Univalent Type Theory, Programming and Verification:

Part 1A: Previous Research & Track Record - Thorsten Altenkirch

See also `http://www.cs.nott.ac.uk/~txa/`

Thorsten Altenkirch received his PhD from the University of Edinburgh in 1993 since October 2006 he has been a Reader in Computer Science at the University of Nottingham and in October 2008 he founded the Functional Programming Laboratory together with Graham Hutton.

Altenkirch is well known for his work on Type Theory and applications of category theory in computer science, and has published over 50 research papers (all available online via google scholar) which are frequently cited (h-index $\geq$ 26). During his work at Nottingham he attracted £1M in research funding, comprising £650,903 as PI in 4 EPSRC grants, £241,075 as CoI in 2 EPSRC grants, £159,038 in 1 fellowship and 1 studentship. Especially relevant for the current project is Observational Equality For Dependently Typed Programming (EP/C512022/1), Theory And Applications of Induction Recursion (EP/G03298X/1), Reusability and Dependent Types (EP/G034109/1). Altenkirch and Ghani have been collaborating on three research grants.

Altenkirch is one of the leading researchers on the emerging subject of Homotopy Type Theory, this is witnessed by his fellowship at the Special Year on Homotopy Type Theory at the Institute of Advanced Study in Spring 2013, where he contributed to the standard reference on the subject [30]. He has having given invited lectures on the subject (in 2012 at HDACT in Lubljana, 2013 at the Curien-fest in Venice, in 2014 at MSC in Lyon and at the IHP in Paris and published a number of papers related to the subject [8, 13, 15, 24].

The Host Institution: University of Nottingham

The School of Computer Science at the University of Nottingham is a research-led School in one of the leading Universities in the UK. The School was ranked 8th in the last Research Assessment Exercise, and the Functional Programming Lab within the School is one of four major research groups, with an international reputation for its work on formally-based approaches to software construction and verification. The FP lab currently comprises 4 academic staff (Thorsten Altenkirch, Venanzio Capretta, Graham Hutton, and Henrik Nilsson) and 9 PhD students. To date the group has received £1.5M of EPSRC funding over 14 projects, and has 12 completed PhD students.

The Functional Programming Lab provides a highly stimulating research environment for researchers and PhD students with weekly research meetings and frequent seminars.

Part 1B: Previous Research & Track Record - Prof Neil Ghani

See also `http://www.cis.strath.ac.uk/~ng`

The Host Institution

The MSP group at the University of Strathclyde is an ideal venue for conducting this research. Dr Johann and Prof Ghani have a well-established, productive collaboration centred on using mathematical structures to guide programming languages research, and are internationally known for their work in λ-calculi, category theory, functional programming, and logical relations. Moreover, Dr Conor McBride's membership in the MSP group ensures local access to a programming perspective that complements their more foundational one, and Epigram, a state-of-the-art dependently-typed programming system of the sort intended to incorporate the results of the proposed research. The MSP group also has eight PhD students and two RAs. The strength of the group has been recognised by the University of Strathclyde, which is just now hiring a new Lecturer for it. Finally, central Scotland is home to vibrant theoretical computer science and functional programming research communities, and the University of Strathclyde is an active participant in the Scottish Informatics and Computer Science Alliance (SICSA). The MSP group also is active in other Scottish meetings, such as ScotCats and SPLS.

Part 2: The Proposed Research and Its Context

1 Introduction

Formal Verification: The cost of software failure is truly staggering[1]. Traditional methods of software verification based upon testing only generate partial guarantees of correctness. Stronger guarantees of software correctness are given by mathematical proofs but the complexity of modern software means that hand written mathematical proofs are untrustworthy. As a result, the only way to ensure truly secure and reliable software is the gold standard of formally verified software, which offers machine-checked mathematical proofs of software correctness. Several decades of pioneering work in the UK and elsewhere have culminated in prototype languages and tools such as Agda, Epigram, Idris and Coq and in the US NuPRL, Twelf, and the Trellys project. These systems are beginning to make their mark, eg Coq has just won both the ACM Software award and the SIGPLAN Programming Languages Software award []. The advanced type-theoretic technology of these systems is raided by more mainstream languages with significant industrial deployment such as Haskell, OCaml, Scala and C#.

The Project: Despite these successes, such systems have a number of short comings in crucial areas, eg when programming with quotient types, supporting abstraction (that is, invariance under different representations of the same structure) and extensional reasoning (proving that programs which behave the same are the same). *Homotopy Type Theory* (HoTT) is widely regarded as a fundamental innovation with great potential to address these and other problems. Delivering this potential requires i) further development of the foundations of HoTT; ii) a programming language and verification environment based on these foundations; and iii) applications demonstrating the effectiveness of this environment to the broader software development community. Therefore we intend to pursue a three-pronged research programme based upon a synthesis of theoretical, applied and impact-focussed research.

- **Theoretical Foundations:** Almost all of our understanding of HoTT exists within a classical framework and hence cannot be used to develop programming language and verification tools. Nevertheless, the recent work by Coquand et al. strongly suggests a constructive presentation of HoTT is possible. We will develop both specific constructive models and a general constructive model theory of HoTT, and then complement these models with type theoretic presentations of them.
- **Programming Languages and Verification:** The key deliverable of the second strand of our research will be a programming language which simultaneously acts as a verification environment based upon HoTT. This is likely to become a major step forward in programming languages design and influence the development of all current and future systems in this area.
- **Generating Impact:** Developing new and fundamentally better ways to construct formally verified software is not just an end in itself, but is also a key prerequisite for engaging others do do so. To help ensure uptake of our research by the wider community, we will produce a number of case studies which will allow users to experiment with our results; at the same time, their practical experiences with it will feed back into our research.

Calibre and Ambition: The foundational nature of HoTT, the consequent potential for solving key open problems in programming languages and formal verification research, and the resulting potential applications to software correctness attest to the quality and calibre of this project. Our ambition is demonstrated by the breadth of our central belief that HoTT is not just a mathematical foundation for type theory, but can also be turned into a programming language and verification environment which - in its treatment of quotients, representational invariance and equality - will become the benchmark standard which current and future systems will seek to emulate.

[1] see the section on National Importance.

2 Scientific and Technological Background:

Programming Languages: Abstraction is essential in programming where identifying common structure is needed to ensure code is clear, clean and concise. This has lead to the development of high level programming languages with expressive type systems capable of closing the *semantic gap* between what programmers know about computational entities and what their types can express about them. The current state of the art are the *dependently typed programming languages* such as Agda, Epigram, Idris and Coq where the programmer can express within the type of his/her program a continuum of precision from basic assertions up to a complete specification - about a programs behaviour. This proposal seeks to become the first of a new breed of univalent dependently typed programming languages which advance the state of the art by offering a powerful, yet computationally tractable, equality and thereby bring to reality the goal of programming up to invariance of representation.

Programme Verification: While the advantages of the certainty afforded by mathematical proof has been recognised for centuries, it has also been recognised that this certainty is undermined by the capacity for humans to make mistakes in their proofs. The advent of computers raised the possibility once more of achieving in practice the promise of mathematical certainty. This potential is now coming to fruition, eg systems such as Coq have been able to formally verify both large mathematical theorems such as the 4-Colour problem, and large software systems such as the CompCert C-compiler However, these systems are not *extensional* in that just because one can prove that objects are behaviourally indistinguishable, one cannot conclude that they are the same. This is a fundamental problem as it weakens the power of the verification system. Our research will address this issue by producing a formal verification system where objects with the same behaviour can be proved equal thereby advancing the state of the art here.

Type Theory: Underlying both formal verification systems and programming languages is the subject of type theory which grew out of Russell's attempts to deal with paradoxes in naive set theory. A major conceptual understanding afforded by type theory was the Curry-Howard correspondence which observed that programs and proofs are actually the same thing, eg proofs are just particular forms of programs. As a result, by developing sophisticated type theories, we advance both the field of programming languages and the field of program verification.

A major step forward within type theory was achieved by Martin-Löf who realised that type theory needed to be extended to cover equality within the system and he did this by introducing the intensional identity type in Martin-Löf Type Theory (MLTT). However, it soon became apparent that this notion of equality type was too weak, eg functions that are pointwise equal cannot be proven to be equal. To address this, Martin-Löf then introduced extensional MLTT which produced a strong equality but at the price of losing decidability of type checking. However, the problem of a strong but decidable theory of equality remained fundamentally unresolved for 40 years.

Logical Relations: Coquand's model is closely reated to parametricity because we can view the interpretation in cubical sets as defining dependently typed logical relations. Parametricity itself is a fundamental technology within computer science and, research into it is currently being funded by EPSRC at Strathclyde. We will use the expertise at Strathclyde to see if variants of the standard presentations of parametricity lead to better behaved refinements of the cubical set model, and feed innovations in the use of parametricity within HoTT back into the general parametricity community. **Needs Repair/rewrite**

Observational Type Theory (OTT) is a step forward, proposed by Altenkirch and McBride [13] in 2007. Its propositional equality is designed to fit the structure of types, and is hence extensional for functions, but its definitional equality remains decidable. However, equality of types themselves is rigidly structural, not up to isomorphism, allowing a proof-irrelevant implementation but precluding the exploitation of higher-dimensional structure. As we advance to a proof-relevant treatment of a more refined equality, OTT offers key insight and techniques to combine a strong equality with decidable type checking.

Homotopy Type Theory (HoTT) is a revolutionary new analysis of equality based on intuitions from homotopy theory. Types are *spaces*; terms are *elements* within a space; the equality type is the space of *paths* in a space. The core of HoTT is the new Axiom of Univalence, introduced by Fields medalist Vladimir Voevodsky, asserting (roughly) that isomorphic types are equal. Crucially, equality proofs carry the data

necessary to program and reason up to invariance of representation. HoTT subsumes function extensionality but also transcends what we could previously express, e.g. *higher* inductive types (HITs) include the usual tree structures, but also quotients and geometric objects such as the torus, where the hole stops us deforming some paths into others. Such paths contain non-trivial structural information, necessitating a proof-relevant equality. The same phenomenon occurs in computer science, e.g. lists are an inductive type; braids are lists with extra paths identifying lists upto twisting of any element past its neighbours, and bags are braids with paths-between-paths identifying braidings which yield the same permutation. **general quotient containers [5, ?]**

Coquand's *cubical* model of HoTT interprets closed expressions as canonical denotations, demonstrating basic feasibility of computing with univalence. However, much work lies ahead: we need to give closed expressions normal forms *within* HoTT, and to recover definitional equalities lost by the cubical interpretation. Further, other models might offer better computational foundations for HoTT, so practical progress towards programming languages and verification tools based on HoTT is best informed by a broader model theory.

3 Methodology and Research Programme:

Our general methodology to the development of HoTT-based software construction and verification will be to harness ideas from a number of different sources: i) the evolving state of the art, e.g. the HoTT book and the cubical sets construction of Coquand et al. as well as the ongoing dialogue with our collaborators; ii) our work on OTT which is a proof-irrelevant prototype of what we seek; iii) our work on the implementation of dependently typed programming languages (Epigram and ΠΣ [10, 19]); iv) our work on datatypes (containers, indexed containers, induction-induction and induction-recursion) [1, 4, 2, 6, 5, 3, 29, 27, 28, 26, 14] ; v) our work on parametricity; and vi) our work on constructing internal models of type theory. These multiple sources will ensure that we neither slavishly follow the hype that inevitably surrounds significant innovation, nor are unaware of current and future advances in HoTT. We have divided the project into the following work packages with WP1 hosted in Leeds, WP2-4 in Nottingham and WP5-8 in Strathclyde. Of course the reality is that we will continue our established practice of working closely together.

WP1: Semantic Foundations of HoTT: We seek semantic insights into HoTT akin to those provided by Cartesian closed categories for the simply typed λ-calculus. A constructive model theory for HoTT is essential because: i) specific implementations of HoTT can be proven sound by giving a specific models of them; ii) a general model theory of HoTT will implicitly predict, and thereby guide, the design space of different presentations and implementations of HoTT; and iii) models of HoTT will provide algebraic techniques to reason about the correctness of implementations which complement syntactic techniques. These models need to be constructive so that programs, even those using Univalence, will compute. While the standard model of HoTT based upon simplicial sets is not constructive, Coquand et al.'s recent cubical set model is constructive and has thus opened the door to a more general model theory.

We will attack this problem from the following directions: i) we will analyse existing models (cubical sets, groupoids, strict ω-groupoids, simplicial sets, globular sets) to isolate exactly how they ensure constructivity, or (where they fail to do so) how to constructivize them; and ii) informed by i), we will develop a model theory for HoTT by both adapting known methods to define Quillen model structures (such as the small object argument) to the constructive setting and by showing how one can build new constructive model structures from old (eg by slicing). A promising starting point for a constructive version of the small object argument comes from Garner's work, where it is related to the construction of free monads. In terms of risk, i) is certainly achievable since it involves only the analysis of existing concrete models, while ii) is a more ambitious goal. Nevertheless, our expertise on semantic models of parametricity (Ghani), model categories (Gambino) and models of type theory (Altenkirch) makes even this ambitious goal feasible. Deliverables from WP1 will be a broad class of models of HoTT which considerably deepen our understanding and map out the design space of its syntactic presentations.

WP2: Univalent Type Theory: Building on WP1, we need syntactic presentations of our models of HoTT in the form of type theories. The challenge is to present the essential data of the model as built from a *finite* collection of type and term constructors - this is particularly difficult given the *arbitrary* higher dimensional

structure of HoTT. One also must prove essential properties such as strong normalization, decidability of definitional equality and canonicity (i.e. all terms reduce to values). It is of particular interest to establish the expressive power of the associated equational theories, e.g. to distinguish carefully between which computation rules hold as definitional equalities and which as propositional equalities. Another key property (required for WP5) is that, as a foundational theory, our type theory ought to be expressive enough to describe its own models. These properties will be established either directly or via the models of WP1.

Cubical sets share structure with logical relations, as Altenkirch has recently observed. We shall exploit this connection to replace the uniform identity type of intensional MLTT with a higher-dimensional equality defined to fit the structure of types. The models from WP1 will drive refinement of our design, until we have a canonical presentation of HoTT. Our preparatory work, and prior expertise in OTT [13], normalisation by evaluation and big-step reduction [11, 12, 16, 9], strengthening definitional equality [7], and logical relations [17] ensures a high probability of delivering an effective presentation of HoTT. Our more audacious goal is to find the generic internal language of ∞-LCCCs in the same way that extensional MLTT is the internal language of LCCCs. *Deliverable: a type theory to underpin our programming language and formal verification environment.* **introduce ∞-LCCC**

WP3: Higher Inductive Types: Our goal is to accomodate HITs in the semantics developed in WP1 and the syntactic framework of WP2. Our idea to achieve this is to develop a universal HIT playing the role for HITs that W-types play for ordinary inductive types. This is feasible as partial progress has already been made ... one can often reduce HITs with higher dimensional constructors to ones with only 0- and 1-dimensional constructors (using the hubs-and-spoke construction). Similarly, [22, 23] has shown how quotient containers can be reduced to ordinary containers in a homotopical setting. A secondary goal is to generate a high-level level syntax for HITs as an alternative to the universal HIT in the same way that strictly positive types provide a grammar for defining various W-types. This will feed into WP7. Our most ambitious goal is to - time permitting - investigate other variations of HITs: coinductive HITs, mixed inductive/coinductive HITS [21], HITs and inductive-inductive and inductive-recursive HITs which open the door towards a more concise representation of dependently typed syntax in type theory [18] by introducing constructors and definitional equalities at the same time. We will also investigate a pattern matching syntax for HITs this is related again to WP7. **More on QCs**

The risk within the first phase of this WP seems relatively low since we have already a good background from our previous work on data types [**?**] including EPSRC grants on Containers and Induction Recursion and because partial results already exist showing that results are available and also guiding the way towards fresh ones. While its not clear we can push the results as far as we imagine with respect to higher inductive recursive types etc, these results are not essential to the rest of the project.

WP4: Programming with Effects: Correctness via Types Most programs interact with their environments, e.g., to read and/or write to the memory and to detect and respond to errors such as attempts to divide by zero or to open files that dont exist. Such programs are called effectful programs and are known to be inherently difficult to reason about because, for example, the result of a program might depend upon the evaluation order. One major advance was Eugenio Moggis idea that effects can be modelled semantically by monads and Wadler later showed that monads could internalised as syntactic sugar to structure effectful programs themselves, eg via the `do`-notation of Haskell. More recently, Plotkin and Power extended this analysis using Lawvere Theories to show how (all most all) computational monads arise from effect-generating operations and equations which these operations satisfy.

Unfortunately, in general, we cannot represent equational theories in current programming languages such as Haskell. Therefore one is often forced to program not using the quotient algebra as desired but using the free algebra and then check - externally to the program - that the program respects the quotient structure. Of course HoTT, can help us formally verify this. But HITs allow us to go further and give an inductive presentation of such quotient structures opening the way to program directly on the quotient algebra and assert the correctness of the program via type checking. Not only is this a particularly efficient form of formal verification, but the correctness of this program can then be used to validate the preconditions of other programs. Executing this research program means formalising both Lawvere Theories in HoTT (using HITs to represent effectful computations) and also the mathematical algebra of Lawvere theories such as tensor products which can be used to combine them. In doing this we will set ourselves the concrete goals of both simplifying and extending

i) McBride and Andjelkovic's work on Frank from free monads to Lawvere Theories; ii) Brady's effects library for Idris; and iii) Bauer and Pretnar's treatments of effects in his programming language Eff. We will do this both within the type theory of WP2 and reflect progress into the programming language of WP7. This work package will therefore act as both validation for the theoretical research done by RA1, and also generate impact by showing how that work can be used to tackle a major programming languages problem. Basic results should be low risk as the fundamental ideas of treating effects via algebraic theories and treating algebraic theories via HITs are established in principle. However, more advanced effects such as indexed-effects which arise in dependently typed programming, or a full-scale integration of our results any in the language of WP7 will be more challenging.

WP5 Formalisation of Meta-Theory of HoTT. As we intend to use HoTT as a formal verification system, we need the highest possible level of trust in its correctness. This means we need to check that the model theory of HoTT, the associated type theory and their relationship are correct. Given that HoTT is inherently a complex and combinatorially intricate mathematical subject because of its higher dimensional structure, a purely paper based verification would be doubtful. Thus, the only way to ensure the required high level of trust in the correctness of our work is to formally verify that this is the case.

We will begin with the relatively low risk task of formalising the cubical sets model of HoTT, our nascent cubical type theory and their relationship in Agda. However this approach will be unsatisfactory in the long run because of the limitations of Agda and hence we will switch to formal verification in our HoTT-based type theory and formal verification system as they are developed. Our belief is that formal verification in HoTT will actually be easier because the syntax of Type Theory will be more efficiently formalised as a HIT which represents not just the type and term constructors but also the associated definitional equality. This is low risk as there is already some formalisation of HoTT in Agda, and more generally, because we have significant expertise in both techniques (such as induction recursion and induction induction) required formalise type theory and also in the verification of properties of the formalisation []. As the project progresses we will consider the more ambitious goals and risky goal of formally verifying properties of the core programming language developed in WP6. At the end of the work package we will have deliverables consisting of formal verification of the key properties of HoTT. Not only will this ensure the required level of trust in our system, but this will have a significant impact upon the formal verification community as it will be the first instance of formal verification in a *HoTT-based* formal verification environment. [2]. **Agda + non-computational univalence is not HoTT!**

WP6: Implementing a Core Programming Language: HoTT in Agda: In order showcase the potential for HoTT based programming languages to the wider programming languages community, and learn from their feedback, we need to build a prototypical implementation of a type checker and interpreter based on the type theory designed in WP2. This system forms a proof of concept implementation lacking most if not all bells and whistles present in modern implementations of Type Theory. That is, while we will implement the type theory of WP2, we will not attempt to implement a high-level syntax for datatypes, universe polymorphism, implicit arguments or pattern matching. However, the language will include a universal HIT based on the work in WP3.

We are planning to use Haskell as an implementation language because there is considerable expertise at Nottingham and Strathclyde in using Haskell to implement type checkers for dependently typed languages including Epigram, ΠΣ and Agda [**?**, 25, 10]. In addition, our prototypical implementation of OTT will be particularly informative as it can be viewed as a precursor of HoTT as it has an recursively defined notion of equality.

Coquand's and Huber's implementation of the cubical set model shows that our approach is feasible in principle and we plan to collaborate with them. However, we should be able to address particular issues such as the fact that certain definitional equalities don't hold in the cubical set model using the technology we have developed in the context of OTT [13] which is the subject of current work at Strathclyde.

The main difficulty within this WP is that the implementation of higher dimensional type theories is a new area — nevertheless our experience in the implementation of dependent type theories leads us to believe this is still of low risk. Having said that, there is some risk that the implementation takes much longer than expected: we will manage this by keeping the scope of the language small. **Play up OTT and Parametricity (Johann)**

WP7: A High Level Programming Language: The aim of this WP is to make the language developed in the

[2] as opposed to formal verification in current systems such as Agda and Coq

previous WPs usable in practice. This relies on a high-level syntax for datatypes including higher inductive types and on integrating known technology such as implicit arguments but also compilation and interfaces to other languages. A central question which needs to be answered is wether it is preferable to integrate our ideas with an existing system such as Agda, Coq or Idris or wether it is preferable to start from scratch. The advantages of the former approach is that we connect with significant user communities, can learn from their experience and avoid duplication of work. However, it is currently not clear wether this is feasible since it would affect the very core of these systems. Either way, we plan to collaborate closely with the developers of these systems to maximise compatibility and impact.

The technical challenges we face are to restrict pattern matching so that it is compatible with HoTT — we plan to build on recent work by Coeckx [20]. There are other issues related to the termination checker which need to be adressed [**?**]. We also want to integrate pattern matching with HITs — this is currently an open problem. **The deliverables with consist of language extensions and some programs**

This is the most ambitious of our work packages because, if successful, we would have produced a new state of the art programming language for software construction and verification. Given the current interest in HoTT, it would immediately attract attention of significant numbers. However, the volume of work required to develop a practical language makes this also the most risky WP. Nevertheless, if all we produce are proof-of-concept implementations of some high level features, leaving significant amounts of the implementation of a practical language to future work, then the project as a whole will still be a massive success because both the foundational, practical and engineering groundwork for a homotopical programming language will have been done. **Libraries for doing HoTT in Agda**

WP8: Generating Impact Through Case Studies: The ultimate goal of the proposed research is to support software construction and verification via HoTT. Our final work package comes full circle to our original motivation: we will apply our programming language to a number of real-world programming problems thereby demonstrating the impact that HoTT can have. By abstracting recurring and effective patterns that arise, we will also develop new methodology to complement the new expressivity of programming in HoTT.

Firstly, HoTT's ability to "work upto isomorphism" opens the way to program correctly using simple and straightforward reference presentations of data structures and then replace these presentations with more efficient equivalents at runtime. For instance, we shall deliver treatments of *numbers* (simple unary representations $\cong$ efficient binary representations), *sequences* (cons-lists $\cong$ finger-trees) and *matrices* (vector-of-vectors $\cong$ sparse encodings). A similar phenomenon occurs with data structures which store redundant information to improve access time, e.g., databases with indices to cut search, and records with cached values to avoid recomputation. **ornaments**. The technical challenge will be to ensure that the efficiency savings are not dominated by the cost of computing with isomorphisms at runtime. Our idea is to use fusion to minimise the number of isomorphisms present at runtime, and to enable the compiler to work with intermediate representations to give fine grain control of the cost of the isomorphisms involved. This will ensure that we maximize the regions within which we use the efficient representations, converting data only at the boundaries. In effect, we will have improved on *data abstraction*, the state-of-the-art tool for managing the craft of implementation, by supporting the refinement of concrete computational models of data.

Secondly, HoTT's richer notion of equality ensures that different representations of a value can be exploited for efficiency purposes but cannot yield inconsistency. For example, whilst treelike structures can be given a canonical form, data such as individual graphs, cycles and multisets often have multiple representatives which should be treated the same by operations. Today's technology presents the dilemma of whether to expose the representation and risk inconsistent treatment or to hide behind an abstraction barrier which offers a fixed repertoire of consistent operations but inhibits us from exploiting the representation to develop unforeseen operations efficiently. At last HoTT offers us a precise deal: we can work with representatives, but we must work up to equality.

4 Quality, Management, and Planning

Relevance to Beneficiaries: This, perhaps more than many projects, is an ambitious project which has the potential to have a significant impact on a large number of researchers. Theoretical computer scientists,

e.g. category theorists, type theorists and logicians will be interested in the fundamental nature of Homotopy Type Theory. Further, programmers will be interested because of the critical mass of programming language examples that Homotopy Type Theory can be applied to — they will particularly appreciate both the languages we develop and the code we will provide and verify. The potential impact of this research can also be gauged by the adventure, timeliness and novelty which we now discuss.

Calibre, Ambition, and Adventure: The potential of HoTT to solve one of the deepest problems in programming languages and verification, and the quality of our ideas for brining this potential to fruition all attest to the calibre of the proposed research. Our ambition is demonstrated by our desire to transform HoTT from an idea with potential to a benchmark in the development of programming languages. The proposed research certainly is not incremental! The adventurous nature of the proposed research is demonstrated by its scope, which ranges from fundamental research (WP1-WP3) to programming languages and verification (WP5-WP7) to impact generation via case studies (WP4 and WP 8). The proposed research is also

...timely: This is an extremely timely moment to embark on the proposed research. Not only do we have our own results to draw on, but there have also been significant recent advances in directly related areas, such as Coquand's cubical model of HoTT. Moreover, programming languages have advanced to the stage where, for the first time, the results of this research both the implemented on their own and also feed into existing languages.

...novel: Fill In: Our goal of turning cutting-edge developments in type theory straight into state-of-the-art programming language and verification techniques distinguishes our approach to HoTT from many others which focus on one or the other. We take this as our goal because we believe that severing the link between foundational understanding and practical application diminishes both.

National Importance: The software market is estimated at $500 billion per year, and this figure is likely to grow significantly in real terms as software becomes ever more ubiquitous. It is thus essential to the UK's national interest to have a strong presence in this market. One crucial aspect of software is that it is correct, i.e., does what's intended and does not go wrong. Even failures of everyday devices like iPods and mobile phones are inconvenient, but software leaking voting records, compromising the global financial sector, or launching nuclear weapons without authorisation can lead to unprecedented and clearly unacceptable global uncertainties.

While testing of programs has dominated the last 50 years of software development, the next 50 years are likely to see an increasing demand for provably correct software. This is partly because testing is by its very nature only a partial guarantee, and partly because programming language technology is finally advancing to the stage where it is feasible to formally verify critical programs. Both programming languages and program verification are identified in EPSRC's portfolio as areas of vital importance for cybersecurity, and EPSRC thus intends to grow them. This proposal uses ideas from mathematics to enhance programming languages and program verification, so lies squarely in their intersection. The UK is a world leader in these areas, but continued investment is required to maintain that status in a rapidly changing world and a rapidly evolving field.

Within programming languages and program verification, we are aiming high. The current state of the art in programming language design is limited by the lack of a clear understanding of how strong equality ought to be within a programming language. Our research will provide a step change in perogramming languages research where the current ad-hoc treatments of equality wiull be replaced by one with a well understood foundation. Thus we expect the results of our research to become the cornerstone for the next generation of high level programming languages cited by both theoreticians and practitioners well into the future. If successful, the proposed research can be expected to have great impact on programming languages and program verification over the next 10 to 50 years, and perhaps even beyond.

Feasibility: We are *uniquely well-positioned* to conduct the proposed research. Drs Gambino and Altenkirch both attended the IAS Special Year on Univalent Foundations in Princeton and hace published influential works on HoTT. Prof Ghani is an expert on parametricity (he is currently PI on an EPSRC grant on parametricity) and Dr McBride is a world expert on programming languages. All members of the team have significant experience in the key areas of category theory, type theory and programming languages which are the three pillars upon which this project is built.

Success criteria: The success of the foundational phase of the proposed research will be demonstrated by

developing the syntax (WP2) and semantics (WP1) of a type theory (programming languages?) based upon the principles of HoTT (validating univalence), extending this to cover HITs (WP3) and verifying the required properties of the language. The success of the languages and verification phase will be demonstrated by the implementation of a programming language (WP 5, 6, 7) whose correctness has been formally verified and within which - for the first time - HoTT programs can be run. Finally, the success of the impact phase will be demonstrated by developing proof-of-concept applications of our results to problems of interest to the wider programming languages and program verification community.

Management and Planning: The rest was for a previous grant. IGNORE ... NG to rewrite: Work on WP1 precedes work on the other work packages because it develops the fundamental techniques to be extended, applied, and implemented. Work on WP2 precedes work on WP3 since morphisms between Lawvere fibrations are needed to establish universal properties of constructions on logical relations. WP4, WP5, and WP6 are independent, but WP1, WP2, and WP3 all feed into them. WP7 will be integrated with the others to the greatest extent possible by starting work on it as soon as results from WP1 are available. Risk *between* work packages is minimised since WP4, WP5, WP6, and WP7 can influence one another, but lack of progress on one will not inhibit progress on others. WP1, WP2, and WP3 are relatively risk free: they build on our previous work, and we already have ideas how to proceed and fallback positions should our first approaches fail. Risk *within* work packages is minimised by applying the right expertise to each. WP1, WP2, and WP3 will be led by Dr Johann and Prof Ghani; Prof Ghani will lead WP4; and Dr Johann will lead WP5 and WP6. The RA will focus on WP7 but may contribute to other work packages as well if they are able.

The RA and Their Training: We will seek two RAs: one with expertise in some of category theory, type theory, semantics of programming languages; and anopther with epxerience in the implementation of functional programming languages and software development and formal proof within modern systems such as Agda and Coq. This is feasible: we know of several highly-qualified researchers — e.g., Fredrik Forsberg, Clément Fumex, Barbara Petit, and Noam Zeilberger — due to finish PhDs or postdoctoral positions within the next year, and we will advertise to recruit the best RA possible. FOP group and MSP group meetings provide opportunities to report on research progress and generate new ideas, and will help integrate the RA into the project. Our reading group for discussing research papers related to the project will also help train the RA. The RA will have the opportunity to write papers and grant proposals, lead research, and help mentor PhD students. At the end of the project the RA will possess highly-desirable knowledge and skills, and be well-positioned to lead future research and/or development efforts. This is important since there is more work to be done in the research and development of next generation programming languages than the active workforce can handle. Overall, the proposed project will have a high impact in terms of training.

Collaboration: In carrying out the proposed research we will collaborate with internationally leading researchers so as to maximise its potential for impact. See our *Pathways to Impact* statement for details.

References

[1] Michael Abbott, Thorsten Altenkirch, and Neil Ghani. Categories of containers. In *Proceedings of Foundations of Software Science and Computation Structures*, 2003.

[2] Michael Abbott, Thorsten Altenkirch, and Neil Ghani. Representing nested inductive types using W-types. In *Automata, Languages and Programming, 31st International Colloqium (ICALP)*, pages 59 – 71, 2004.

[3] Michael Abbott, Thorsten Altenkirch, and Neil Ghani. Containers - constructing strictly positive types. *Theoretical Computer Science*, 342:3–27, September 2005. Applied Semantics: Selected Topics.

[4] Michael Abbott, Thorsten Altenkirch, Neil Ghani, and Conor McBride. Derivatives of containers. In *Typed Lambda Calculi and Applications, TLCA*, 2003.

[5] Michael Abbott, Thorsten Altenkirch, Neil Ghani, and Conor McBride. Constructing polymorphic programs with quotient types. In *7th International Conference on Mathematics of Program Construction (MPC 2004)*, 2004.

[6] Michael Abbott, Thorsten Altenkirch, Neil Ghani, and Conor McBride. ∂ for data. *Fundamentae Informatica*, 65(1,2):1 – 28, March 2005. Special Issue on Typed Lambda Calculi and Applications 2003.

[7] Guillaume Allais, Conor McBride, and Pierre Boutillier. New equations for neutral terms: A sound and complete decision procedure, formalized. In *Proceedings of the 2013 ACM SIGPLAN Workshop on Dependently-typed Programming*, DTP '13, pages 13–24, New York, NY, USA, 2013. ACM.

[8] Thorsten Altenkirch. Extensional equality in intensional type theory. In *In LICS 99*, pages 412–420. IEEE Computer Society Press, 1999.

[9] Thorsten Altenkirch and James Chapman. Big-step normalisation. *Journal of Functional Programming*, 19(3-4):311–333, 2009.

[10] Thorsten Altenkirch, Nils Anders Danielsson, Andres Löh, and Nicolas Oury. ΠΣ: Dependent types without the sugar. *Functional and Logic Programming*, pages 40–55, 2010.

[11] Thorsten Altenkirch, Martin Hofmann, and Thomas Streicher. Categorical reconstruction of a reduction free normalization proof. In David Pitt, David E. Rydeheard, and Peter Johnstone, editors, *Category Theory and Computer Science*, LNCS 953, pages 182–199, 1995.

[12] Thorsten Altenkirch, Martin Hofmann, and Thomas Streicher. Reduction-free normalisation for a polymorphic system. In *11th Annual IEEE Symposium on Logic in Computer Science*, pages 98–106, 1996.

[13] Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. Observational equality, now! In *PLPV '07: Proceedings of the 2007 workshop on Programming languages meets program verification*, pages 57–68, New York, NY, USA, 2007. ACM.

[14] Thorsten Altenkirch, Peter Morris, Fredrik Nordvall Forsberg, and Anton Setzer. A categorical semantics for inductive-inductive definitions. In Andrea Corradini, Bartek Klin, and Corina Cirstea, editors, *CALCO 2011: Fourth International Conference on Algebra and Coalgebra in Computer Science*, volume 6859 of *Lecture Notes in Computer Science*, pages 70 – 84. Springer, 2011.

[15] Thorsten Altenkirch and Ondřej Rypáček. A syntactical approach to weak omega-groupoids. In *Computer Science Logic (CSL'12)*, pages 16–30, 2012.

[16] Thorsten Altenkirch and Tarmo Uustalu. Normalization by evaluation for $\lambda^{\to 2}$. In *Functional and Logic Programming*, number 2998 in LNCS, pages 260 – 275, 2004.

[17] Robert Atkey, Neil Ghani, and Patricia Johann. A relationally parametric model of dependent type theory. In *Proceedings of the 41st annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 503–516. ACM, 2014.

[18] James Chapman. Type theory should eat itself. *Electronic Notes in Theoretical Computer Science*, 228:21–36, 2009.

[19] James Chapman, Thorsten Altenkirch, and Conor McBride. Epigram reloaded: A standalone typechecker for ETT. In Marko van Eekelen, editor, *Trends in Functional Programming Volume 6*. Intellect, 2006.

[20] Jesper Cockx, Dominique Devriese, and Frank Piessens. Pattern matching without K. In *International Conference on Functional Programming (ICFP 2014)*. ACM, September 2014.

[21] Nils Anders Danielsson and Thorsten Altenkirch. Mixing induction and coinduction. Draft, 2009.

[22] Håkon R. Gylterud. Symmetric containers. Master's thesis, University of Oslo, 2011.

[23] Joachim Kock. Data types with symmetries and polynomial functors over groupoids. In *28th Conference on the Mathematical Foundations of Programming Semantics*, Bath, 2012.

[24] Nicolai Kraus, Martín Hötzel Escardó, Thierry Coquand, and Thorsten Altenkirch. Generalizations of Hedberg's theorem. In *Typed Lambda Calculi and Applications, 11th International Conference, TLCA 2013*, pages 173–188, 2013.

[25] Andres Löh, Conor McBride, and Wouter Swierstra. A tutorial implementation of a dependently typed lambda calculus. *Fundamenta informaticae*, 102(2):177–207, 2010.

[26] Peter Morris and Thorsten Altenkirch. Indexed containers. In *Twenty-Fourth IEEE Symposium in Logic in Computer Science (LICS 2009)*, 2009.

[27] Peter Morris, Thorsten Altenkirch, and Neil Ghani. Constructing strictly positive families. In *The Australasian Theory Symposium (CATS2007)*, January 2007.

[28] Peter Morris, Thorsten Altenkirch, and Neil Ghani. A universe of strictly positive families. *International Journal of Foundations of Computer Science*, 20(1):83–107, 2009.

[29] Peter Morris, Thorsten Altenkirch, and Conor McBride. Exploring the regular tree types. In Christine Paulin-Mohring Jean-Christophe Filliatre and Benjamin Werner, editors, *Types for Proofs and Programs (TYPES 2004)*, Lecture Notes in Computer Science, 2006.

[30] The Univalent Foundations Program. *Homotopy Type Theory:Univalent Foundations of Mathematics*. Institute for Advanced Study, 2013.