PhD abstracts

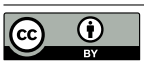
GRAHAM HUTTON
University of Nottingham, UK
(email: graham.hutton@nottingham.ac.uk*)*

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 twenty-two 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



© 2026 Copyright held by the owner/author(s).
This work is licensed under a Creative Commons Attribution 4.0 International License.

Journal of Functional Programming, Volume 36, Article 16 DOI: 10.46298/jfp.18491
Submitted June 2026; accepted August 2026

Proof assistant-based formalisation of core Erlang

PÉTER BERECZKY
Eötvös Loránd University, Hungary

Date: January 2026; Advisor: Dániel Horpácsi and Zoltán Horváth
URL: https://tinyurl.com/5tz652d2

The motivation of this dissertation is a particular field of computer science: refactoring existing code. Refactorings are program transformations which improve the internal structure without affecting the observable behaviour of the code. While there are dedicated refactoring tools in the industry, these tools are only verified by tests in most cases. Moreover, developers tend to do their refactorings by hand instead of using a tool, since they do not trust it; thus, more reliable solutions are necessary. The correctness of refactoring tools can be based on formal verification: by using a formal semantics of the object language, and a suitable program equivalence definition, we can show that a refactoring is correct if the programs before and after the transformation are equivalent. Note that a formal semantics is not only applicable to reasoning about refactorings, for example a semantics can also be used to verify the correctness of programs, compilers, and various language tools.

This dissertation discusses our work on formalising Core Erlang, the sublanguage of Erlang, which is an impure functional programming language with strict evaluation, uncurried function abstractions and applications, lightweight processes and asynchronous communication. We define the sequential semantics of Core Erlang in various definition styles (including big-step and small-step approaches), and compare and evaluate the definitions. Thereafter, we point out numerous semantic properties and applications of the sequential semantics, including validation, definitions of program equivalence, and refactoring correctness. Finally, we also define a representative, concurrent Core Erlang subset including the semantics of asynchronous message-passing, error handling, process spawning, and termination. We investigate refactoring correctness in the concurrent setup with program equivalence based on (barbed) bisimulation. All of our results are also machine-checked by the Coq proof assistant.

Internal constructions in homotopical type theory

JOSHUA CHEN
University of Nottingham, UK

Date: September 2025; Advisor: Nicolai Kraus and Graham Hutton
URL: https://repository.nottingham.ac.uk/handle/123456789/51717

The aim of this thesis is to investigate certain constructions that resist satisfactory full internalizations in plain homotopy type theory, i.e. intensional Martin-Löf type theory with the univalence axiom.

In Part I, we study internal higher categorical models of homotopical type theory, via wild categories with families (cwfs). We formulate coherence conditions on wild cwfs that suffice to recover properties expected of models of dependent type theory. The result is a definition of a 2-coherent wild cwf, which admits as instances both the syntax and the "standard model" given by a universe type. We also identify a higher "splitness" coherence condition that is satisfied by all set-level cwfs and univalent 2-coherent wild cwfs.

In Part II, we apply some of the theory developed in Part I and report on a partial investigation into the construction of type-valued Reedy-fibrant inverse diagrams in plain HoTT.

A foundationally verified intermediate verification language

JOSHUA M. COHEN
Princeton University, USA

Date: May 2025; Advisor: Andrew W. Appel
URL: http://arks.princeton.edu/ark:/88435/dsp01nz806311z

Annotation-based deductive verifiers have emerged in recent years as practical, capable, and increasingly scalable tools for verifying programs in languages including C, Rust, Go, OCaml, and Dafny. Intermediate verification languages (IVLs) like Why3, Boogie, and Viper have made it significantly easier to build such verifiers by performing part of the translation from high-level, expressive program logics to SMT formulas once and for all. But these verifiers are difficult to fully trust: their toolchains comprise hundreds of thousands of lines of code and bugs in any part can result in unsoundness. An alternate approach to building program verifiers is the foundational one, where the language semantics, program logic, and user-supplied proofs are defined in a proof assistant, giving an end-to-end soundness theorem.

These tools, including VST, CakeML, and Bedrock2, provide very strong guarantees but require manual proof and are inaccessible to non-experts. The two approaches have largely remained separate. IVL-based tools are not themselves verified, while interactive tools cannot take advantage of the automation provided by the IVL and SMT-based pipelines. In this thesis, we take steps towards bridging this gap by providing an implementation of the Why3 IVL in the Coq proof assistant, giving the first foundationally verified IVL. First, we give a novel formal semantics for a core subset of the logic implemented by Why3 – polymorphic first-order logic with recursive types, functions, predicates, and pattern matching. We prove sound a compiler from this logic to first-order logic, giving the first machine-checked soundness proofs for a sophisticated pattern matching compiler and a first-order axiomatization of algebraic data types. We then develop a new framework to idiomatically implement OCaml APIs in Coq and use this to produce a Coq implementation of the Why3 logic API. Our IVL implementation is fully executable both within Coq and via extraction to OCaml. Using the latter, we demonstrate our tool's practicality by running existing Why3-based tools and test suites against it, while the former allows our tool to serve as a future back-end for foundational verifiers such as VST.

Purely-functional stream processing

JONATHAN MICHAEL THOMAS DOWLAND
Newcastle University, UK

Date: November 2025; Advisor: Paul Watson
URL: http://theses.ncl.ac.uk/jspui/handle/10443/6607

Extracting value from streams of data generated by sensors and software is key to the success for many important problem domains including the Internet of Things (IoT).

However there are many non-functional challenges to be overcome in achieving this, including very high data rates, a deployment environment featuring nodes with differing capabilities and limitations, energy and bandwidth constraints, performance requirements, and security guarantees.

Most modern stream-processing systems leave addressing these challenges for the application programmer to solve by making manual adjustments to their program. This entanglement of the functional and non-functional aspects of stream processing increases the risk of mistakes and the potential cost of future maintenance.

We describe an alternative approach — a declarative architecture ("*StrIoT*") based on purely-functional programming. The application programmer provides a program encoding the functional requirements, a separate description of the operating environment, and the applicable non-functional requirements. *StrIoT* then derives functionally-equivalent program variants, generates corresponding deployment plans and automatically deploys the best-cost plan.

In order to explore the viability of purely-functional programming for this problem domain, we designed and built a proof-of-concept, end-to-end implementation of this architecture using the purely-functional language Haskell, Linux containers and orchestration technology.

This thesis focusses on two key components: the Optimiser and Evaluator.

The Optimiser leverages purely-functional semantics to apply term rewriting and derive functionally-equivalent variants of the user's Haskell program.

The Evaluator filters and selects deployments plans using modelling techniques including queueing theory, in order to select the plan which will perform best for two non-functional requirements: bandwidth and cost.

To evaluate these components and approaches we implemented several solutions to real-world stream-processing problems. In some cases, *StrIoT* can generate programs which meet applicable non-functional requirements even when the user-supplied program does not.

Increasing confidence in dependently typed programs

THOMAS EKSTRÖM HANSEN
University of St Andrews, UK

Date: July 2026; Advisor: Edwin Brady
URL: https://doi.org/10.17630/sta/1560

There exist numerous methods for testing and verifying software, including property based testing, automatic code generation, model checking, and proof assistants. Although these tools help ensure that software behaves according to its specification, they are typically deployed in isolation rather than as reinforcements for one another. As software complexity grows, so does the complexity of modelling and checking the system. While rigorous approaches such as model checking and formal proofs provide strong correctness guarantees, these have difficulty scaling and often require reimplementing the code in a separate verification language. This risks the semantics being lost in translation, as there is no guarantee that the verified model behaves the same as the deployed code. Programming languages with dependent types – such as Idris2 – present novel ways to enforce correctness through dependently typed terms. However, the complexity of the dependent types also grows with the complexity of the system. This raises the question: "How does one know that the typed model used for checking matches the specification?" Since dependent types are effectively type-level programs, how might one try to ensure that they accurately model the problem and do not themselves contain errors?

This thesis explores several approaches to this problem. It presents a tool for generating types and code from diagrams, leveraging human strengths in visual reasoning. Additionally, a type-level property based testing framework is presented, allowing dependently typed stateful models to be tested. Both approaches are evaluated over stateful systems of increasing complexity, with even small systems giving rise to subtle bugs in a dependently typed model. These results demonstrate the value of the tools presented, which combine multiple degrees of verification into a single system and reduce the expertise required to increase the confidence in dependently typed programs while strengthening the relation between the checked model and the implementation.

Functional program synthesis with higher-order functions and recursion schemes

MATHEUS CAMPOS FERNANDES
Universidade Federal do ABC, Brazil

Date: October 2025; Advisor: Fabrício Olivetti de França and Emílio de Camargo Francesquini
URL: https://arxiv.org/abs/2511.23354

Program synthesis is the process of generating a computer program following a set of specifications, such as a set of input-output examples. It can be modeled as a search problem in which the search space is the set of all valid programs. As the search space is vast, brute force is usually not feasible, and search heuristics, such as genetic programming, also have difficulty navigating it without guidance. This text presents two novel genetic programming algorithms that synthesize pure, typed, and functional programs: HOTGP and Origami. HOTGP leverages strong types and a functional grammar to constrain the search space and improve synthesis performance. It synthesizes Haskell code, with support for higher-order functions, λ-functions, and parametric polymorphism. Experimental results show that HOTGP is competitive with the state of the art. Additionally, Origami is an algorithm that tackles the challenge of effectively handling loops and recursion by exploring Recursion Schemes, in which the programs are composed of well-defined templates with only a few parts that need to be synthesized. Preliminary results on a prototype implementation show that, once the choice of which recursion scheme is made, the synthesis process can be simplified. The first implementation of Origami can synthesize solutions in several Recursion Schemes and data structures, improving over its prototype. Results also show that it is competitive with other GP methods in the literature, as well as LLMs. The latest version of Origami employs a novel procedure, called AC/DC, designed to improve the search-space exploration. Experimental results in 3 benchmarks show that it achieves considerable improvement over its previous version by raising success rates on every problem. Compared to similar methods in the literature, it has the highest count of problems solved with success rates of 100%, $\geq$ 75%, and $\geq$ 25% across all benchmarks. In 18% of all benchmark problems, it stands as the only method to reach 100% success rate, being the first known approach to achieve it on any problem in PSB2. It also demonstrates competitive performance to large language models, achieving the highest overall win-rate against Copilot among all GP methods.

Embedding complex languages
in the functional programming vernacular

SAMANTHA FROHLICH
University of Bristol, UK

Date: March 2026; Advisor: Meng Wang
URL: https://samfrohlich.github.io/Content/Papers/thesis.pdf

Programming languages have come a long way since punching holes in cards. Increasingly, they feature powerful but complex language features to meet programming demands of today. For example, bidirectional programming techniques have developed to allow for two related operations to be programmed with one piece of code; and type systems have advanced to ensure the correct programming of quantum computers.

As we develop these powerful tools, it is important to ensure that their complexity is not felt by the programmer. The paradigm of functional programming, with its emphasis on abstraction, provides the perfect vernacular for programming these complexities unburdened of finer details. In particular, the functional technique of embedding, where a new language is created within another, has a lot of potential. It facilitates the quick prototyping of domain specific languages, which are well suited for supporting complexities because they are designed specifically for each domain.

The aim of this thesis is to bring benefits of the functional programming vernacular to complex languages via embedding. In particular, it focusses in on two functional idioms:

Monadic Composition: this is introduced as a new paradigm for bidirectional programming (BX). This thesis uses this paradigm to contribute an exploration of logic programming as a new application domain for BX, and to enhance the power of bigenerators in the domain of property-based testing.

Binders: this thesis contributes a type-safe technique for the embedding of ergonomic binders into languages with complex semantics and type systems.

Interactive Haskell type inference exploration

SHUAI FU
Monash University, Australia

Date: January 2025; Advisor: Tim Dwyer and James Stuckey
URL: https://doi.org/10.26180/28127612.v1

Correctness and robustness of software are vital, especially in high-consequence domains like engineering, finance, and healthcare. Static type checking helps to assert program correctness but presents challenges, such as a steep learning curve to understand and resolve the reported type errors. We propose methods to help programmers interactively explore and debug type errors. We introduce several novel techniques embodied in working software tools, developed and evaluated using human-centered design methods. ChameleonIDE and Goanna are type debugging systems focused on interactive visualization and recommending solutions, respectively. GeckoGraph, a graphical notation for polymorphic types, supports understanding complex function types.

Computational aspects of rewriting in higher-dimensional diagrams

DIANA-MARIA KESSLER
Tallinn University of Technology, Estonia

Date: June 2025; Advisor: Amar Hadzihasanovic and Paweł Sobociński
URL: https://doi.org/10.23658/taltech.47/2025

Higher-dimensional rewriting is founded on the observation that certain rewrite systems correspond to directed cell complexes in different dimensions. This gives us a geometric grip on rewrite systems and rewrite derivations, connecting computational mathematics to higher categories and homotopy theory: rewrite systems are directed cell complexes and rewrite rules are directed homotopies.

In this thesis we adopt a computational perspective and view higher-dimensional rewriting as a model of computation in which an n-dimensional rewrite is a computation on $(n-1)$-dimensional data. Our motivation is the question of whether such a model of computation would be a feasible one.

With this in mind, we start by studying the computational complexity of building diagrams in the setting of diagrammatic sets. One of the main results is a *traversal algorithm* for solving the isomorphism problem in time $O(n^2 \log(n))$. We continue by studying the higher-dimensional subdiagram matching problem for rewritable subdiagrams. We provide an algorithm for solving this problem in arbitrary dimension as well as an upper bound on its running time. In the general case, the problem turns out to be in NP. We further show that under certain acyclicity conditions (that are satisfied by all diagrams of dimension less than or equal to 3), the running time of the algorithm is polynomial in the size of the diagram.

The diagrams mentioned so far correspond to pasting diagrams. We end the thesis with the study of acyclicity conditions in a more general framework of diagrams. We show that under one of the acyclicity conditions that we study for the subdiagram matching problem, the ω-category presented by a diagram shape is freely generated in the sense of polygraphs. We further show that under stronger acyclicity conditions this ω-category is equivalent to the one obtained from an augmented directed chain complex in the sense of Steiner, or consists only of subsets of cells in the diagram.

Propositions as sessions taken apart

WEN KOKKE
University of Edinburgh, Scotland

Date: July 2025; Advisor: Philip Wadler
URL: https://era.ed.ac.uk/items/476bcde1-eda7-40c4-88a4-3019b1eec90d

The foundations of functional programming are built on the λ-calculus, a powerful model of computation whose canonicity is affirmed by its correspondence with intuitionistic logic. The foundations of concurrent computation are built on less firm ground.

There is a wide variety of process calculi. None enjoy the same canonicity as the λ-calculus, nor an exact correspondence with logic. This thesis continues the work towards to towards a canonical foundation for concurrent computation based on Girard's Classical Linear Logic (CLL) and starts from the propositions-as-sessions correspondence proposed by Wadler.

Drawing on the work of Abramsky, Bellin and Scott, Caires and Pfenning, and others, Wadler proposed the process calculus CP, which has an exact correspondence with CLL. Drawing on the work of Honda, Honda et al., Gay and Vasconcelos, and others, Wadler proposed the session-typed functional language GV, which provides a practical foundation for session-typed concurrency in functional languages. Via an operational correspondence, Wadler connects GV and CP, and, thereby, session-types to CLL.

This thesis provides an in-depth study of Wadler's propositions-as-sessions correspondence, highlighting its strengths and repairing some of its shortcomings.

Due to its use of commuting conversions, CP is non-confluent, and its semantics are difficult to realise using the operations of concurrent computation it purports to model. I address this by removing the commuting conversions and show that the resulting process calculus is well-behaved and retains its connection to linear logic.

Due to its lack of a parallel composition operator, CP is ill-suited to an analysis using standard concurrency theory. GV lacked an operational semantics, polymorphism, and replication until these were provided by Lindley and Morris. Unfortunately, their GV has similar problems as CP in its treatment of parallel composition, which complicates its metatheory. I address this by adding a parallel composition operator and show that the resulting calculi are well-behaved and correspond hypersequential CLL.

Finally, I study Dardha and Gay's Priority CP, which increases CP's expressivity at the cost of its compositionality and correspondence to logic. As with CP, I remove the commuting conversions from Priority CP, and complete the correspondence with Priority GV, the analogous variant of GV.

Comparing functional programs, or how to put λ-terms back in order

ADRIENNE LANCELOT
Institut Polytechnique de Paris, France

Date: November 2025; Advisor: Beniamino Accattoli and Giulio Manzonetto
URL: https://theses.hal.science/tel-05407883

Program equivalence is a central topic in the study of programming languages. Certifying that compiler transformations do not change the semantics of a program is needed to formally prove correct compilers.

This thesis aims at further developing the theory of program equivalences for the untyped pure λ-calculus, on several viewpoints.

We suggest to switch from program equivalences to program preorders, which is an usual switch in the study of programming languages but perhaps not something fully accepted at the level of the λ-calculus. We show that some central concepts in the λ-calculus reformulate to simple properties on preorders for λ-terms.

Contextual equivalences are the most natural notion of comparison for programs: contextual comparison is about testing programs in any execution environment and checking that they behave in a similar way. This equivalence is mostly un-usable to provide proof of equivalences for expressive enough programming languages.

We introduce a quantitative refinement of contextual equivalence, trying to obtain a cost-sensitive program equivalence as natural as contextual comparison. Interaction equivalence tests programs in any execution environment, checks that their behavior is similar but also takes into account the number of communications (or interactions) between the program and its environment.

Finally, we explore program equivalences for another evaluation paradigm, namely call-by-value. Call-by-value is more efficient and closer to the actual evaluation mechanism used by programming languages despite it being less studied than the original evaluation of the λ-calculus. It leads to a wide range of program equivalences, usually studied with effects added to the language. We study it here in a pure setting and show that call-by-value is a rich framework with many different notions of equivalences.

We provide a unifying view of all these programs equivalence and their main properties, abstracting over the evaluation mechanisms considered.

Graded modal types for memory and communication safety

DANIELLE MARSHALL
University of Kent, UK

Date: March 2026; Advisor: Dominic Orchard
URL: https://kar.kent.ac.uk/113723/

Using types to delineate various kinds of values is a ubiquitous technique in modern programming. However, one common pattern is for a piece of data to represent access to a resource (such as a reference or channel) which needs to be used in a certain way. This presents challenges for using types to ensure software correctness, as we need to reason about *how* data is used rather than *what* data is being used.

Linear types are one approach to this problem, separating data that can be used any number of times from data that will be used exactly once. Graded type systems such as the one underlying the Granule research language generalise this to allow for more precise restrictions on how data must be used. However, existing graded systems generally focus on local properties pertaining to single pieces of data, whereas sometimes we are interested in global properties that hold over an entire program.

In this thesis, we demonstrate that graded types can be extended to allow for the verification of global program behaviour, encapsulating key properties from the paradigms of both *memory* and *communication* safety.

First, we discuss memory safety. We describe the relationship between linearity and the similar but contrasting idea of uniqueness, which ensures that only a single mutable reference to a value can exist, and embed uniqueness into an existing graded calculus by introducing a new modality. We then extend this system to describe ownership and borrowing as popularised by the Rust programming language, using a new form of fractional grading which allows for multiple immutable references to a single value. The overall system is made formal through a heap-based operational model.

Second, we discuss communication safety. We present a formalisation of session types alongside grading, using an operational model extended with processes and channels to show that concurrency issues are ruled out and resolving problems with Granule's prior instantiation of session-typed channels. We then introduce primitives for non-linear communication, allowing for essential communication patterns that cannot be expressed with purely linear session types to be precisely represented. All systems discussed are fully implemented in Granule.

A generalized algebraic theory of directed equality

JACOB NEUMANN
University of Nottingham, UK

Date: September 2025; Advisor: Thorsten Altenkirch and Graham Hutton
URL: https://jacobneu.phd/final.pdf

We develop a directed type theory capable of synthetic reasoning about 1-categories, with straightforward semantics in the category of categories. All our semantic notions are defined as generalized algebraic theories, permitting modular reasoning about different type theories and ensuring the existence of initial syntax models.

We define the category model of type theory—the directed analogue of Hofmann and Steicher's groupoid model—where contexts are interpreted as categories and types as (split opfibrant) displayed categories. Adapting the groupoid model semantics of key type formers, (dependent) functions and identity types in particular, requires us to adopt a modal typing discipline to track the variances of terms. We define a succession of model notions—polarized categories with families, neutral-polarized categories with families, and directed categories with families—to abstractly capture more and more of the type theory of the category model; this includes an axiomatization of how the groupoid model's undirected type theory is situated inside the category model's directed type theory.

The symmetric (i.e. invertible) identity types of the groupoid model become directed hom-types of the category model; we can show by metatheoretic argument that our polarized typing discipline prevents invertibility of hom-types from being provable in our directed type theory. Each type therefore has the structure of a synthetic category and each function a synthetic functor; this is all proved within the theory using the eliminator for hom-types, directed path induction. We expound a new style of category-theoretic reasoning appropriate to this setting, where the universal mapping properties of standard category theory are all phrased as principles of induction.

To support these developments, we outline generalized algebra, the mathematical discipline concerned with generalized algebraic theories. In particular, we highlight the construction of an initial algebra for every GAT and the fact that every GAT gives rise to a "concrete category with families", a model of type theory whose contexts are algebras and types are displayed algebras. The groupoid and category models can be viewed as a modification of the concrete CwFs of groupoids and categories, respectively, which ensures the fibrancy of their types.

Functional DSL compilers with lightweight proofs

DAVID RICHTER
Technische Universität Darmstadt, Germany

Date: April 2025; Advisor: Mira Mezini
URL: https://doi.org/10.26083/tuprints-00030213

To investigate compilation techniques for high-level programming languages, we present a series of compilers covering a range of language features. This thesis is structured into two parts: (i) compiling and enforcing protocols, and (ii) intermediate representations and optimizations.

We begin the first part, by comparing two emerging trends in programming languages for distributed protocols – choreographic and multitier programming. Choreographic and multitier programming are both global programming models, in the sense that a single program specifies the behavior of all participants in the distributed protocol. To investigate similarities and differences, we present an algorithm to translate a restricted version of choreographic programming to multitier programming, and vice versa. Then, we present our own global language called Prisma, for writing programs that run partially on a smart contract. Besides splitting the source program into a program for the smart contract and a program for its clients, Prisma also enforces the control flow of the contract even in the presence of clients who deviate from the protocol.

In the second part, we investigate intermediate representations and optimizations. In functional compilers, A-normal form is a common intermediate representation, whose design can be derived from the concept of laws of monads. However, monads are usually associated with sequentiality and bringing evaluation into a total order. We present a notation par-seq that compiles to monadic operations to express dependencies between terms, and applicative functor operations to express independent terms. Then, we present the indexed A-normal form (AiNF), an extension of A-normal form for array operations, based on the idea that arrays are dual to functions, and show how it can create optimization opportunities in combination with partial evaluation and common subexpression elimination. Finally, we present the density compiler repot. It takes a probabilistic program as input and then searches for a program to compute the corresponding probability density function to output. A key part of this search is the use of conditional A-normal form (canf) to perform program inversion.

A formal approach to exchange design and regulation

SUNEEL SARSWAT
Tata Institute of Fundamental Research, India

Date: August 2024; Advisor: Raja Natarajan
URL: https://drive.google.com/file/d/1itdaGKMCDp-29sGI45xQNVu8kocuvdeB

Financial exchanges for foreign currency, stocks, and commodities are organized marketplaces that execute trades by matching traders' buy and sell orders. Recent instances of exchanges violating regulatory guidelines and operational rules motivate the need for more robust exchange design and oversight. This work proposes a comprehensive framework for formalizing and certifying double auctions—the key mechanisms exchanges use to match trading orders. We focus on the two primary types of double auctions employed by exchanges: call auctions and continuous auctions, where call auctions further divide into uniform price and dynamic price variants. Our main contributions are as follows. All results are implemented and verified in Coq with OCaml code extracted via Coq's extraction mechanism.

- **Call auctions:** We formalize call auction mechanisms and provide algorithms for uniform and dynamic price auctions with correctness proofs.
- **Continuous auctions:** We specify continuous double auctions through three necessary properties and prove their sufficiency for completely determining the input-output behavior. We then formally verify that a standard matching algorithm satisfies these properties.
- **Uniqueness theorems:** We establish uniqueness theorems for both call and continuous auctions that enable us to build automated checkers guaranteed to detect errors in exchange trade logs if they generate transactions that violate the specifications. We extract verified programs from our formalized algorithms to build these checkers.
- **Efficiency:** We establish tight complexity bounds for all three auction types in the comparison model. For uniform price auctions, we design a linear-time algorithm, improving upon the previous best-known $O(n \log n)$ algorithm for n orders. For dynamic price auctions, we prove an $\Omega(n \log n)$ lower bound, establishing optimality of the best known algorithm. For continuous auctions, we show that our algorithm runs in $O(n \log n)$ time and prove a matching lower bound of $\Omega(n \log n)$.
- **Applicability:** We develop preprocessing steps to instantiate our framework for specific exchange configurations and trading rules. We validate our automated checkers on both real exchange data and synthetic datasets, demonstrating practical feasibility and violation detection capabilities.

On the syntax and the semantics of propositional dependent type theories

MATTEO SPADETTO
University of Leeds, UK

Date: October 2024; Advisor: Nicola Gambino, Federico Olimpieri and Michael Rathjen
URL: https://etheses.whiterose.ac.uk/id/eprint/35802/

This thesis aims to contribute to the theory of *propositional dependent type theories*, also known in the literature as weak, objective, and axiomatic dependent type theories, via syntactic and semantic, specifically category theoretic, approaches. Propositional theories are dependent type theories in which the computation rules consist of propositional equalities, rather than the definitional equalities that characterise reductions and expansions in formal systems like Martin-Löf type theory.

Our first contribution is an analysis of the syntax of theories with identity types, dependent sum types, and dependent product types, in propositional form. Specifically, we show that dependent sum types with the reduction rule in propositional form admit, in the presence of identity types with the reduction rule in propositional form, a negative presentation. In other words, we prove that the elimination rule for dependent sum types and the corresponding reduction in propositional form are equivalent to the condition that the corresponding introduction rule enjoys the *existence property* in propositional form.

Secondly, we examine to what extent the deductive power of propositional theories of dependent types decreases compared to their extensional counterparts. Following an approach devised by Martin Hofmann, we prove a conservativity result for extensional theories over propositional ones, relative to the class of *h-elementary statements*, i.e. statements inductively obtained by quantifying over atomic h-sets. The argument exploits the soundness of the semantics of extensional dependent type theories, phrased using the notion of a category with attributes as a notion of model. Informally, we show that, for judgements essentially concerning h-sets, reasoning with extensional or propositional theories is equivalent.

Finally, we focus on the semantics of propositional dependent type theories from a higher categorical perspective. Adopting Richard Garner's approach in the 2-dimensional study of dependent types, we prove that the type constructors of propositional theories can be encoded into natural 2-dimensional category theoretic data, and use this fact to show that the semantics of propositional theories admits a presentation via 2-categorical models called *display map 2-categories*. This formulation generalises the one provided by Garner for intensional theories. We prove that the interpretation of propositional theories within display map 2-categories is well-defined and enjoys the soundness property.

Trustworthy machine learning for high assurance systems

AMY STELL
University of Exeter, UK

Date: November 2025; Advisor: Achim D. Brucker
URL: https://hdl.handle.net/10779/exe.30610607

Deploying neural networks in safety-critical applications requires mathematical guarantees about their behaviour that cannot be provided through empirical testing alone. While neural networks have demonstrated remarkable performance across diverse domains, their susceptibility to adversarial examples and lack of formal specifications present challenges for high-assurance systems where predictable behaviour is essential. This thesis presents a comprehensive formal neural network verification framework using interactive theorem proving in Isabelle/HOL.

First, we establish a complete formalisation of interval arithmetic for neural networks. Second, we develop three complementary formal representations of neural networks (directed graph, list-based, and matrix-based sequential models), each optimised for different verification scenarios with proven equivalences that enable conversion between the representations. Third, we prove the inclusion isotonicity, interval extension, and Lipschitz continuity of neural network computations.

Our verification methodology provides three approaches. Closed-form verification produces exact mathematical definitions through symbolic execution. Interval analysis provides a scalable approximation with soundness guarantees. Interval refinement combines both approaches through subdivision and refinement, which offers precision that converges to arbitrary accuracy.

Our tooling infrastructure enables a complete workflow of formal machine learning verification. An automated import mechanism translates trained TensorFlow networks into formal representations while automatically establishing various mathematical properties required for verification.

Our approach is demonstrated through different case studies. Verification of a digit recognition network provides a comparative analysis of all three verification approaches, while the ACAS Xu airborne collision avoidance system case study demonstrates scalability to safety-critical applications.

Key contributions include: (1) the first comprehensive neural network verification framework in an interactive theorem prover capable of handling networks trained with standard machine learning frameworks; (2) the development of novel verification methods that allow for formal analysis and reasoning about complex network properties; (3) demonstrated practical applicability through successful verification of safety-critical benchmarks.

Designing a high-level programming language for interaction nets

MARC THATCHER
University of Sussex, UK

Date: December 2025; Advisor: Ian Mackie
URL: https://tinyurl.com/4xdsm2e3

Interaction nets are a computational model that represent programs as two-dimensional networks evaluated according to a system of rules, rather than a linear sequence of instructions. They provide a model of asynchronous parallel processing, but how can programmers utilise this?

Several interaction net programming languages have been proposed, starting with the first paper describing the model, but all are explicit: they describe networks and rules by stating all the nodes and connections involved, one by one. This makes them relatively easy to implement but hard to program with—they are the interaction net equivalent of programming in assembly language.

This thesis proposes an alternative approach: a high-level language with a modern functional syntax that maps directly to a computationally complete subset of interaction nets.

We first introduce function-constructor nets, then a calculus for describing them and algorithms to translate between the calculus and nets. We next establish properties of the calculus and the translations and discuss several extensions to the base calculus. We describe a prototype implementation of the calculus as a programming language.

We then add types, along with an inference system and a reconstruction algorithm which is shown to terminate with the most general type.

Finally, we introduce a novel notion of productivity for interaction nets. For nets that satisfy certain criteria, this notion provides information about termination and deadlock. We also present a decidable method for quantifying and analysing this notion.

A new framework for syntax with binders

JAN VAN BRÜGGE
Heriot-Watt University, Scotland

Date: June 2026; Advisor: Jurriaan Hage and James McKinna
URL: https://jvanbruegge.github.io/phd-thesis/

Nominal Isabelle provides powerful tools for meta-theoretic reasoning about syntax of logics or programming languages, in which variables are bound. It has been instrumental to major verification successes, such as Gödel's incompleteness theorems. However, the existing tooling is not compositional. In particular, it does not support nested recursion, linear binding patterns, or infinitely branching syntax.

It also cannot provide a strengthened induction theorem for inductive definitions like standard β-reduction, as it relies on a syntactic criterion to ensure soundness which rules out many definitions for which such a strengthening would be sound.

Taking advantage of recent theoretical advancements that overcome these limitations through a modular approach using the concept of a Map-Restricted Bounded Natural Functor (MRBNF), we develop and implement a new definitional package for binding-aware datatypes and codatatypes in Isabelle/HOL. We also provide semantic criterion for the strengthening of inductive definitions that is able to capture e.g., standard β-reduction in the untyped λ-calculus without the need to add extra side conditions.

Finally, through the new notion of a Multi-Nominal Monad (MN-Monad), we are able to carry the substitutive structure of datatypes through composition and least fixpoint construction, which enables the automatic definition of substitution operators that compose across types.

Proof theory of semi-substructural logics

CHENG-SYUAN WAN
Tallinn University of Technology, Estonia

Date: June 2025; Advisor: Tarmo Uustalu and Niccolò Veltri
URL: https://doi.org/10.23658/taltech.43/2025

In this thesis, we investigate the proof theory of semi-substructural logics. These are the logical counterparts of skew monoidal categories and their variants. Unlike traditional monoidal categories, skew monoidal categories relax the associativity and unitality laws from natural isomorphisms to natural transformations with specific orientations, making these logics intermediate between non-commutative intuitionistic linear logic or Lambek calculus and their non-associative variants.

We begin with a proof-theoretical analysis of skew monoidal closed categories via their corresponding logic, skew non-commutative multiplicative intuitionistic linear logic (SkNMILL) with the sequent calculus with stoup (LSkG). We prove that LSkG enjoys cut-elimination, Craig interpolation, variations of Maehara interpolation and proof-relevant interpolation. An interesting problem with variants of skew monoidal categories is their coherence problem, i.e. can we have an effective method to determine whether there is a canonical morphism between two objects and whether there can be at most one, or how to enumerate them or how to establish equality of two morphisms if there can be multiple morphisms. The case of skew monoidal categories is more delicate than that of monoidal categories because the relaxation allows for multiple canonical morphisms for some pairs of objects. To solve the coherence problem, we develop a focused sequent calculus with tag annotations that derives just one representative for every equivalence class of derivations in LSkG.

Subsequently, we extend both LSkG and its focused calculus to several variants: a skew commutative extension (ComLSkG), an additive extension (LSkGA) with conjunction and disjunction, and their combinations, each of which corresponds to a specialization of skew monoidal categories.

For semi-substructural logics that cannot be characterized well by the stoup sequent calculus, e.g. skew monoidal bi-closed categories, we develop axiomatic and tree sequent calculi and prove their equivalence. Additionally, we prove soundness and completeness with respect to ternary relational semantics for these logics and the correspondence theorem between structural laws and frame conditions.

These investigations demonstrate, we believe, that semi-substructural logics are interesting and enjoy rich syntactic and semantic properties, beyond their motivation as logical counterparts of variants of skew monoidal categories.

Cut-elimination and normalization for ComLSkG and LSkGA were formalized in Agda.

Reliability and expressivity of bidirectional transformations

RUIFENG XIE
Peking University, China and KU Leuven, Belgium

Date: May 2026; Advisor: Zhenjiang Hu and Tom Schrijvers
URL: https://tinyurl.com/y95ymm49

Bidirectional transformations (BXs, or lenses) are a generic theory for describing the synchronisation between two different data structures. Lens frameworks not only provide practical ways to develop bidirectional programs, but also specify the correctness properties and supports compositional programming via law-preserving combinators.

Despite the theoretical strengths, generic BX theories are not widely used in production. On one hand, existing BX frameworks cannot guarantee the reliability of lenses. It is expected that certain updates cannot be synchronised properly, yet this possibility of synchronisation failure is not formally communicated for these lenses. On the other hand, practical applications often require access to certain effects, such as logging, runtime errors, mutable states, or even unrestricted I/O, but existing BX frameworks are not expressive enough to support liberal use of side effects.

To address the two limitations, we propose two developments of the lens theory: contract lenses and effectful lenses. First, we introduce contracts into lenses to ensure their reliability. These contracts formally specify valid updates that are accepted by the backward transformation of a lens, and the theory makes sure that all valid updates can be properly synchronised. We propose a collection of generic recursion patterns for bidirectional transformations. Second, we introduce monadic effects into lenses to improve their expressivity. Our effectful lenses can have different user-selected effects in their two directions. We define the pure-likeness predicates and use them to generalise the two well-known round-trip properties to effectful lenses, and we prove that composition preserves the round-trip properties. We provide a rich combinator language, which enables compositional programming for effectful lenses. We also present a case study to illustrate the flexibility and expressivity of our framework. Third, we consider a practical synchronisation task, parsing and printing. We develop the theory of biparsers as an instance of effectful lenses, with support for exact-printing. We demonstrate the expressivity of our framework with biparsers for subsets of JSON and YAML as a case study.

Advanced type systems

VÍT ŠEFL
Charles University, Czech Republic

Date: March 2025; Advisor: Tomáš Dvořák
URL: http://hdl.handle.net/20.500.11956/197824

While the results of type systems research are being incorporated into many mainstream programming languages, this process has generally been conservative and slow. Recent research results have the potential to greatly increase the expressiveness of their type systems. This thesis demonstrates the application of three such results in practical programming: the use of a polymorphic lambda calculus to eliminate errors in template metaprograms, the efficiency of zippers obtained through algebraic data type differentiation, and practical uses of additive types in substructural type theories. We further demonstrate the viability of the proposed solutions by defining and implementing two new programming languages, as well as providing a detailed performance analysis of zippers and their imperative counterparts.