JFP **34**, e10, 12 pages, 2024.

doi:10.1017/S0956796824000108

PhD Abstracts

GRAHAM HUTTON
University of 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 eleven 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

A Formal Framework for Understanding Run-Time Checking Errors in Gradually Typed Languages

FELIPE ANDRES BAÑADOS SCHWERTER
The University of British Columbia, Canada

Date: April 2024; Advisor: Ronald Garcia
URL: https://tinyurl.com/5f6ed6zj

Although Abstracting Gradual Typing provides a systematic approach to design gradual languages, the original framework has limitations: first, it accepts design choices that lead to type inconsistencies sneaking through evaluation. Second, when a type inconsistency is identified at run time, evaluation halts without providing any feedback on the parts of the program related to the failure, a safe approach yet unhelpful for debugging.

This dissertation addresses these two limitations of the Abstracting Gradual Typing framework. For the first limitation, I impose an extra constraint on the acceptable designs for gradual types: forward completeness of every type operation. This stricter constraint guarantees that, throughout evaluation, gradual types and runtime evidence objects cannot lose precision and will only represent information consistent with the original static type system. I introduce a new design for a gradual language with record subtyping that fulfills this restriction.

For the second limitation, I provide a specification for runtime program slicing that can be systematically applied to languages designed using Abstracting Gradual Typing. Slicing can separate the portions of a program that are guaranteed to be uninvolved in a runtime failure. Unlike the standard blame approach, slicing does not assume that types are correct. The slicing semantics can be used to provide a debugging tool, and I apply empirical research methods to explore whether this runtime type slicing approach is useful to developers.

https://doi.org/10.1017/S0956796824000108 Published online by Cambridge University Press

Domain Theory in Constructive and Predicative Univalent Foundations

TOM DE JONG
University of Birmingham, UK

Date: February 2023; Advisor: Martín Hötzel Escardó
URL: https://tinyurl.com/mv2xtkck

We develop domain theory in constructive and predicative univalent foundations (also known as homotopy type theory). That we work predicatively means that we do not assume Voevodsky's propositional resizing axioms. Our work is constructive in the sense that we do not rely on excluded middle or the axiom of (countable) choice. Domain theory studies so-called directed complete posets (dcpos) and Scott continuous maps between them and has applications in programming language semantics, higher-type computability and topology. A common approach to deal with size issues in a predicative foundation is to work with information systems, abstract bases or formal topologies rather than dcpos, and approximable relations rather than Scott continuous functions. In our type-theoretic approach, we instead accept that dcpos may be large and work with type universes to account for this. A priori one might expect that complex constructions of dcpos result in a need for ever-increasing universes and are predicatively impossible. We show that such constructions can be carried out in a predicative setting. We illustrate the development with applications in the semantics of programming languages: the soundness and computational adequacy of the Scott model of PCF and Scott's D_∞ model of the untyped λ-calculus. Both of these applications make use of Escardó's and Knapp's type of partial elements. Taking inspiration from work in category theory by Johnstone and Joyal, we also give a predicative account of continuous and algebraic dcpos, and of the related notions of a small basis and its rounded ideal completion. The fact that nontrivial dcpos have large carriers is in fact unavoidable and characteristic of our predicative setting, as we explain in a complementary chapter on the constructive and predicative limitations of univalent foundations. Our account of domain theory in univalent foundations is fully formalised with only a few minor exceptions. The ability of the proof assistant Agda to infer universe levels has been invaluable for our purposes.

https://doi.org/10.1017/S0956796824000108 Published online by Cambridge University Press

Achieving Self-Sustainability in Interactive Graphical Programming Systems

JOEL JAKUBOVIC
University of Kent, UK

Date: March 2024; Advisor: Tomas Petricek and Stefan Marr
URL: https://tinyurl.com/mv2z8e2w

Programming is fraught with accidental complexity. Software, including tools used for programming, is inflexible and hard to adapt to one's specific problem context. Programming tools do not support *Notational Freedom,* so programmers must waste cognitive effort expressing ideas in suboptimal notations. They must also work around problems caused by a reliance on plain text representations instead of *Explicit Structure.*

The idea of a *Self-Sustainable* programming system, open to adaptation by its users, promises a way out of these accidental complexities. However, the principles underlying such a property are poorly documented, as are methods for practically achieving it in harmony with Notational Freedom and Explicit Structure. We trace the causes of this difficulty and use them to inform our construction of a prototype self-sustainable system. By carefully reflecting on the steps involved in our specific case, we provide insight into how self-sustainability can be achieved in general, and thus how a motivated programmer can escape the aforementioned sources of accidental complexity.

https://doi.org/10.1017/S0956796824000108 Published online by Cambridge University Press

Verified Compilation of a Purely Functional Language to a Realistic Machine Semantics

HRUTVIK KANABAR
University of Kent, UK

Date: August 2023; Advisor: Olaf Chitil
URL: https://tinyurl.com/2jzsbdzs

Formal verification of a compiler offers the ultimate understanding of the behaviour of compiled code: a mathematical proof relates the semantics of each output program to that of its corresponding input. Users can rely on the same formally-specified understanding of source-level behaviour as the compiler, so any reasoning about source code applies equally to the machine code which is actually executed. Critically, these guarantees demand faith only in a minimal trusted computing base (TCB). To date, only two general-purpose, end-to-end verified compilers exist: CompCert and CakeML, which compile a C-like and an ML-like language respectively.

In this dissertation, I advance the state of the art in general-purpose, end-to-end compiler verification in two ways. First, I present PureCake, the first such verified compiler for a purely functional, Haskell-like language. Second, I derive the first compiler correctness theorem backed by a realistic machine semantics, that is, an official specification for the Armv8 instruction set architecture.

Both advancements build on CakeML. PureCake extends CakeML's guarantees outwards, using it as an unmodified building block to demonstrate that we can reuse verified compilers as we do unverified ones. The key difference is that reuse of a verified compiler must consider not only its external implementation interface, but also its proof interface: its top-level theorems and TCB. Conversely, a realistic machine semantics for Armv8 strengthens the root of CakeML's trust, reducing its TCB. Now, both CakeML and the hardware it targets share a common understanding of Armv8 behaviour which is derived from the same official sources.

Composing these two advancements fulfils the title of this dissertation: PureCake has an end-to-end correctness theorem which spans from a purely functional, Haskell-like language to a realistic, official machine semantics.

https://doi.org/10.1017/S0956796824000108 Published online by Cambridge University Press

Verified Compilation from BitML to Bitcoin: An Agda Odyssey

ORESTIS MELKONIAN
University of Edinburgh, UK

Date: April 2024; Advisor: Philip Wadler, Aggelos Kiayias and Brian Campbell
URL: https://tinyurl.com/v2njc59f

Blockchain technology has taken the financial world by storm in recent years, allowing for programmable contracts to be enacted amongst participants in a decentralised fashion. Bugs in those programs, however, can lead to huge monetary losses and cannot in principle be amended after detection, due to the blockchain being an immutable data structure.

This incentivizes a high-assurance approach to developing smart contracts, which so far has mainly consisted of approximate methods of static analysis. Here, we strive for something more radical, namely the use of interactive proof assistants grounded in Type Theory to develop such contracts and formally verify their correctness by proving logical propositions within the same system.

Specifically, we take existing work on the *Bitcoin Modelling Language* (BitML) — a high-level process calculus for expressing contracts that compile down to Bitcoin transactions — and encode its definitions, semantics, and translation procedure in the Agda proof assistant.

BitML is one of the most mature works at the confluence of Blockchain and Programming Languages, which justifies the tremendous amount of effort required to mechanise the intricate results of the original paper, compared to various more lightweight alternatives such as model checking.

We can then prove properties about BitML contracts as Agda programs, in particular the main meta-theoretical result of the BitML paper, *compilation correctness*, which states that it suffices to prove properties at the more abstract level of BitML contracts, and then provably transfer them to the low-level of Bitcoin transactions.

By virtue of working in a type-theoretic proof assistant whose underlying logic is *constructive*, we can say that the central research goal of this thesis amounts to producing a *verified compiler* from BitML contracts to Bitcoin transactions.

This whole dissertation is a type-checked Agda script, and the corresponding formalisations are publicly available in HTML format.

https://doi.org/10.1017/S0956796824000108 Published online by Cambridge University Press

Effects and Effect Handlers for Probabilistic Programming

MINH NGUYEN
University of Bristol, UK

Date: December 2023; Advisor: Meng Wang and Roly Perera
URL: https://tinyurl.com/2cmk9fys

Probabilistic programming languages allow programmers to construct statistical models, representing random variables they know and those they wish to learn. Using the same language, the programmer can then simulate data from the model, or apply an inference algorithm to learn the relationships between the model's variables. Although used widely, existing probabilistic languages do not fully support modular and type-safe programming, which has specific impacts on end-users. When modelling, models are either not readily composable, or are restricted to a specific instance of simulation or inference, thus limiting their reusability. Most inference frameworks are then designed without a disciplined approach to side-effects, which can result in monolithic implementations where the structure of the inference algorithms is obscured and programming (customising) them is hard.

This thesis describes a novel approach for designing modular and type-safe probabilistic programming languages, based on *algebraic effects and effect handlers* – a typed functional programming technique for structuring effects. The approach is demonstrated in Haskell as a host language. Part I develops a language for probabilistic models that are modular, first-class, and reusable for both simulation and inference; it shows how these features enable new highly expressive treatment of models, such as composition and higher-orderness. Part II then develops a framework for inference programming that is modular and type-driven, where specific algorithms can be modularly derived from abstract classes of inference algorithms; it illustrates how the approach reveals the algorithms' high-level structure, and makes it possible to tailor and recombine their parts into new variants.

https://doi.org/10.1017/S0956796824000108 Published online by Cambridge University Press

Probabilistic Reasoning in Computation and Simple Type Theory

SIMONA PROKIĆ
University of Novi Sad, Serbia

Date: April 2024; Advisor: Silvia Ghilezan and Zoran Ognjanović
URL: https://tinyurl.com/bdcw5bh9

This thesis investigates two different approaches for probabilistic reasoning in models of computation. The most usual approach is to extend the language of untyped lambda calculus with probabilistic choice operator which results in probabilistic computation. This approach has shown to be very useful and applicable in various fields, e.g. robotics, natural language processing, and machine learning. Another approach is to extend the language of a typed calculus with probability operators and to obtain a framework for probabilistic reasoning about the typed calculus in the style of probability logic. Our contribution has four parts.

First, we study the lazy call-by-name probabilistic lambda calculus extended with let-in operator, and program equivalence in the calculus. Probabilistic applicative bisimilarity has proved to be a suitable tool for proving the context equivalence in probabilistic setting. We prove that the probabilistic applicative bisimilarity is fully abstract with respect to the context equivalence in the probabilistic lambda calculus with let-in operator.

Next, we introduce Kripke-style semantics for the full simply typed combinatory logic, that is, the simply typed combinatory logic extended with product types, sum types, empty type and unit type. The Kripke-style semantics is defined as an extensional Kripke applicative structure, which has special elements corresponding to basic combinators, provided with the valuation of term variables. We prove that the full simply typed combinatory logic is sound and complete with respect to the proposed semantics.

We introduce the logic of combinatory logic, that is, a propositional extension of the simply typed combinatory logic. We prove that the axiomatization of the logic of combinatory logic is sound and strongly complete with respect to the proposed semantics. In addition, we prove that the proposed semantics is the new semantics for the simply typed combinatory logic containing the typing rule that ensures that equal terms inhabit the same type.

Finally, we introduce the probabilistic extension of the logic of combinatory logic. We extend the logic of combinatory logic with probability operators and obtain a framework for probabilistic reasoning about typed combinatory terms. We prove that the given axiomatization is sound and strongly complete with respect to the proposed semantics.

https://doi.org/10.1017/S0956796824000108 Published online by Cambridge University Press

Functional Programming for Securing Cloud and Embedded Environments

ABHIROOP SARKAR
Chalmers University, Sweden

Date: March 2024; Advisor: Mary Sheeran
URL: https://tinyurl.com/bdm6jr2n

The ubiquity of digital systems across all aspects of modern society, while beneficial, has simultaneously exposed a lucrative attack-surface for potential attackers. Consequently, securing digital systems becomes of critical importance. In this dissertation, we address the security concerns of two classes of digital systems: (i) cloud systems, co-locating multiple applications and relying on a large, trusted code base for software virtualisation, and (ii) embedded systems, resource-constrained environments that employ unsafe programming languages for application development.

The thesis underlying our dissertation is that *digital systems can be protected from a wide range of critical attacks by employing functional programming-based techniques, ensuring software isolation in the cloud, and facilitating high-level, declarative and memory-safe abstractions in embedded systems.* Our approach here is to employ functional programming-based techniques, which focus on building software by composing pure functions, avoiding shared state, mutable data, and side-effects, to enhance the security of both cloud and embedded systems. For cloud systems, we use functional programming abstractions to partition security-critical software into compartmentalised structures that use modern hardware protection mechanisms such as Trusted Execution Environments (TEEs) for software isolation. For embedded systems, we present high-level functional programming constructs that raise the level of abstraction and provide safety features to resource-constrained embedded systems. The dissertation is organised into two parts.

Part I introduces a domain-specific language (DSL) designed for programming TEEs, such as Intel SGX, contributing the following: (1) It facilitates automatic type-based program partitioning between trusted and untrusted code, (2) It supports dynamic information flow control mechanisms for ensuring data confidentiality, (3) It integrates with an automated remote attestation framework to preserve TEE integrity, and (4) It offers a tierless programming model that helps minimise errors arising from multi-tier confidential computing applications, requiring adherence to complex data exchange protocols.

Part II contributes a functional language runtime and a functional reactive programming language targeting embedded systems, which allows expressing classical (1) concurrent, (2) I/O-bound, and (3) timing-aware embedded systems applications in a declarative manner.

The programming artifacts resulting from this dissertation are made publicly available, along with the evaluation procedures, encouraging further experiments in securing both cloud and embedded systems.

https://doi.org/10.1017/S0956796824000108 Published online by Cambridge University Press

Bootstrapping Extensionality

FILIPPO SESTINI
University of Nottingham, UK

Date: December 2023; Advisor: Thorsten Altenkirch and Graham Hutton
URL: https://tinyurl.com/5n8eufyz

Intuitionistic type theory is a formal system designed by Per Martin-Loef to be a full-fledged foundation in which to develop constructive mathematics. One particular variant, intensional type theory (ITT), features nice computational properties like decidable type-checking, making it especially suitable for computer implementation. However, as traditionally defined, ITT lacks many vital extensionality principles, such as function extensionality. We would like to extend ITT with the desired extensionality principles while retaining its convenient computational behaviour. To do so, we must first understand the extent of its expressive power, from its strengths to its limitations.

The contents of this thesis are an investigation into intensional type theory, and in particular into its power to express extensional concepts. We begin, in the first part, by developing an extension to the strict setoid model of type theory with a universe of setoids. The model construction is carried out in a minimal intensional type theoretic metatheory, thus providing a way to bootstrap extensionality by "compiling" it down to a few building blocks such as inductive families and proof-irrelevance.

In the second part of the thesis we explore inductive-inductive types (ITTs) and their relation to simpler forms of induction in an intensional setting. We develop a general method to reduce a subclass of infinitary IITs to inductive families, via an encoding that can be expressed in ITT without any extensionality besides proof-irrelevance. Our results contribute to further understand IITs and the expressive power of intensional type theory, and can be of practical use when formalizing mathematics in proof assistants that do not natively support induction-induction.

https://doi.org/10.1017/S0956796824000108 Published online by Cambridge University Press

Parsley: Optimising and Improving Parser Combinators

JAMIE HYDE WILLIS
Imperial College London, UK

Date: March 2024; Advisor: Nicolas Wu
URL: `https://tinyurl.com/w4tyuj7u`

Parser combinators are a functional abstraction for parsing that abstracts hand-written recursive-descent parsers behind a high-level set of combinators. While these kinds of parsers are popular in the functional programming community, they have been historically criticised:

1. Parser combinator performance is sub-par compared with handwritten parsers.
2. The high-level grammar is obscured by the combinators compared with parser generators.
3. The error messages generated by parser combinators are not of fantastic quality.

This dissertation addresses each of these complaints with the work split across two libraries, both called `parsley`: one in Haskell and the other in Scala. Within these libraries, different issues are tackled.

Haskell `parsley` addresses the performance concerns of parser combinators by modelling them as a strongly-typed deep embedding, allowing for optimisations and analysis to be performed. To eliminate the overheads of interpretation and achieve high performance, `parsley` makes use of staged metaprogramming to convert the continuation-passing style automaton into Haskell code resembling hand-written recursive descent parsers; this is faster than contemporary parser combinator libraries in Haskell.

Writing parsers is often an ad-hoc exercise; this dissertation introduces parser combinator design patterns that help structure and standardise how these parsers should be written. These patterns focus on a handful of issues: cleanly handling precedence hierarchies and expression parsing; organising and distinguishing between low-level tokens and higher-level parsing; and abstracting away semantic actions and meta-data processing. This helps to make clean, maintainable, parsers.

Finally, Scala `parsley` has focused on improving the design of parser combinator error systems. The design of this improved system is explored as well as how it can be implemented efficiently, minimising impact on the “happy path.” This gives rise to more parsing patterns as well as enriching existing patterns to incorporate errors. The new patterns help provide the tools to build bespoke, descriptive, errors.

https://doi.org/10.1017/S0956796824000108 Published online by Cambridge University Press

Language-Based Techniques for Policy-Agnostic Oblivious Computation

QIANCHUAN YE
Purdue University, USA

Date: May 2024; Advisor: Benjamin Delaware
URL: https://tinyurl.com/57nva6d7

Protecting personal information is growing increasingly important to the general public, to the point that major tech companies now advertise the privacy features of their products. Despite this, it remains challenging to implement applications that do not leak private information either directly or indirectly, through timing behavior, memory access patterns, or control flow side channels. Existing security and cryptographic techniques such as secure multiparty computation (MPC) provide solutions to privacy-preserving computation, but they can be difficult to use for non-experts and even experts.

This dissertation develops the design, theory and implementation of various language-based techniques that help programmers write privacy-critical applications under a strong threat model. The proposed languages support private structured data, such as trees, that may hide their structural information and complex policies that go beyond whether a particular field of a record is private. More crucially, the approaches described in this dissertation decouple privacy and programmatic concerns, allowing programmers to implement privacy-preserving applications modularly, i.e., to independently develop application logic and independently update and audit privacy policies. Secure-by-construction applications are derived automatically by combining a standard program with a separately specified security policy.

https://doi.org/10.1017/S0956796824000108 Published online by Cambridge University Press