with the fact that it is a theorem prover and programming language, allows the writing of verified programs, where the verification takes place in the same language in which the program is written, avoiding the problem of translation from one language to another (with possible translation mistakes).

Agent-Based Logics in Dependent Type Theory

COLM BASTON
University of Nottingham, Nottingham, UK

Date: May 2025; Advisor: Venanzio Capretta and Graham Hutton
URL: https://colmbaston.uk/files/thesis.pdf

This thesis is on the formalisation of mathematics in Martin-Löf type theory. This is a class of dependently typed functional programming languages whose rules form a language suitable for the statement and proof of mathematical theorems. Programs in type theory can be mechanically checked to be well-typed, which corresponds to proofs being checked to be valid. In particular, we explore the semantics of certain agent-based logics and seek to give them appropriate representations using the concepts of type theory.

We embed the syntax of epistemic modal logic in type theory as predicates over a type of possible worlds. Knowledge operators are defined in this setting by stating a set of properties they must satisfy. We prove this knowledge operator semantics equivalent to the traditional relational semantics of epistemic logic. Common knowledge is then defined in the embedding as a coinductive predicate, whose proofs are infinite data structures. We prove this representation is equivalent to the intuitive definition as iterated universal knowledge, and prove it is equivalent to the relational interpretation. In coalition logic, we represent game forms and playable effectivity functions in type theory and outline a proof of their equivalence.

Building Trustworthy Smart Contracts using Interactive Theorem Proving

DANIEL BRITTEN
The University of Waikato, Hamilton, New Zealand

Date: October 2024; Advisor: Steve Reeves and Vimal Kumar
URL: https://hdl.handle.net/10289/16566

There are varying approaches to the verification of smart contracts using formal methods. This thesis advocates for the use of high-level specifications coupled with a verified compiler to low-level bytecode, such as for the Ethereum Virtual Machine. Taking this approach allows for specifications to more closely match natural language, while ensuring that the specifications apply to the real bytecode executed on-chain.

Interactive theorem proving can provide the foundation for developing provably correct smart contracts. Due to the immutable nature of smart contracts and their potential to manage highly valuable assets and tokens representing power, techniques to ensure their correctness are of paramount importance.

This thesis extends the DeepSEA (Deep Simulation of Executable Abstractions) smart contract language targeting the Ethereum Virtual Machine by mitigating the issues associated with reentrancy and introducing a model of relevant aspects of a blockchain. This enables the specification and verification of two case studies, which exemplify the approach of developing provably correct smart contracts.

The specifications for the case studies are written in the language of the Coq Proof Assistant, making arbitrary mathematical statements expressible. The blockchain model enables stating and proving temporal properties relating to the execution of a smart contract over time.

While smart contracts are an ideal application area for formal methods in general and interactive theorem proving in particular, the techniques exemplified in this thesis could be applied throughout software engineering. The relatively young age of smart contract languages and typically small size of smart contract programs makes the application of interactive theorem proving more tractable. Future work could involve demonstrating the applicability to many interrelated smart contracts and to larger software projects in different domains.

Multimode Type Theory as a Library in Type Theory

JORIS CEULEMANS
KU Leuven, Leuven, Belgium

Date: March 2025; Advisor: Dominique Devriese and Andreas Nuyts
URL: https://lirias.kuleuven.be/retrieve/801459

We investigate the feasibility of implementing modal type theory as a library within type theory. More specifically, we try to construct a formalisation in the dependently typed language Agda of Multimode Type Theory (MTT). MTT is a general framework for modal dependent type theories developed by Gratzer, Kavvos, Nuyts and Birkedal, and is parametrised by a mode theory that specifies the available modalities and their interaction.

First, we present Sikkel: an Agda library for Multimode Simple Type Theory (MSTT). MSTT is a simply typed fragment of MTT and can therefore be used to write programs but not proofs. Sikkel contains an implementation of the MSTT syntax in which a user can write modal programs. The well-typed programs will then be interpreted in Sikkel's semantic layer, which is an Agda formalisation of presheaf models of type theory. This interpretation allows us to extract certain Sikkel programs to actual Agda programs, which can then be used in other parts of an Agda project where the expressivity offered by modalities is not needed. Just like MTT, our library is parametrised by a mode theory and can be applied in different modal situations. We illustrate this generality by implementing example programs using guarded recursion and proving a result using a restricted form of parametricity.

Subsequently, we develop a multimode logical framework μLF for MSTT, in which a user can prove properties of MSTT programs by making use of modal primitives. This logical framework is implemented in an extension of the Sikkel library called BiSikkel, which is again parametrised by a general mode theory. Although μLF is not as expressive as a full implementation of dependently typed MTT in Agda, we can use BiSikkel to prove interesting properties of streams in guarded recursive type theory and to implement an example involving parametricity predicates.

Finally, we construct a structurally recursive substitution algorithm for MTT. Previously, substitution in MTT was based on an axiomatic system for which it is a priori not clear that every substitution can be computed. We prove the soundness and completeness of our substitution algorithm with respect to the MTT axioms.

Categorical Foundations of First-Order Abstract Syntax

LAWRENCE DUNN
University of Pennsylvania, Philadelphia, USA

Date: May 2025; Advisor: Val Tannen and Steve Zdancewic
URL: https://comono.id/dissertation.pdf

The representation of syntax is central to programming languages and formal verification. Yet existing approaches abstract away from the first-order nominal treatment of syntax found in textbooks, where substitution requires systematic renaming of bound variables to avoid variable capture. Although this representation closely mirrors what compilers actually implement, it remains notoriously difficult to formalize in proof assistants. Moreover, translating to a more convenient internal representation does not eliminate the need to reason about nominal variable binding; it merely shifts the burden to verifying the correctness of the translation. A rigorous mathematical treatment of concrete first-order representations of syntax is therefore required. This work introduces decorated traversable monads (DTMs) as a foundation for extrinsically scoped, extrinsically-typed first-order abstract syntax. DTMs unify several classical and lesser-known categorical structures under a novel set of coherence laws relating them to each other. The result can be characterized from three theoretically equivalent perspectives, each offering distinct practical advantages. The accompanying Rocq library, Tealeaves, formalizes these results and establishes their practical applicability. Tealeaves faithfully reproduces the functionality of other tools used with Rocq, while extending them with new capabilities. Tealeaves naturally supports variadic and mutually recursive binders, and it provides certified translations between different representations of variables within a unified framework. Tealeaves provides the first datatype-generic formalization of α-equivalence that both (i) matches its conventional definition and (ii) comes with an executable proof that α-equivalence classes correspond bijectively to well-formed locally nameless terms, which in turn correspond to de Bruijn terms in a well-formed environment. This result enables new forms of modular, reusable, and mechanically verified reasoning about capture-avoiding substitution, helping bridge a critical gap in the formal verification of programming languages and their implementations.

Back to Beck: The State Monad and the Continuation Monad from the Viewpoint of Monadicity

YUNING FENG
University of Birmingham, Birmingham, UK

Date: August 2024; Advisor: Paul B. Levy
URL: https://etheses.bham.ac.uk/id/eprint/15269/

We introduce control algebras, which are algebras of two operations called call/cc and abort. The two operations are natural ones that occur in programs that need to manipulate their planned future work. The significance of the formulation is that it is operationally natural, yet the algebras are equivalent to the formulation instantiated from a general notion of algebras in category theory, namely algebras of a monad.

We also consider the control effect and the state effect from the perspective of Beck's monadicity theorem. In particular, we notice connections between operationally meaningful objects in the effects and coequalisers of Beck's pair; the latter are important ingredients in Beck's monadicity theorem. In the case of control, such objects are the set of continuations for a given algebra of the control monad; in the case of state, such objects are the set of determined elements.

Scaling the Evolution of Verified Software

KIRAN GOPINATHAN
National University of Singapore, Singapore, Singapore

Date: August 2024; Advisor: Ilya Sergey
URL: https://kirancodes.me/docs/thesis.pdf

Formal verification is on the cusp of becoming mainstream. The past decade of verification research has convincingly demonstrated the efficacy of verification techniques for certifying large-scale real world software systems, such as compilers, web servers, distributed systems, file systems and even whole operating system kernels. Despite these successes, however, the challenge of *maintaining* such verified artefacts in the face of their inevitable evolution unfortunately remains largely unaddressed and persists as a critical obstacle that must be overcome if verification is ever to achieve wider adoption.

The focus of this thesis is in tackling this challenge of maintaining evolving verified software. To this end, this thesis identifies and develops a suite of techniques that can be deployed and applied in anger to manage the burden of verified software maintenance. The thesis incorporates and adopts a threefold strategy for proof-maintenance, investigating tackling evolution through: first, (1) classical formal techniques of composition and the use of reduction arguments, then second, (2) through the lens of certified synthesis, drawing from prior works on proof-carrying-code and finally (3) under the umbrella of repair, developing the novel technique of proof-driven-testing to scale proof-repair strategies to operate in a real-world setting. Each technique is motivated using a running case study, provided in an accompanying artefact, that demonstrates the efficacy of the technique in a practical setting.

Through the evaluation of each strategy, this thesis finds that the proposed framework is effective at considerably reducing the maintenance burden across a variety of verified codebases with significant impacts, including, for example, reducing the sizes of manual proof scripts of complex and subtle algorithms by half, or reducing the time spent by proof engineers by hours, or, in some cases, eliminating the maintenance burden entirely and synthesizing all required code and proofs automatically.

Functional Abstract Interpretation

SEBASTIAN GRAF
Karlsruhe Institute of Technology, Karlsruhe, Germany

Date: December 2024; Advisor: Gregor Snelting and Jonathan Immanuel Brachthäuser
URL: https://dx.doi.org/10.5445/IR/1000177868

Functional programming languages encourage expressing large parts of a program as declarative data flow pipelines, free of side effects such as shared mutable state. Such pipelines traverse recursive data by pattern matching and share the repetitive code of these traversals by defining higher-order functions. Writing programs in functional style eliminates large classes of programmer errors, hence higher-order functions and pattern matching have been adopted by most general purpose programming languages today.

However, pattern matching introduces new modes of failure as well: It is easy to forget a case, and input data that is not covered leads to a crash at runtime. Thus, a compiler should integrate a static program analysis to warn about such uncovered pattern-matches before the program is run.

A compiler should also generate fast code for programs involving higher-order functions. More than 30 years of practical research have evolved the Glasgow Haskell Compiler (GHC) into an industrial strength tool. This evolution brought forth a number of useful and efficient compiler optimisations that are informed by static higher-order analyses. However, the more proficient a higher-order analysis becomes, the harder it gets to explain its implementation to maintainers, let alone convince interested bystanders of its correctness.

In this thesis, I present two results of my work to improve GHC: the first is a static analysis for pattern-match coverage checking that is both more efficient and more precise than the state of the art; the second is a design pattern for deriving static higher-order analyses and dynamic semantics alike from a generic denotational interpreter, in order to share intuition and correctness proofs. This design pattern generalises Cousot's seminal work on trace-based abstract interpretation to higher-order analyses such as GHC's Demand Analysis.

Foundations and Applications of Modal Type Theories

JASON Z. S. HU
McGill University, Montreal, Canada

Date: May 2025; Advisor: Brigitte Pientka
URL: https://escholarship.mcgill.ca/concern/theses/bv73c644d

Over the past few decades, type theories as mathematical foundations have been extensively studied and are well understood. Many proof assistants implement type theories and have found important applications to provide critical security guarantees. In these applications, users often write meta-programs, programs that generate other programs, to implement proof search heuristics and improve their work efficiency. However, as opposed to the deep understanding of type theories, it remains unclear what foundation is suitable to support meta-programming in proof assistants. In this thesis, I investigate modal type theories, a specific approach to this problem. In modal type theories, modalities are a way to shallowly embed syntax into the systems, so users can write meta-programs that manipulate syntax through these modalities.

I explore two different styles of modal systems. In the first part, I investigate the Kripke-style systems, which faithfully model the familiar quasi-quoting style of meta-programming. I develop an explicit substitution calculus and scale it to dependent types, introducing Mint. I prove strong normalization of Mint, which implies its logical consistency, using an untyped domain model.

Nevertheless, the Kripke-style systems only support composition and execution of code, and they cannot easily support a general recursion principle on the structure of code. To support such a general recursion principle, I develop the layered style, where a system is divided into nested layers of sub-languages. The layered style scales quite naturally to dependent types, introducing DeLaM. DeLaM allows users to compose, execute and recurse on dependently typed code. I prove that DeLaM is weakly normalizing and its convertibility problem between types and terms is decidable. Hence, DeLaM provides a type-theoretic foundation to support type-safe meta-programming in proof assistants.

Foreign Function Verification Through Metaprogramming

JOOMY KORKUT
Princeton University, Princeton, USA

Date: November 2024; Advisor: Andrew W. Appel
URL: http://arks.princeton.edu/ark:/88435/dsp01k930c143z

CertiCoq is a compiler from Coq to C that is verified in Coq. Thanks to CertiCoq's mechanically checked proof of compiler correctness, users can be sure that programs they write and verify in Coq's rich type system output the same results when compiled to C (and to machine language, via the CompCert verified compiler). However, in practice, large programs are rarely written in a single language; additional languages are used for better performance or for capabilities that the primary language lacks. In particular, because Coq lacks user-defined primitive types, mutation and input/output actions, CertiCoq-compiled code must interact with another language to have those capabilities. Specifically, Coq code must be able to call C code and C code must be able to inspect and generate Coq values and call Coq code. But what happens to the correctness proofs when these two languages interact? A foreign C function has to be memory-safe, return the expected result and have only the expected side effect. A specification that expresses these requirements must combine plain Coq, for the functional parts, and a program logic for C (such as the Verified Software Toolchain [VST]), for the verification of C functions. While VST is embedded in Coq, its specification language is quite different. VST proofs about C foreign functions do not directly translate to proofs about Coq primitives either. Bridging this gap by connecting plain Coq and VST theorems requires a system of techniques and methods, many of which are made feasible by using metaprogramming as a general methodological approach. In this dissertation, I describe these techniques and methods. Using these methods, the user can relate foreign functions and types to their functional models, generate VST specifications about the foreign C functions and write plain Coq proofs about the Coq counterparts of the foreign functions. I also provide examples of Coq programs with primitive types, mutation and I/O actions, along with specifications and proofs about these programs, to demonstrate VeriFFI, the verified foreign function interface for CertiCoq.

Types That Count: A Journey Across Qualitative and Quantitative Intersection Type Disciplines

DANIELE PAUTASSO
University of Turin, Turin, Italy

Date: May 2025; Advisor: Felice Cardone and Luca Paolini
URL: https://hdl.handle.net/2318/2073973

Since their introduction more than 45 years ago, *intersection types* have found countless uses in both the syntactic and semantic study of programming languages. This thesis continues such tradition, presenting two lines of research centred around intersection type disciplines.

A first line explores the connections between idempotent (qualitative) and non-idempotent (quantitative) intersection type systems for lambda-calculus. We start by introducing *uniform types*, a decidable restriction of non-idempotent intersection types that turns out to be the quantitative counterpart of simple types. We provide two alternative proofs of this correspondence. In the first one, a general intersection type inference *semi-algorithm* is progressively refined into an *algorithm* tailored to the uniform system, exploiting the fact that term normalisation and termination of inference procedures are intimately related. Following a complementary approach, the result is also achieved by means of a novel, mechanically verified proof of *subject expansion* in intersection type systems without nullary intersection. As notable by-products, our investigation yields a new syntactic proof of *strong normalization for simply typed terms*, as well as a family of *perpetual reduction strategies*. Finally, leveraging intuitions developed in the uniform case, and taking inspiration from Linear Logic, we present some preliminary ideas on a general technique for transforming *qualitative* type derivations into *quantitative* ones.

A second line of research employs quantitative types in the study of probabilistic computations, with the aim of integrating the expressiveness of higher-order programming languages with the efficiency of Bayesian networks. Specifically, we introduce a Linear Logic-inspired probabilistic lambda-calculus and an associated non-idempotent intersection type system, whose intrinsic resource-consciousness allows to track the generation of random variables and the dependencies between them. We use this information to endow typed terms with a *compositional* and *cost-aware* semantics based on *factors* and to prove *soundness and completeness* of our language w.r.t. Bayesian networks. Higher-order programs are put into correspondence with Bayesian networks operationally, via *rewriting*, and semantically, via an *acyclic graph structure* built on top of type derivations. Our perspective contributes to a rigorous understanding (in a functional setting) of *templates*, informal depictions of stochastic models used to enhance the expressiveness of standard Bayesian networks.

Proof Theory, Syntactic Representations, Logic, and Sharing

JUI-HSUAN WU
Institut Polytechnique de Paris, Palaiseau, France

Date: December 2024; Advisor: Dale Miller and Beniamino Accattoli
URL: https://theses.fr/2024IPPAX123

This thesis focuses on the design of terms and the structure of proofs. Generally speaking, a term can denote any syntactic structure, including sentences, programs, etc. A notion of *sharing*, made possible by naming and referencing terms, is essential in various settings. The goal of this thesis is to propose an appropriate syntax that is both expressive and compact and that provides a mechanism of sharing.

This is where structural proof theory comes in. In this thesis, we consider the implicational fragment of intuitionistic logic and base our study on focused proof systems, systems that refine sequent calculi by *focusing*, a notion introduced by Andreoli as a technique improving proof search in linear logic. Another key notion in our study is *polarisation*. In fact, the polarities of logical connectives in intuitionistic logic (and classical logic) are often ambiguous, unlike in linear logic. Positive or negative polarity can therefore be arbitrarily assigned to connectives and even to atoms. This choice of polarising connectives and atoms has no impact on the provability of a sequent but can induce proofs of very different forms. It is thanks to this aspect that we design different syntaxes using specific polarisations.

Using Liang and Miller's focused proof system LJF, we define two different syntaxes, namely the negative syntax and the positive syntax, by considering the two opposing uniform polarizations. The negative syntax is the usual tree-like syntax in which sharing is not possible while the positive syntax allows for sharing within a term via a restricted form of explicit substitutions. In the context of untyped λ-terms, these two syntaxes correspond to negative λ-terms (i.e. the usual λ-terms) and positive λ-terms, respectively. We then put our focus on the positive λ-terms and define the positive λ-calculus: a call-by-value λ-calculus with explicit substitutions. We show that, thanks to its compactness, the positive λ-calculus remarkably captures the essence of useful sharing, a notion of reduction on terms with sharing introduced by Accattoli and Dal Lago to study reasonable cost models of the λ-calculus.