JFP **33**, e6, 7 pages, 2023.

doi:10.1017/S0956796823000035

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 six 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

On the Design of a Gradual Dependently Typed Language for Programming

JOSEPH S. EREMONDI
University of British Columbia, Canada

Date: May 2023; Advisor: Ronald Garcia
URL: https://tinyurl.com/yppdmhz9

Dependently typed programming languages provide a way to write programs, specifications, and correctness proofs using a single language. If a dependent type checker accepts a program, the programmer can be assured that it behaves according to the specification given in its types. However, dependently typed programming languages can be hard to use.

Gradual types provide a way to mix dynamically and statically typed code in a single language. Under this paradigm, programs may have imprecise types, causing certain type checks to be deferred to run time.

We build the theoretical foundations for combining gradual and dependent types in a programming language, with the aim of making dependent types easier to use. The differences between these two paradigms lead to inherent tensions when choosing the properties such a language should satisfy. Gradual typing's effectful nature conflicts with the compile-time reductions of dependent type checking. Gradual run-time type comparisons clash with dependent types containing terms that bind variables. This dissertation identifies such tensions and proposes a design that finds balance between the conflicting goals.

Our contribution has three parts:

First, we present a foundational calculus for gradual dependent types, with functions, function types and universes. To ensure that type checking terminates, we reduce compile-time terms with *approximate normalization*, producing imprecise results when the available type information cannot guarantee termination. We use hereditary substitution to show that approximate normalization always terminates.

Second, we present a notion of propositional equality for gradual dependent types. We devise a method of tracking run-time consistency information between imprecise equated terms, and introduce a composition operator in the language itself.

Third, we show that the first and second contributions can be combined, giving a language with approximate normalization that supports inductive types and propositional equality with dynamic consistency tracking. Since hereditary substitution does not scale to inductive types, we use a syntactic model to establish termination. The same technique is used to model non-terminating run-time semantics using guarded type theory, paving the road for mechanizing the metatheory of gradual dependent types.

A Dependently Typed Programming Language with Dynamic Equality

MARK LEMAY
Boston University, USA

Date: May 2022; Advisor: Hongwei Xi
URL: https://tinyurl.com/mvbmwutj

Dependent types offer a uniform foundation for both proof systems and programming languages. While the proof systems built with dependent types have become relatively popular, dependently typed programming languages are far from mainstream.

One key issue with existing dependently typed languages is the overly conservative definitional equality that programmers are forced to use. When combined with a traditional typing workflow, these systems can be quite challenging and require a large amount of expertise to master.

This thesis explores an alternative workflow and a more liberal handling of equality. Programmers are given warnings that contain the same information as the type errors that would be given by an existing system. Programmers can run these programs optimistically, and they will behave appropriately unless a direct contradiction confirming the warning is found.

This is achieved by localizing equality constraints using a new form of elaboration based on bidirectional type type inference. These local checks, or casts, are given a run-time behavior (similar to those of contracts and monitors). The elaborated terms have a weakened form of type soundness: they will not get stuck without an explicit counter example.

The language explored in this thesis will be a Calculus of Constructions like language with recursion, type-in-type, data types with dependent indexing and pattern matching.

Several meta-theoretic results will be presented. The key result is that the core language, called the cast system, "will not get stuck without a counter example"; a result called cast soundness. A proof of cast soundness is fully worked out for the fragment of the system without user defined data, and a Coq proof is available. Several other properties based on the gradual guarantees of gradual typing are also presented. In the presence of user defined data and pattern matching these properties are conjectured to hold.

A prototype implementation of this work is available.

Twisted Cubes and their Applications in Type Theory

GUN PINYO
University of Nottingham, UK

Date: May 2023; Advisor: Thorsten Altenkirch
URL: https://tinyurl.com/5cyymsd3

This thesis captures the ongoing development of twisted cubes, which is a modification of cubes (in a topological sense) where its homotopy type theory does not require paths or higher paths to be invertible. My original motivation to develop the twisted cubes was to resolve the incompatibility between cubical type theory and directed type theory.

The development of twisted cubes is still in the early stages and the intermediate goal, for now, is to define a twisted cube category and its twisted cubical sets that can be used to construct a potential definition of (infinity, n)-categories.

The intermediate goal above leads me to discover a novel framework that uses graph theory to transform convex polytopes, such as simplices and (standard) cubes, into base categories. Intuitively, an n-dimensional polytope is transformed into a directed graph consists 0-faces (extreme points) of the polytope as its nodes and 1-faces of the polytope as its edges. Then, we define the base category as the full subcategory of the graph category induced by the family of these graphs from all n-dimensional cases.

With this framework, the modification from cubes to twisted cubes can formally be done by reversing some edges of cube graphs. Equivalently, the twisted n-cube graph is the result of a certain endofunctor being applied n times to the singleton graph; this endofunctor (called twisted prism functor) duplicates the input, reverses all edges in the first copy, and then pairwisely links nodes from the first copy to the second copy.

The core feature of a twisted graph is its unique Hamiltonian path, which is useful to prove many properties of twisted cubes. In particular, the reflexive transitive closure of a twisted graph is isomorphic to the simplex graph counterpart, which remarkably suggests that twisted cubes not only relate to (standard) cubes but also simplices.

Computing with Extensionality Principles in Type Theory

LOÏC PUJET
Nantes Université, France

Date: December 2022; Advisor: Nicolas Tabareau
URL: https://tinyurl.com/26dsjvzs

In this thesis, I study several possibilities to extend intuitionistic type theory with extensionality principles, such as function extensionality or Voevodsky's univalence axiom, while preserving the computational properties of the proofs. In the first part, I develop a complete meta-theory for the observational equality of Altenkirch et al. In particular, I obtain a formal proof of normalization, canonicity and decidability of the conversion for an observational type theory with impredicative proof-irrelevant propositions. Then in a second part, I sketch a translation from homotopy type theory to observational type theory based on the model of Coquand et al in cubical sets. Finally, in the last part I explain how to take advantage of the computational properties of cubical type theory to obtain elegant synthetic proofs of classical results from homotopy theory, in particular the construction of the Hopf fibration and the 3×3 lemma for homotopy pushouts.

A Programming Language with Versions

YUDAI TANABE
Tokyo Institute of Technology, Japan

Date: March 2023; Advisor: Hidehiko Masuhara
URL: https://tinyurl.com/ynm5kxrn

While modern software development heavily relies on versioned packages, the concept of versions is rarely supported in the semantics of programming languages, resulting in bulky and unsafe software updates. The dissertation proposes a programming language that intrinsically supports versions. To establish a basis of finer-grained version control in language semantics, the author proposes a language called VL, with core calculus for supporting multiple versions, a compilation method to the core, and an inference algorithm determining the version of each expression. The author proved the type safety of the core calculus to guarantee consistent versions in a program. The author also implements VL, a minimal but adequate functional language that supports data structures and a module system, and conducts a case study involving the simultaneous use of multiple versions.

Iso-Recursive Subtyping: New Theory and Extensions

YAODA ZHOU
University of Hong Kong, Hong Kong

Date: January 2023; Advisor: Bruno Oliveira
URL: https://tinyurl.com/bdewt2zv

The Amber rules are well-known and widely used for subtyping iso-recursive types. They were first briefly and informally introduced in 1985 by Luca Cardelli in a manuscript describing the Amber language. Despite their use over many years, important aspects of the metatheory of the iso-recursive style Amber rules have not been studied in depth or turn out to be quite challenging to formalize.

This dissertation proposes a new theory of iso-recursive subtyping. After revisiting the problem of subtyping iso-recursive types, we introduce a novel declarative specification for Amber-style iso-recursive subtyping. Informally, the specification states that two recursive types are subtypes if all their finite unfoldings are subtypes. With the help of intermediate weakly positive subtyping rules, the Amber rules are shown to be sound and complete with respect to this declarative specification. We then show two variants of sound, complete and decidable algorithmic formulations of subtyping that employ the idea of double unfoldings. Compared to the Amber rules, the double unfolding rules have the advantage of: (1) being modular; (2) not requiring reflexivity to be built-in; (3) leading to an easy proof of transitivity of subtyping; and (4) being easily applicable to subtyping relations that are not antisymmetric. As far as we know, this is the first comprehensive treatment of iso-recursive subtyping dealing with unrestricted recursive types in a theorem prover.

The new formulations not only shed new insights on the theory of subtyping iso-recursive types, but they also enable extensions with more complex features. We show three extensions in this thesis. Firstly, at the type level, we present an extension with record types and intersection types, showing how our new formulations can be applied non-antisymmetric subtyping. Secondly, at the term level, we apply it to a record calculus with merge operators, solving a current open problem for such calculi of how to support recursive types and the binary methods. Finally, we combine iso-recursive types with bounded quantification conservatively, and show that such integration is helpful to encode positive f-bounded polymorphism and subtyping between algebraic datatypes.