Homotopy Type Theory: Unified Foundations of Mathematics and Computation (MURI Proposal)

TEAM MEMBERS:

Jeremy Avigad (Carnegie Mellon University)
Steve Awodey (PI, Carnegie Mellon University)
Robert Harper (Carnegie Mellon University)
Daniel Licata (Wesleyan University)
Michael Shulman (University of San Diego)
Vladimir Voevodsky (Institute for Advanced Study)

COLLABORATORS:

Andrej Bauer (Ljubljana U., Slovenia)
Thierry Coquand (Gothenburg U., Sweden)
Nicola Gambino (Leeds U., UK)
David Spivak (MIT)

April 29, 2014

Introduction

The aim of this proposal is to enable a tightly knit group of expert researchers in logic, mathematics, and computer science to pursue a recent theoretical breakthrough that is reshaping the foundations of those disciplines. *Homotopy type theory* opens up a fundamentally new direction in the foundations of mathematics, computation, and information with new, higher-dimensional data types not previously available in set theory and extensional type theory. Its applications range from allowing mathematicians to work with the "right" notion of equality for mathematical structures, to directly expressing higher mathematical concepts such as n-categories and homotopy types, to offering powerful and flexible new generic programming techniques that facilitate the development of certified software. At the conceptual center of these developments is the new homotopy-theoretic semantics for constructive type theories. The computational proof assistants being developed on this basis will facilitate the large-scale formalization of logic and mathematics, with far-reaching practical implications for mathematics and information science.

The feasibility of this new approach has been demonstrated during a special year at the Institute for Advanced Study (Princeton) in 2012–13 by a group of leading researchers from logic, computer science, homotopy theory, and category theory selected from around the world. Several of the core members of this group are assembled here into a MURI team, and are now uniquely positioned to advance this research program, solving critical open problems and addressing specific issues identified during the IAS year in a proven, effective collaboration. These include developing algorithms to support the new foundations, furthering its applications to pure and applied mathematics and computation, and enhancing existing proof assistants or developing new ones to implement the new foundations, leading to the future wide-spread use of computational proof assistants. The resulting large-scale formalization of logic and mathematics will enable the creation of powerful scientific tools with impact on challenging problems of DoD interest.

Constructive Type Theory and Proof Assistants

As mathematics has become increasingly specialized and technical, new discoveries have become increasingly difficult to evaluate, and proofs have become increasingly large artifacts which may require years of work by experts to verify. The development of mathematical proofs now requires confronting many of the same challenges as the development of large software systems, such as managing complicated interdependencies between components, integrating work by various authors, and working effectively with an artifact that is too big to understand all at once. Moreover, mathematical proofs increasingly *are* large software systems, as proofs of theorems such as the four-color theorem and the Kepler conjecture include computer calculations that are too long to check by hand, and evaluating mathematical results requires verifying the correctness of code.

One way to manage this complexity is to use a computer program called a *proof assistant* to develop proofs and software and verify their correctness. The proof assistant translates a mathematical argument to certain trusted foundational axioms, which ensures that it follows the "rules of mathematics." The assistant also provides interactive development tools, and procedures for automatically discovering proofs. These tools allow a community of mathematicians to build shared, searchable libraries of formalized definitions, theorems, and proofs, facilitating the large-scale formalization of mathematics. There is a substantial and growing body of formalized mathematics using proof assistants—including elementary number theory, multivariate real analysis, complex analysis, measure theory and measure-theoretic probability, abstract algebra, Galois theory, linear algebra, finite group theory, and representation theory, and more— in systems including Agda [Nor07], Coq [BC04], Isabelle [NPW02], HOL [GM93], HOL light [Har96], and Mizar [GKN10]. The same tools can be used to verify computer programs. Thomas Hales, a leading researcher currently directing a large-scale project to verify his celebrated proof of the Kepler conjecture, has recently argued persuasively that formal verification of code is essential to national security [Hal13]: "Formal verification of computer code can be advised when human life or large economic interests are at stake: aircraft control systems, widely adopted cryptographic standards, or nuclear reactor controllers. Formal verification reduces the software defect rate to a level that is scarcely attainable by other means."

However, the effort required to prove a theorem or write a program using these tools is generally at least an order of magnitude higher than using informal methods, which is not tenable for developments that already take years. While there are certainly improvements to be made in user interfaces, automated theorem proving, and other aspects, one potential avenue for radically improving the practicality of proof assistants is to revisit the very foundations on which they are built. The current commonly-accepted foundation of mathematics, Zermelo-Fraenkel set theory with the Axiom of Choice (ZFC), is a particular theory in first-order logic that was established in the first quarter of the 20^{th} century and has remained essentially unchanged since then. However, ZFC is, in certain ways, ill-suited for use in proof assistants: the gap between a high-level argument and its encoding in ZFC is large, and it is difficult to describe the computational aspects of proofs and programs in it. This motivates considering alternate foundations of mathematics.

A leading alternative to set theory is *constructive type theory* [ML98, ML75, ML84], whose basic objects are structured *data types* consisting of lists, trees, functions, etc. Type theory is constructive in three senses. First, it provides *axiomatic freedom*, in that it does not in general affirm certain classical principles, such as the law of the excluded middle ("For every proposition A, either A is true or it is false") and the axiom of choice. To see why the absence of a principle can be a strength, it is useful to consider an analogy to geometry, as in Euclid's *Elements*. Euclid's geometry is *synthetic*, in that it consists of a formal system of rules for describing and reasoning about primitive notions of points, lines, angles, and circles. Euclidean geometry has a *model* in the cartesian plane $\mathbb{R}^2$—every statement of Euclidean geometry can be translated to a true statement about the plane—so one way to prove a theorem about geometry in $\mathbb{R}^2$ is to deduce it from Euclid's axioms, which by the model implies that it holds in the plane. However, if we omit the parallel axiom, Euclidean geometry *also* has models in the hyperbolic plane or on the surface of a sphere (describing sailing on the surface of the earth), so a proof that does not use the parallel axiom actually proves a theorem in many different settings. Moreover, we can replace the parallel axiom with extensions specific to hyperbolic or spherical geometry, to better describe these subjects.

Similarly, because type theory does not insist on certain classical principles (such as the law of the excluded middle), it can be used—like geometry without the parallel axiom—to *synthetically* describe models where these principles fail, and it can be extended to better describe these alternate contexts. For example, the function type $A \to B$ can be modeled not only by the set-theoretic functions from A to B, but also as continuous functions on topological spaces [Sco72]. This allows continuous functions to be described exactly as simply as set-theoretic functions, without ever mentioning topology or continuity. Internally to the system, the idea of continuity is implicit in the sense that *all* functions are continuous; it becomes explicit in the model. A synthetic proof is divided into reasoning in the formal system, which is specific to the theorem in question, and the model construction, which is reused across all theorems. (This is similar to the division of a software project into general-purpose libraries and application-specific code, which is well-known to improve code reuse and modularity.) Consequently, synthetic proofs can be cleaner, more elegant, more reusable, and much easier to formalize in proof assistants.

A second aspect of constructivity is that type theory has a *computational interpretation* as a programming language, where functions are interpreted as computable functions from their domain to their range. Crucially, type theory provides a true higher-order notion of computability, where computable functions are treated as *extensional* mathematical entities—two functions that behave the same on all arguments are indistinguishable—which is not true of encodings as Turing machines in ZFC. Strikingly, the absence of excluded middle and related principles is key not only to synthetic mathematics but also to computation: as a computable function, the law of the excluded middle would require that every computational problem can be solved. Because of this computational interpretation, type theory is the basis of many modern programming languages. Some of these languages provide sophisticated mechanisms for software verification, where important functional, safety, or security properties of code are checked by the proof assistants. Computation also facilitates automated theorem proving, because the prover can run programs.

A third aspect of constructivity is that type theory unifies the notions of "proposition" and "set" by the propositions-as-types principle, according to which proofs are first-class mathematical constructions, on the same footing as numbers or groups or mappings. This enables a *proof-relevant* conception of mathematics, where proofs are themselves objects of study. For example, type theory includes an equality type $x =_A y$, whose elements are proofs that x equals y in the type A. In some ways, this type is similar to equality in set theory. For example, it satisfies a version of Leibniz's indiscernability of identicals: given evidence p for an equality $x = y$, there is a function $\mathsf{transport}_C\,(p) : C(x) \to C(y)$ that transports any property or structure C along an equality. However, in other ways, it is unfamiliar. For example, $\mathsf{transport}_C\,(p)$ is a function, which opens up the possibility that it may do real work. Moreover, proofs of equality can themselves be the subjects of equalities: there is a type $p =_{(x =_A y)} q$ of equalities between two proofs of equality p and q, and a type $r =_{(p=q)} s$ where r and s equate p and q, and so on. The meaning of these equations between proofs of equations was, until recently, quite mysterious; but it turns out to be the key to a radical expansion of the kind of mathematics that can be described synthetically in type theory. This new interpretation is called *homotopy type theory*.

Homotopy Type Theory and Univalent Foundations

The idea of homotopy type theory was discovered independently in about 2006 by Awodey and his student Michael Warren [AW09] and Voevodsky [Voe06], who later proposed the Univalent Foundations program on this basis [Voe10]. Their startling observation is that the equality type can in fact be used to describe the structure of paths in a space, as studied in homotopy theory, the branch of mathematics devoted to the study of continuous deformations. Proofs of $x =_A y$ can be thought of as paths from the point x to the point y in the space A. Proofs of $p =_{(x =_A y)} q$ can be thought of as "paths between paths" or homotopies, and so on. This interpretation imbues the higher equality types with significant mathematical and computational content, and allows types to be used to synthetically describe spaces. It leads to a new foundation of mathematics and computation, where types/spaces replace sets as the basic mathematical objects. This new foundation is being developed via a fruitful symbiotic relationship between type theory and homotopy theory, where homotopy theory suggests improvements to type theory, and type theory, with these improvements, can be used to develop mathematics and software in new ways. Moreover, since certain types that are "homotopy sets" behave like conventional sets, there is no loss in expressiveness.

One such improvement is Voevodsky's *univalence axiom*, which states that for a universe type "$\mathcal{U}$" whose elements X,Y are other types (as often used for generic programming), the identity type $X =_{\mathcal{U}} Y$ is equivalent to the type $\mathsf{Equiv}(X,Y)$ of equivalences between the types X and Y. This allows type theory to describe the "classifying spaces" of homotopy theory, but also formally justifies the identification of isomorphic structures: it yields a function

$$\mathsf{ua} : \mathsf{Equiv}(X,Y) \to (X =_{\mathcal{U}} Y)$$

which transforms an isomorphism of groups (say) into an *equality* of groups. This should be regarded not as a collapse of isomorphism to equality, but an expansion of the meaning of "equal" to include isomorphism, exploiting proof-relevance. Computationally, this builds in invariance under data representation: since equivalent types are equal, all structures and algorithms can be automatically transported across any equivalence. Univalence is unavailable in set theory, where "perverse" statements can distinguish between isomorphic structures, and in extensional type theory, where equalities have no computational content; but it is compatible with intensional type theory, and so can enhance proof assistants such as Coq and Agda.

A second improvement is *higher inductive types*: data types whose constructors include not only elements, but equalities between elements (or certain other equalities). These were introduced to describe the basic constructions of homotopy theory with a "logic of homotopy types." For instance, the circle $\mathbb{S}^1$ is generated by one element $\mathsf{b} : \mathbb{S}^1$ and one equality $\mathsf{loop} : \mathsf{b} =_{\mathbb{S}^1} \mathsf{b}$, modeling a space generated by one point and one non-trivial loop. Thus we can define a function $f : \mathbb{S}^1 \to A$ by pattern-matching on these constructors by specifying $f(\mathsf{b}) : A$ and $f(\mathsf{loop}) : f(\mathsf{b}) =_A f(\mathsf{b})$. Such types support formal proofs of classical results in homotopy theory, such as calculating homotopy groups of spheres. If higher inductive types can be shown to be constructive, these proofs would become new algorithms. Computationally, higher inductive types would allow a programmer to specify when two structures should be considered representations of the same underlying object—by adding an equality constructor to a data type—in a way that is automatically respected by all operations. This subsumes quotient types, which have been problematic in intensional type theory, and has applications to modeling domain-specific specification logics and version control systems.

The resulting conception of homotopy type theory is a foundational system of type theory with an intrinsic homotopical character, augmented by new principles of reasoning that strengthen this interpretation, and with an accompanying implementation in a proof assistant. This new approach could remake foundations into a practical tool for the working mathematician. The "geometric" intuition underlying the univalent approach makes it well-suited both to human mathematical practice and to computer formalization. It provides promising new methods for formalization, reducing the gap between formal and informal proofs, and enabling synthetic approaches to disciplines that would otherwise be quite difficult to formalize. Moreover, if the computational interpretation of type theory can be extended to homotopy type theory, the resulting system will provide the first account of higher-order computable functions on higher-dimensional spaces and categories, with promising applications to verified mathematical computations and to software verification. As this new foundation becomes better understood and better implemented, it has the potential to revolutionize our understanding of the relation between mathematics and computation, and to underlie the development of powerful, practical tools for the working scientist in fields farther removed from pure mathematics such as hardware and software verification, cyber-security, artificial intelligence, robotics, and human-computer interaction.

This MURI project continues the ground-breaking work done in 2012–13, when the Institute for Advanced Study (IAS) School of Mathematics hosted a special year devoted to the topic *Univalent Foundations of Mathematics*, organized by Steve Awodey, Thierry Coquand, and Vladimir Voevodsky. This work conclusively demonstrated the viability of this new foundation and its practical advantages. For the special year, a team of 32 leading researchers in the areas of computer science, logic, and mathematics from around the world was assembled at the IAS to develop this new foundation of mathematics. The goals of the IAS program, which included modifying existing proof assistants to support homotopy type theory, a computational interpretation of univalence and higher inductive types, formalizing new areas of mathematics in proof assistants, and relating homotopy type theory to conventional foundations, were realized beyond expectations. In addition to a body of theoretical results pertaining to the foundations, a substantial amount of mathematics was developed in this new system, including basic results in homotopy theory, higher category theory, set theory, and the beginnings of real analysis. In parallel, efforts were focused on the development of new and existing computer proof assistants for the formalization of these and future results. Two working prototype proof assistants based on modifications of the existing systems Coq and Agda were implemented, and several independent but related libraries of code were established on which future work can be built. Moreover, formalized proofs of significant results in homotopy theory were given, such as computing many homotopy groups of spheres. In a remarkable collaborative effort, a textbook was also written, developing both the foundations and various specialized areas of mathematics in the new logical system. This book not only serves as a record of the results of the special year, but also as a useful introduction for researchers entering the field (see [Pro13]).

Overview of Proposed Work

The main goal of this project is *to further develop a new foundation of mathematics and computation based on type theory with improvements suggested by homotopy theory*, and to apply the resulting insights to develop new mathematical results, libraries of formalized mathematical proofs, verified software systems, improvements to existing proof assistants, and/or experimental prototypes of new proof assistants.

The MURI team assembled here to accomplish this goal includes the organizers of the IAS special year, who are also the founders of this research program, together with other highly successful participants. In addition, a number of "unfunded collaborators" are included in the team concept; these are individuals who have made vital contributions at IAS and formed especially effective collaborations with the research group. Based on the open problems arising in these recent investigations, the following research topics, to be pursued in the proposed work, are judged to be the most promising directions toward likely solutions and significant advances. The exposition is divided into four thematic sections, corresponding to the four main areas of work at present.

Foundations (Section 1.1) We propose to develop mathematics using the new principles of univalent foundations, and to further extend type theory to better support these developments. For example, it is necessary to investigate the theory of higher inductive types, which will form the basis for a native implementation in proof assistants (a partial implementation in Coq is already in progress). Another topic is to investigate a new class of type-theoretic languages combining homotopical equalities (paths) and strict equalities in one system, enabling proof assistants to fruitfully distinguish directed computational behavior from more flexible mathematical identities.

Formalization (Section 1.2) Formalization of mathematics using homotopy type theory has already been a striking success using existing proof assistants. For example, synthetic homotopy theory permits much more abstract and yet elementary and concise proofs. We propose to build on these initial successes and develop software libraries of mathematical results, demonstrating the advantages of homotopy type theory. Areas of mathematics under investigation include homotopy theory and category theory, but also analysis (higher inductive types support a new powerful constructive definition of the real numbers) and even set theory. We also propose to formalize the mathematics that justifies homotopy type theory itself, including its syntax and semantics.

Computation (Section 1.3) It is an open problem whether there is a computational interpretation of homotopy type theory, though partial results for truncated theories and work on constructive simplicial and cubical models suggest that there should be one. A solution to this problem would advance applications to programming, verification, and proof automation, and could even provide insights into difficult problems in algebraic topology. We propose to investigate this problem, to implement prototype implementations of algorithms that are developed, and to develop programming and software verification applications that will be possible if this work succeeds.

Semantics (Section 1.4) Models of homotopy type theory relate the new foundations to existing ones, justify synthetic approaches to mathematics, and answer questions about the consistency and constructivity of the theory. Models of type theory with enhancements such as higher inductive types and univalence can be constructed using the spaces of classical homotopy theory. Importantly, however, there are also "nonstandard" models in which the spaces are enhanced in some way, such as with differentiable structure or infinitesimal thickenings. We propose to investigate models of homotopy type theory, and to apply these models to synthetically incorporate many other kinds of mathematics. A notable example is synthetic differential geometry and differential cohomology, which can provide a formal context for quantum gauge field theory. We also propose to investigate how to embed classical arguments in the richer language of constructive type theory. These models and embeddings may allow proof assistants based on homotopy type theory to interact with other systems, and take advantage of existing libraries and tools.

1 Technical Approach

1.1 Foundations

In this section we discuss two strands of proposed work: on developing mathematics using the new features of homotopy type theory, and on extensions to type theory to support these developments.

While homotopy type theory is, in general, a constructive theory of types or spaces, it includes familiar set-based and non-constructive mathematics as a special case. A *proposition* is a type with the property that any two elements are equal. Set-based mathematics takes place in a restricted fragment of types called *sets*, whose types of paths are propositions. Propositions and sets are just the first rungs in an infinite hierarchy of *n-types*, which are types whose n-paths are sets. While the law of the excluded middle for arbitrary spaces is not compatible with univalence, it is consistent to assume the law of the excluded middle for propositions, so non-constructive mathematics can be carried out. Thus, one can always develop mathematics by using traditional definitions and methods. Even for such conventional set-based developments, univalence offers some new ideas, such as the principle that bijective sets and isomorphic structures are connected by an equality.

However, sometimes there is a better way to express a piece of mathematics, taking advantage of the new features of homotopy type theory. Some areas of set-based mathematics clearly fit into a corresponding area of the mathematics of spaces; for example, the theory of finite groups is naturally a part of the theory of homotopy finite groups, related to notions of homotopy lie groups and p-compact groups. Some areas of mathematics, such as category theory, require substantial change for proper univalent presentation; for example, it is better to define a 1-category with not a *set* but a fully higher-dimensional *type* of objects, and then investigate when the paths in the type of objects correspond to the isomorphisms in the category. Some areas can be expressed synthetically, such as homotopy theory itself, which can be studied not by starting from a combinatorial or topological model, but by using the intrinsic homotopy structure of types. Some areas of mathematics can be expressed synthetically if we make additional extensions to type theory; for example, types can be given an intrinsic topological or smooth structure, which allows aspects of differential geometry to expressed synthetically and has applications to mathematical physics. Finally, there are areas, such as analysis, where substantial modifications to conventional developments may be desirable, but we do not yet know how the new development should go.

The book *Homotopy Type Theory: Univalent Foundations of Mathematics* [Pro13], written during the IAS special year primarily by the members of this MURI team, presents a first foray into developing various areas of mathematics in the new foundations. Since its release in late June 2013, some 683 copies of the book have been sold (a striking number, for an advanced mathematical monograph, especially since it is available free online), and 18,730 copies have been downloaded. The book describes both the basic definitions of the new foundations, and some preliminary applications to homotopy theory, category theory, set theory, and analysis.

A major component of this project will be the continued development of various areas of mathematics in the new foundations. This will create new targets for formalizations and motivate possible further improvements to type theory. A related thrust is the further development of the type theoretic foundations themselves. We next discuss the new features of homotopy type theory that we now employ or plan to investigate and how they improve the development of mathematics.

Univalence The univalence axiom states that the type $A =_{\mathcal{U}} B$ of proofs of equality between A and B is equivalent to the type $\mathsf{Equiv}(A,B)$ of equivalences between A and B. Equivalence is a generalization of the notion of bijection or isomorphism to higher-dimensional structures, so the special case of univalence for sets implies that bijective sets are equal, in the sense that a bijection $b : A \cong B$ induces a proof $\mathsf{ua}(b)$ of $A = B$. Therefore, any property or structure can be transported along a bijection: given property or structure $C : \mathcal{U} \to \mathcal{U}$, the rules for the equality type stipulate that there is a function $\mathsf{transport}_C\,(\mathsf{ua}(b)) : C(A) \to C(B)$, which transforms A's into B's in context C. Moreover, $\mathsf{transport}_C\,(\mathsf{ua}(b))$ is itself an equivalence, with inverse given by $\mathsf{transport}_C\,(\mathsf{ua}(b^{-1}))$. An example application of this principle is as follows: An algebraic structure such as a group can be represented by a type family $\mathsf{GroupStructure} : \mathcal{U} \to \mathcal{U}$, where the type $\mathsf{GroupStructure(A)}$ includes the identity ($\mathsf{ident} : A$), inverse ($\mathsf{inv} : A \to A$), and multiplication operations of the group ($\otimes : (A \times A) \to A$) as well as proofs of the associativity, unit, and inverse laws; (e.g. $\mathsf{assoc} : \Pi x,y,z : A.x \otimes (y \otimes z) = (x \otimes y) \oplus z$). Now it is common for a single mathematical concept to be representable as a set in multiple ways, which are all in bijection with each other. In informal math, one might omit the "obvious" proof that a group structure can be transported along a bijection. In homotopy type theory, the proof is indeed obvious, because it is a single application of univalence: Given a bijection $b : A \cong B$, the function $\mathsf{transport}_{\mathsf{GroupStructure}}\,(\mathsf{ua}(b))$ induces a bijection between $\mathsf{GroupStructure}(A)$ and $\mathsf{GroupStructure}(B)$.

Moreover, one can define the notion of a homomorphism between two groups (a map between them that preserves the group operations) and then the notion of isomorphism (homomorphisms back and forth that compose to the identity). Theorems in algebra are typically invariant under group isomorphism, in the sense that if a theorem holds for one group, then it also holds for any other group that is isomorphic to it. Consequently, it is common to adopt informal "abuses of notation" where isomorphic groups are identified. However, with univalence, an isomorphism i between groups G and H can indeed be used to construct an element of the equality type $G = H$. Thus, in univalent foundations, these informal abuses of notation become actual mathematical reasoning principles, and indeed further mathematical structures for investigation.

The idea that univalence allows one to define a mathematical concept with the "right" notion of equality extends to higher-dimensional objects, such as categories. Usually a category consists of a set of objects and, for any two objects, a set of morphisms. In homotopy type theory, we instead define a *pre-category* to have a (possibly higher-dimensional) type of objects (but still a set of morphisms), and then say that a *category* is a pre-category in which the paths in the type of objects correspond to the isomorphisms—a relativized version of the univalence axiom. This yields a new notion of "Rezk completion" [AKS13], which is a universal way to replace a pre-category with a category. Categories of this sort enjoy several useful properties that pre-categories, or categories constructed in set-based foundations, do not. For example, in category theory, it is often necessary to use a coarse notion of equality, such as equivalence (rather than isomorphism) of categories. For pre-categories, equality corresponds to isomorphism, but for categories, equality corresponds to equivalence. Thus, univalent foundations provides the "right" notion of equality for these higher structures, in the sense that an equivalence between categories C and D induces an element of the equality type $C = D$. Moreover, when used synthetically to describe nonstandard semantic models (see §1.4), such categories correspond to *stacks* of ordinary categories, which are well-known to be the "correct" notion of internal category in such settings in algebraic topology and geometry.

Proposed work on univalence includes investigating how univalence simplifies the development, and formalization (see §1.2), of other areas of mathematics. Proposed work also includes investigating its constructivity (§1.3) and semantics (§1.4), which are discussed below.

Higher inductive types Higher inductive types are a new way to introduce types, by giving generators not only for elements of the type, but also for equalities or paths. Higher inductive types have applications in many areas of mathematics, including category theory and analysis, but the most paradigmatic is synthetic homotopy theory: we can study the homotopy theory of types, representing paths by the equality type, and using higher inductive types to construct basic spaces. These synthetic theorems can be interpreted in topological or combinatorial models (such as simplicial sets), and in other contexts as well. During the IAS special year, the members of this MURI team and collaborators developed a number of basic results in homotopy theory, including calculations of some homotopy groups of spheres, such as $\pi_n(S^n)$, $\pi_k(S^n)$ for $k < n$, $\pi_3(S^2)$ and $\pi_4(S^3)$; a calculation of the total space of the Hopf fibration; proofs of the Freudenthal suspension theorem, the Blakers-Massey theorem, and the Seifert-van Kampen theorem; a construction of Eilenberg-Mac Lane spaces; and an investigation of the theory of covering spaces; proofs of the Eilenberg-Steenrod axioms for cohomology; and a construction of spectral sequences. These proofs are not simple rephrasings of traditional proofs; instead, they combine key ideas from traditional proofs with new mechanisms that are emphasized in type theory. This leads to clean and elegant proofs, which, as we discuss further in §1.2, are very amenable to formalization in proof assistants. Moreover, if univalence and higher inductive types can be given a computational interpretation (see §1.3), then these new proofs will be constructive. This may potentially lead to new results, because many of the theorems in this area of mathematics involve calculations of groups or maps, and using constructive methods creates possibilities for using a computer to do these calculations. For example, the calculation of $\pi_4(S^3)$ in homotopy type theory is a proof that there exists a number k such that $\pi_4(S^3) = \mathbb{Z}_k$. Given constructivity, this proof could be executed, in order to automatically calculate k (which should be 2).

While specific higher inductive types have been formulated precisely, proposed work on higher inductive types includes developing a general schema that would allow formal study of a useful class of higher inductive types. This is necessary for studying their computational behavior (§1.3) and semantics (§1.4). Solutions to these problems may eventually enable a native implementation in proof assistants; a partial implementation in standard Coq is already in progress.

Exact equality and other approaches to "infinite objects" Above, we described a univalent approach to 1-categories. Extending this approach to higher dimensions is a major open problem, which may motivate some extensions of homotopy type theory. There are currently many equivalent notions of "higher-dimensional category"; it remains to be seen which are most amenable for use in homotopy type theory. However, all of them require as underlying data an infinite system of sets or types with maps between them. If we formalize this directly in homotopy type theory as a diagram of "sets" (types whose path-spaces are propositions), then there is no problem; but this approach would not support an analogous notion of "relative univalence", and hence in particular would not represent the correct "higher stacks" in ∞-topos models. If instead we use a system of general types, then the corresponding diagram must be required to commute up to "all higher homotopies". Making this precise is a difficult problem of higher category theory, which is what we are trying to define; thus there is a "chicken and egg" problem.

One approach to solving this is to make use of type dependency, so that rather than a diagram of unrelated types and maps between them, we have a sequence of types, each dependent on the previous ones. In this case, the required commutativities hold by definition, hence automatically coherently. While it is possible to define some higher-dimensional structures in this way, such as "globular types", the more important ones for higher category theory, such as (semi-) simplicial types, involve combinatorial complexity that no one has yet succeeded in encoding with dependent types working purely inside the type theory (see [Her13] for a recent attempt). It may be that only an extra dose of ingenuity is required; but even if so, the difficulties involved in merely making the definition suggest that such structures might be difficult to use in practice.

This motivates the exploration of variations in the underlying type theory that are better adapted for the purpose. One possibility is a type theory with two kinds of equality: one a homotopical "path type" as discussed above, and the other a stricter "exact equality". While transporting a structure or property along a path can have significant content, transporting along an exact equality would be "silent," like a traditional notion of equality in set theory or extensional type theory. Having an exact equality type would require some distinction between types that are "fibrant," in the sense that they respect paths, because the exact equality type itself does not respect paths.

There are other possible solutions to this problem. One is to strengthen the ties between the type theory and the metatheory, so that type dependency structures can be defined by induction in syntax and then instantiated internally. A second is to augment the type theory by a basic notion of "diagram of types", whose behavior is specified by rules generalizing those for single types.

Finally, a third possibility is to sidestep the matter by working "synthetically" in the internal logic (see §1.4) of an ∞-topos of diagrams of the appropriate shape. Yet other possibilities probably remain to be thought of. Proposed work includes exploring the design space of type theories, with the need to represent "infinite objects" as a guiding desideratum.

Modalities As discussed in §1.4, homotopy type theory has semantics not only in classical algebraic topology, but in other ∞-toposes; thus all the mathematics mentioned previously can be interpreted in these models. However, just as classical mathematics validates special principles not satisfied in all models (such as the law of excluded middle and the axiom of choice), particular stack models have their own special behavior. By adding corresponding rules to homotopy type theory, we can perform different kinds of synthetic mathematics that will apply to suitable models.

Often the additional rules are *higher modalities*. These are analogous to the modalities $\Box A$ ("necessarily A") and $\Diamond A$ ("possibly A") of classical modal logic; however, they apply not only to propositions, but to sets and higher-dimensional types. The most basic modalities, which always exist, are the n-truncations $\|A\|_n$, which discard all information above dimension n. In particular, the (-1)-truncation is a "squash" [Con83, Con85] or "bracket" type [AB04], which discards existential witnesses, forcing any two inhabitants of $\|A\|_{-1}$ to be equal. As for other modalities, the most extensively studied example (see [SS12]) is *cohesive* homotopy type theory, which corresponds to models incorporating topological or smooth structure. Its new higher modalities are $\flat$ (which makes the cohesion discrete), $\sharp$ (which makes it codiscrete), and $\int$ (the "shape" or "fundamental ∞-groupoid"); they are related by an adjoint string $\int \dashv \flat \dashv \sharp$. This enables a simple description of concepts in differential cohomology. For instance, if F is any type, let $B\mathrm{Aut}(F)$ denote the subtype of the universe $\mathcal{U}$ consisting of all types isomorphic to F; it may be defined as $\sum_{A:\mathcal{U}} \|A = F\|_{-1}$. The univalence axiom implies that $B\mathrm{Aut}(F)$ is a *classifying space* for bundles with fiber F; but cohesively we also have the type $\flat B\mathrm{Aut}(F)$, which is a classifying space for such bundles equipped with a *flat connection*. These are the basic ingredient of higher quantum gauge field theory, the basic framework of modern physics. Thus, cohesive homotopy type theory is a natural framework for the formal analysis of such theories [SS12].

Proposed work includes formalizing this "synthetic cohesive mathematics", and investigating applications of modalities to language-based security (see §1.3). The ∞-toposes of diagrams mentioned above are also cohesive, so higher category theory may also involve higher modalities.

Resizing Rules To avoid paradox, types are stratified by size into various universes. We plan to investigate resizing rules that improve the flexibility of this stratification. For example, classical mathematics is impredicative: a proposition can be determined by quantification over all propositions, including itself. This principle can consistently be added to type theory by a rule that a homotopy proposition in any universe is in (or is equivalent to a proposition in) the smallest universe. Similar such resizing rules would allow for a systematic construction of truncations, quotients, and other higher inductive types with good computational behavior.

1.2 Formalization of mathematics

Next, we describe proposed work on formalizing mathematics using proof assistants. This work will develop a valuable library of formalized mathematics, and investigate the extent to which the new foundations improve the process of doing so. Preliminary results have been promising. For example, univalence provides an improved treatment of isomorphism and equivalence—while current proof assistants do not understand a statement like "by abuse of notation, we identify isomorphic groups", in univalent foundations, isomorphism between groups yields an element of the equality type. Synthetic homotopy theory has also provided very promising initial results.

Existing formalizations One central task will be to extend the library of theorems that have been derived and formally verified in this framework. There are currently three such libraries: one developed by Voevodsky in the Coq proof assistant; a second library, developed by Bauer, Lumsdaine, Shulman, and others, in Coq; and a third library developed by Licata, Brunerie, Hou, and others in the Agda proof assistant. Since homotopy type theory is still in an exploratory developmental phase, these parallel developments are highly desirable, in that they make it possible to experiment with different styles of formalization and different ways of incorporating automation.

All three libraries include basic homotopy-theoretic constructions and their properties, including operations on paths, transport along paths, properties of dependent products and sums, contractible spaces, fibrations, equivalences and their properties, as well as proofs of the surprising fact, discovered by Voevodsky, that the univalence axiom implies function extensionality. The work [AKS13] on category theory in univalent foundations described in Section 1.1 was developed formally in Voevodsky's library. Another library of facts about limits of diagrams over graphs [AKL13] has been developed based on the second Coq library. Most of the formalizations in synthetic homotopy theory mentioned above were developed in the Agda library; see e.g. [LS13, Pro13]. These include calculations of some homotopy groups of spheres, such as $\pi_n(S^n)$ and $\pi_k(S^n)$ for $k < n$; proofs of the Freudenthal suspension theorem, the Blakers-Massey theorem, and the Seifert-van Kampen theorem; a construction of Eilenberg-Mac Lane spaces; and an investigation of the theory of covering spaces.

We have found that in many cases, the informal and formal development of the mathematics go hand in hand, in the sense that the proof assistant proved to be an extremely valuable tool in guiding the exploration and checking the results. Moreover, especially when doing the "synthetic" sort of mathematics that homotopy type theory is well-adapted to, the formal and informal proofs tend to be much more similar, both in content and length, than is usual for fully formalized mathematics.

Proposed formalizations We plan to develop formalized libraries in several areas of mathematics, exploring the extent to which the new foundations described above facilitates the formalization of logic, set theory, algebra, arithmetic and number theory, real and complex analysis, analytic and algebraic geometry, homotopy theory, category theory, and other topics. A starting point is the results in homotopy theory, set theory, and analysis in the Homotopy Type Theory book [Pro13] which have not been formalized in a proof assistant. A formalization of real analysis following the book would improve on existing techniques that use setoids—and given a computational interpretation of homotopy type theory, would be a verified implementation of arbitrary-precision real arithmetic. In homotopy theory, results waiting to be formalized include a calculation of the total space of the Hopf fibration, calculations of $\pi_3(S^2)$ and $\pi_4(S^3)$, proofs of the Eilenberg-Steenrod axioms for cohomology, and a construction of spectral sequences. Some of these formalizations may motivate changes to the libraries or proof assistants; for example, attempts to formalize the calculation of the total space of the Hopf fibration have gotten stuck on large calculations with paths, so this task is a good test for work on a computational interpretation. There are also many homotopy theoretic theorems which have not yet been proved synthetically, but are simple to state, and thus potential targets for formalization. Given a computational interpretation, these formalizations could be used to automatically compute invariants of such homotopy-theoretic structures, a notoriously difficult task in algebraic topology. An example mentioned in §1.1 is automatically computing the k such that $\pi_4(S^3) = \mathbb{Z}_k$.

Another topic for formalization is the mathematics that underlies homotopy type theory itself. Models of homotopy type theory (§1.4) show its consistency relative to classical mathematics and justify new rules for synthetic mathematics. For instance, the univalence axiom was originally motivated by its truth in the "standard" model of Kan simplicial sets. As other new features are added to type theory, it is important to extend the existing models to encompass them. The basic idea used to construct models of type theory is to describe an algebraic structure, known as a "contextual category" or "C-system", and show that the formal syntax of a particular type theory is a presentation of the *initial* C-system with additional operations corresponding to the rules of that theory. Given this, in order to obtain a model in (say) Kan simplicial sets, it suffices to construct a C-system out of them: The interpretation of types and terms in the model is then supplied by the unique map of C-systems from the syntactic one into the semantic one. There are many technical details involved in such an argument, both in formulating the syntax of type theory and in constructing the relevant model structures, and in very few cases have they been written out carefully in the literature. For the Calculus of Constructions (the core type theory underlying Coq), it was shown by [Str91]. Since semantics is so important both for consistency and for applications, we propose to formalize the model construction itself in a proof assistant. This formalization can then support the iterative development of new features and applications for homotopy type theory, by formally verifying their semantic consistency as they are introduced.

Proposed work on proof automation Despite the advances in interactive theorem proving in recent years, spelling out every last detail of a formal argument can be tedious and difficult, even in a framework as powerful as homotopy type theory. One thing that makes formalization more palatable is when the computer can automatically fill in straightforward parts of an argument. The use of formal methods is even more advantageous when the computer can help find nontrivial proofs and constructions. As a result, an important goal is to develop better automated support.

In one of the Coq libraries, a modicum of automation is already used to simplify expressions and construct paths, using Coq's internal "tactic" language. But although this language is easy to use, it is slow and not very expressive. When it comes to conventional classical reasoning, a wide array of reasoning tools have been integrated with interactive theorem provers, whereas no such tools are currently available for homotopy type theory. Part of the problem is that the language is so expressive; because the framework can encode so much, it is hard to find fully general ways of coping with all the sorts of search and decision problems that may arise. But this only means that additional thought and care is required to isolate fragments and areas where additional context makes it possible to design efficient and effective reasoning procedures.

Proposed work on verified mathematical computation Because of the work on a computational interpretation (see §1.3), homotopy type theory may offer new possibilities for verified mathematical computation. Both the Appel-Haken proof of the four color theorem and Hales' proof of the Kepler conjecture were controversial because they relied on computations that are too long to check by hand. The ability to verify these computations is one of the most important advances in Gonthier's verification of the four-color theorem and Hales' work on the Flyspeck project [HHM$^+$10]. At present, however, the mathematical community has not developed general means of verifying symbolic computations. Some progress has been made; for example, the Kenzo system can be used to calculate homology groups and other algebraic invariants of topological constructions [HPRS11, RR13, RER09], and some of the relevant algorithms, but by no means the whole system, have been verified in Isabelle, ACL2, and Coq [ABR08, HPR11]. If work on a computational interpretation succeeds, then homotopy type theory would provide a single framework for specifying and computing with mathematical objects, holding out the promise of carrying out robust and verified mathematical computation. We plan to investigate applications to these problems; a first step would be extracting algorithms from the above formalizations.

1.3 Computation

A critical open problem is whether homotopy type theory has a computational interpretation. This would allow its use as a programming language, with univalence and higher inductive types being new programming constructs. On the computer science side, univalence would mediate between different representations of data structures. On the mathematical side, this would provide a mechanism for verified mathematical calculations in areas such as homotopy theory.

Formally, by a *computational interpretation* of a type theory, we mean an algorithm that, given a closed element e (one with no undischarged assumptions) of natural number type $\mathbb{N}$, deduces an equation between e and some numeral k. Because e will generally involve functions with arbitrary domain and range, this requires computing with functions of arbitrary type—stating the overall theorem only for $\mathbb{N}$ offers some flexibility to solutions. In this section, we give some example programming applications of such a computational interpretation, and then describe previous and proposed work on this problem.

Structure transport and code reuse In a proof or program, one often needs to work with different definitions of the same data structure—one might lead to simpler definitions, while another may be more efficient. Often these representations are in bijection with each other, so univalence can be used to mediate between them. For example, in §1.1, we discussed how univalence enables automatically lifting a bijection $b : A \cong B$ to a bijection between group structures on A and B. A computational interpretation would interpret this lifting as an algorithm. For instance, consider two representations of the integers, $\mathbb{Z}$ in unary, and $\mathbb{Z}'$ in binary; these are related by a bijection $b : \mathbb{Z} \cong \mathbb{Z}'$, which by univalence ($\mathsf{ua}$) automatically lifts to a bijection between group structures:

$$\begin{aligned}&t : \mathsf{GroupStructure}(\mathbb{Z}) \to \mathsf{GroupStructure}(\mathbb{Z}')\\ &t(x) :\equiv \mathsf{transport}_{\mathsf{GroupStructure}}(\mathsf{ua}(b), x)\end{aligned}$$

However, when univalence is added as an axiom to type theory, this function t does not compute: when applied to a group structure $G_{\mathbb{Z}}$ on unary numbers, $t(G_{\mathbb{Z}})$ is a "stuck" program, and does not compute to an explicit implementation of a group structure on binary numbers, because the standard computation rule for $\mathsf{transport}$ is insufficient in the presence of univalence. Thus, type theory with the univalence axiom does not immediately have a computational interpretation.

However, even with these stuck terms, it is still possible to develop and formalize mathematics: $t(G_{\mathbb{Z}})$ *is* related by a path to an implementation of a group structure on $\mathbb{Z}'$ which does not use univalence. The group structure $G_{\mathbb{Z}}$ consists of operations such as addition $\oplus_{\mathbb{Z}} : (\mathbb{Z} \times \mathbb{Z}) \to \mathbb{Z}$ and identity and inverses, along with proofs of the group laws. One way to derive a group structure on $\mathbb{Z}'$, without using univalence, is by "wrapping" $\oplus_{\mathbb{Z}}$ with the bijection—converting arguments from binary to unary and converting the result from unary to binary. This defines a group structure on $\mathbb{Z}'$, which is provably equal to $t(G_{\mathbb{Z}})$. Current developments perform a lot of this kind of reasoning manually; a computational interpretation would provide insight into how to automate it.

Similarly, programmers often need to manipulate many different but bijective implementations of a data structure. These arise sometimes simply because different code is written by different people, but also for more technical reasons. For instance, libraries for datatype-generic programming implement operations, such as equality tests or printing, once and for all for a class of data types. The representation of the data type in the library is often bijective, but not exactly equal, to the one that programmers use in other code. Multiple implementations also arise in verified programming: a simple data structure like lists will be refined many times to track different invariants, such as sorted lists, or lists with a known length. "Forgetting" the refinement results in a data structure that is bijective with the original type of lists. We propose to investigate a computational interpretation of univalence supporting these applications, and to work on making these automatic conversions efficient, possibly by using optimizations such as deforestation [Wad90].

Modular programming Univalence also has applications to the modular development of software. Consider a simple dictionary interface DICT, which includes a type dict of dictionaries, together with insertion and lookup functions. One can implement both a reference implementation $\mathsf{SlowDict}$ of this interface using lists of key-value pairs—which requires only extremely simple code, but has suboptimal running time— and an efficient implementation $\mathsf{FastDict}$ using a balanced binary tree, which requires much more complex code.

One approach to program verification is to write both implementations and prove a relationship between them, such as giving a bijection b between the two dicts and proving that all operations behave the same up to b (e.g. $b(\mathsf{SlowDict.insert}(d_1,(k,v))) = \mathsf{FastDict.insert}(b(d),(k,v))$). In this case, by univalence, the two implementations will in fact be connected by an equality. Thus, programmers can reason about client code by instantiating it with the reference implementation, and automatically obtain the corresponding properties for the efficient implementation. For example, a client of this module would be a function $\mathsf{client} : \mathsf{DICT} \to C$. Because functions take equals to equals, $\mathsf{client}(\mathsf{SlowDict})$ will be equal to $\mathsf{client}(\mathsf{FastDict})$; thus any specification one proves about $\mathsf{client}(\mathsf{SlowDict})$ automatically holds for $\mathsf{client}(\mathsf{FastDict})$. This style of reasoning is similar to the reasoning about abstract types enabled by relational parametricity [Rey83]. While parametricity allows a broader class of relationships between the two implementations, the technique described here has more general applications; for example, one can expand the interface to include convenient conversion to and from the reference implementation, modeling the concept of views for abstract types [Wad87]. Moreover, integrating parametricity into proof assistants is a topic of current research [BJP10]. Preliminary work [Shu13b] suggests that relationally parametric univalent models exist, suggesting a potential avenue towards adding parametricity to homotopy type theory.

Language-based security In language-based approaches to security, the *non-interference property* says that changing secret inputs does not change public outputs. One existing approach to modeling non-interference is via indexed monads [CKP05], but this has the problem that secret information can never be "declassified", because it is impossible to escape from the monad. We plan to investigate whether modalities as discussed in §1.1, which can be implemented by higher inductive types, offer an alternative foundation for non-interference. For example, interpreting "secret" as (-1)-truncated gives a coarse notion of security, because a function $f : \|S\|_{-1} \to X$ must behave the same on any two arguments, though the "secret" input is permitted to influence any "secret" ((-1)-truncated) part of the output. It may be possible to extend this idea by considering different modalities corresponding to secrecy levels.

Modeling computational phenomena as HITs We will also explore modeling computational phenomena as higher inductive types. One preliminary idea along these lines is to model the notion of a *patch* used in version control systems and collaborative editors. A patch represents the difference between two document states; e.g. a basic patch might be "change character a to character b at position 4." If the language of patches is chosen appropriately, they can be modeled as paths in a higher inductive type, whose generators are the basic patches. Moreover, patch equality can be modeled as paths between paths. Writing an interpreter for this language of patches is a programming application of the techniques used in calculating homotopy groups (§1.1).

Previous work on the computational interpretation Thus far, the members of this MURI team have given computational interpretations for fragments of homotopy type theory, and work on several approaches to the general problem is in progress. Conventionally, computation refers to a sequence of definitional expansions that relate a closed program $e : \mathbb{N}$ to a numeral. However, in homotopy type theory, we might instead interpret computation as producing a path. Preliminary work has investigated both of these possibilities for fragments of homotopy type theory where all types are 1-types, which includes many of the programming examples mentioned above. Several of these preliminary results provide proofs of *canonicity* (every element of $\mathbb{N}$ is equal to a numeral); so a *computational interpretation* is a proof of canonicity together with an algorithm for computing the numeral, or a constructive proof of canonicity.

All of these results are based, either formally or informally, on semantic interpretations of type theory in category-theoretic structures, such as groupoids, or homotopy theoretic structures, such as simplicial sets. Licata and Harper [LH12] prove canonicity for a 1-truncated type theory with a univalent universe of sets; this type theory adds many definitional equalities to the syntax, inspired by the groupoid model of type theory [HS98]. However, they do not give an algorithm, and definitional equality is not decidable, so this does not give a computational interpretation. Coquand and Barras [CB13] construct a truncated simplicial model in standard intensional type theory. This does give an algorithm for executing programs, but their work does not give a proof of canonicity—one would need to prove additionally that the equations in the model correspond to paths in the theory. A similar approach, considered by Awodey and Gambino, is to construct the groupoid model of type theory [HS98] in a constructive framework, such as extensional type theory [CAB$^+$86] or the "sets" inside homotopy type theory. This also would give an algorithm, but require a separate proof of canonicity. Shulman [Shu13b] has given a semantic proof of canonicity up-to-paths ("homotopy canonicity"), but it is not clear whether this result is constructive and therefore gives an algorithm.

Proposed work on the computational interpretation A first goal is to tie together the partial results on 1-truncated homotopy type theory: we have canonicity proofs without algorithms and algorithms without canonicity proofs, and should be able to combine the two. This would enable a computational implementation of 1-truncated homotopy type Theory, which would be sufficient for many of the executing the programming examples sketched above. Licata and Harper are investigating a combination of their canonicity proof with the homotopy canonicity proof by Shulman, which would give a constructive proof of homotopy canonicity; this is one potential route to this goal. We could then consider implementations of the resulting algorithms in existing or prototype proof assistants, depending on exactly what result is proved.

Another piece of proposed work is to extend these results to include higher inductive types (e.g. to be able to execute the calculation of $\pi_1(S^1)$). None of these existing results consider higher inductive types, though it seems that many of them would be compatible with them.

Towards the more general problem of giving a computational interpretation of full homotopy type theory, recent work by Coquand et. al. [BCH13] gives a constructive model of type theory in cubical sets, yielding an algorithm for executing programs, but not a proof of canonicity. To obtain canonicity, important work remains to relate the equational theory of this cubical model to the globular syntax of type theory, or to revise the syntax of type theory to be cubical. The former possibility could result in an algorithm to "compute up to homotopy" within existing type theory, which could be implemented by modifying existing proof assistants such as Coq. The latter could potentially lead to prototype implementations of new proof assistants, based on a cubical type theory. It may also be possible to use the insights gained in the cubical model to give a constructive globular definition of weak ∞-groupoids, which would be another approach to this problem.

1.4 Semantics

By *semantics* of a formal system, such as type theory, we mean the investigation of models of that theory constructed within some other system, such as a model of type theory constructed in set theory. Such a model shows that the first system is at least as consistent as the second (an absolute consistency proof is impossible, by Gödel's incompleteness theorem). While one model is enough to show consistency, having many models of a theory is also useful for establishing independence and constructivity results. If a statement is false in some model—such as the law of excluded middle—then it can not be provable in the system, even if it holds in the "standard" model. A model of one theory in another can also reduce computational problems in the first to the second. In the present work, we rely heavily on models to suggest new principles to incorporate synthetically, and to interpret synthetic theorems as statements about more explicit mathematical objects. For foundational theories like type theory, each model furnishes a separate "universe of mathematics" with an "internal language" in which all the usual theorems are true for the usual reasons, but which when viewed "from the outside" can have very informative content.

Expected models In the "standard" model of ordinary type theory, types are interpreted by sets, whereas in "nonstandard" models, they may be interpreted by sheaves, diagrams, topological spaces, smooth manifolds, or even computational objects such as PERs. The standard model yields consistency, while the nonstandard ones give independence results and allow type theory to be used as an "internal language" for many other structures. One direction of research is to improve these models to handle the new features of homotopy type theory. Since models of ordinary type theory are naturally formulated as certain kinds of categories called 1-toposes, we expect models of homotopy type theory to be naturally formulated as certain categories with homotopy theory, now commonly known as "∞-toposes" [TV05, Rez05, Lur09]. This is also suggested by a clear similarity between the univalence axiom and one of the defining features of an ∞-topos (an "object classifier"). In particular, most models of ordinary type theory can be "homotopified" into ∞-toposes that ought to provide semantics for homotopy type theory.

In addition to relative consistency, a precise result of this sort would provide a fertile ground for independence results, as well as ways to use homotopy type theory as an "internal language" for constructions in ∞-toposes. This encompasses equivariant and parametrized homotopy theory, homotopical algebraic geometry, and also more exotic kinds of "synthetic" mathematics. The most explored example of this sort is called "cohesive" homotopy type theory, in which the types (corresponding to objects of the ∞-topos) have not only ∞-groupoid structure but "smooth structure", thereby admitting a synthetic kind of differential geometry and calculus. Just as synthetic homotopy theory is a natural home for classical algebraic topology, cohesive homotopy theory is a natural home for differential cohomology and quantum gauge field theory [Sch13, SS12].

Constructing such models inside a known computational system, such as ordinary intensional type theory, would yield an algorithm for computing with homotopy type theory. In addition to these models in "Grothendieck" ∞-toposes (consisting of sheaves), analogies to ordinary type theory suggest that there should also be models in "elementary" ∞-toposes, including "realizability" examples built out of PERs or assemblies. Models of this sort may improve our understanding of how to compute in homotopy type theory, as well as providing a way to attack more independence results. Finally, it would be best if all of these models could be constructed inside homotopy type theory itself, making it an autonomous foundation for mathematics.

Known models At present, the best framework for constructing models of homotopy type theory uses "Quillen model categories", a toolbox from abstract homotopy theory giving a way to "present" ∞-categories. The "basic" Quillen model category consists of simplicial sets, and presents the ∞-topos of ∞-groupoids; but any ∞-topos can be presented by some such model category. Awodey and Warren [AW09] have shown how any Quillen model category yields a semantics for type theory, modulo coherence issues. And Voevodsky has shown [KLV12] that the resulting semantics in simplicial sets satisfies the univalence axiom (indeed, this was the motivating model for the formulation of univalence), using a method which also deals with the coherence questions. Unpublished work of Lumsdaine and Warren gives a general way to solve such coherence questions for all type forming operations except for universes, and Cisinski [Cis12] and Gepner and Kock [GK12] have shown that any ∞-topos can be presented by a Quillen model category to which this theorem applies. Shulman [Shu13b, Shu13a] has shown that in addition to simplicial sets, a few other very special ∞-toposes admit coherent univalent universes. Finally, unpublished work of Lumsdaine and Shulman shows that the semantics of most Quillen model categories supports higher inductive types as well. Thus, we know that any ∞-topos admits a model of homotopy type theory, with higher inductive types and an "incoherent" version of univalence; and in some special cases (including simplicial sets) we also have coherent univalence.

All of these models are constructed inside of set theory. However, the construction of coherent univalent universes in [Shu13b] can also be performed in homotopy type theory, providing least a small class of model constructions completely independently of set theory. Moreover, work is in progress on a constructive model in cubical sets [BCH13], and Awodey and Bauer have constructed a 1-truncated realizability model using groupoids inside an ordinary realizability model.

Current and future work An obvious direction to explore is whether all ∞-toposes can be presented by Quillen model categories that satisfy univalence coherently. This seems to be a difficult problem, but it may be possible to achieve by a careful choice of presentations. However, there are other potential ways to achieve a satisfactory theory of ∞-topos models. It may be possible to avoid Quillen model categories and construct a model of type theory directly in an ∞-category. This would probably require some sort of coherence theorem for type theory, such as extending the universal property of the syntactic C-system discussed in §1.2 to an ∞-categorical one.

On the other hand, we could modify type theory to make it match the desired categorical models more closely. This may also be necessary to describe ∞-topos models internally to homotopy type theory, due to the problem of infinite objects described in §1.1. Thus, we plan to investigate type theories with fewer judgmental equalities, and with some way to represent infinite constructions internally. The interplay between these desiderata and the constructive algorithmic needs of type theory, as described in §1.3, will guide the team through the design space to the best solutions.

In addition to putting the basic theory of model construction on a firm footing, we also plan to investigate particular models more deeply. Homotopy type theory, and in particular the univalence axiom and higher inductive types, should suggest a good definition of “elementary ∞-topos” (which is still lacking). We can then investigate “realizability” models of this sort, generalizing Awodey and Bauer’s work to higher dimensions. Such categories may not be presentable by Quillen model categories, so they provide another reason to seek a better general theory of models.

References

[AB04] Steven Awodey and Andrej Bauer. Propositions as [types]. *J. Logic Comput.*, 14(4):447–471, 2004.

[ABR08] Jesús Aransay, Clemens Ballarin, and Julio Rubio. A mechanized proof of the basic perturbation lemma. *J. Automat. Reason.*, 40(4):271–292, 2008.

[AKL13] Jeremy Avigad, Krzysztof Kapulkin, and Peter Lumsdaine. Homotopy limits in type theory. To appear in *Mathematical Structures in Computer Science*, 2013.

[AKS13] Benedikt Ahrens, Krzysztof Kapulkin, and Michael Shulman. Univalent categories and the Rezk completion. To appear in *Mathematical Structures in Computer Science*; arXiv:1303.0584, 2013.

[AW09] S. Awodey and M. A. Warren. Homotopy theoretic models of identity types. *Math. Proc. Camb. Phil. Soc.*, 146:45–55, 2009.

[BC04] Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development: Coq'Art: The calculus of inductive constructions*. Springer-Verlag, Berlin, 2004.

[BCH13] Marc Bezem, Thierry Coquand, and Simon Huber. A model of type theory in cubical sets. Preprint, September 2013.

[BJP10] Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. Parametricity and dependent types. In *International Conference on Functional Programming*, 2010.

[CAB$^+$86] Robert L. Constable, Stuart F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, Douglas J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing Mathematics with the NuPRL Proof Development System*. Prentice Hall, 1986.

[CB13] Thierry Coquand and Bruno Barras. A generalization of takeuti-gandy interpretations. Preprint, April 2013.

[Cis12] Denis-Charles Cisinski. Blog comment on post "The mysterious nature of right properness". `http://golem.ph.utexas.edu/category/2012/05/the_mysterious_nature_of_right.html\#c041306`, May 2012.

[CKP05] Karl Crary, Aleksey Kliger, and Frank Pfenning. A monadic analysis of information flow security with mutable state. *Journal of Functional Programming*, 15:249–291, 3 2005.

[Con83] Robert L. Constable. Mathematics as programming. In *Proceedings of the Workshop on Programming and Logics*, volume 164 of *LNCS*, pages 116–128. Springer-Verlag, 1983.

[Con85] Robert L. Constable. Constructive mathematics as a programming logic I: Some principles of theory. In *Annals of Mathematics*, volume 24, pages 21–37. Elsevier Science Publishers, B.V. (North-Holland), 1985. Reprinted from *Topics in the Theory of Computation*, Selected Papers of the Internationall Conference on Foundations of Computation Theory, FCT '83.

[GK12] David Gepner and Joachim Kock. Univalence in locally cartesian closed ∞-categories. arXiv:1208.1749, 2012.

[GKN10] Adam Grabowski, Artur Korniłowicz, and Adam Naumowicz. Mizar in a nutshell. *J. Formaliz. Reason.*, 3(2):153–245, 2010.

[GM93] M. J. C. Gordon and T. F. Melham, editors. *Inroduction to HOL: A theorem proving environment for higher-order logic*. Cambridge University Press, 1993.

[Hal13] Thomas C. Hales. Formalizing NIST standards. *Jigger Wit*, 2013. http://jiggerwit.wordpress.com/2013/11/04/formalizing-nist-standards/.

[Har96] John Harrison. HOL light: a tutorial introduction. In Mandayam Srivas and Albert Camilleri, editors, *Proceedings of the First International Conference on Formal Methods in Computer-Aided Design*, pages 265–269, 1996.

[Her13] Hugo Herbelin. A dependently-typed construction of semi-simplicial types. To appear in *Mathematical Structures in Computer Science*, 2013.

[HHM$^+$10] Thomas C. Hales, John Harrison, Sean McLaughlin, Tobias Nipkow, Steven Obua, and Roland Zumkeller. A revision of the proof of the Kepler conjecture. *Discrete Comput. Geom.*, 44(1):1–34, 2010.

[HPR11] Jónathan Heras, Vico Pascual, and Julio Rubio. Proving with ACL2 the correctness of simplicial sets in the Kenzo system. In *Logic-based program synthesis and transformation*, volume 6564 of *Lecture Notes in Comput. Sci.*, pages 37–51. Springer, Heidelberg, 2011.

[HPRS11] J. Heras, V. Pascual, J. Rubio, and F. Sergeraert. fKenzo: a user interface for computations in algebraic topology. *J. Symbolic Comput.*, 46(6):685–698, 2011.

[HS98] M. Hofmann and T. Streicher. The groupoid interpretation of type theory. In Sambin and Smith [SS98], pages 83–111.

[KLV12] Chris Kapulkin, Peter LeFanu Lumsdaine, and Vladimir Voevodsky. The simplicial model of univalent foundations. arXiv:1211.2851, 2012.

[LH12] D. Licata and R. Harper. Canonicity for 2-dimensional type theory. *POPL*, 2012.

[LS13] Daniel R. Licata and Michael Shulman. Calculating the fundamental group of the circle in homotopy type theory. In *LICS'13*, 2013. arXiv:1301.3443.

[Lur09] J. Lurie. *Higher Topos Theory*. Princeton University Press, 2009.

[ML75] P. Martin-Löf. An intuitionistic theory of types: predicative part. In H. E. Rose and J. C. Shepherdson, editors, *Logic Colloquium '73*, pages 73–118, Amsterdam, 1975. North-Holland.

[ML84] P. Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, Napoli, 1984.

[ML98] P. Martin-Löf. An intuitionistic theory of types. In Sambin and Smith [SS98], pages 127–172. This paper was originally a 1972 preprint from the Department of Mathematics at the University of Stockholm.

[Nor07] Ulf Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Chalmers University of Technology, 2007.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL. A proof assistant for higher-order logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, 2002.

[Pro13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study, 2013.

[RER09] Ana Romero, Graham Ellis, and Julio Rubio. Interoperating between computer algebra systems: computing homology of groups with Kenzo and GAP. In *ISSAC 2009—Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation*, pages 303–310. ACM, New York, 2009.

[Rey83] J. C. Reynolds. Types, abstraction, and parametric polymorphism. In *Information Processing '83*, pages 513–523. North-Holland, Amsterdam, 1983.

[Rez05] Charles Rezk. Toposes and homotopy toposes. `http://www.math.uiuc.edu/~rezk/homotopy-topos-sketch.pdf`, 2005.

[RR13] Ana Romero and Julio Rubio. Homotopy groups of suspended classifying spaces: an experimental approach. *Math. Comp.*, 82(284):2237–2244, 2013.

[Sch13] Urs Schreiber. Differential cohomology in a cohesive infinity-topos. arXiv:1310.7930, 2013.

[Sco72] Dana Scott. Continuous lattices. *Springer Lecture Notes in Mathematics*, 274:97136, 1972.

[Shu13a] Michael Shulman. The univalence axiom for elegant Reedy presheaves. arXiv:1307.6248, 2013.

[Shu13b] Michael Shulman. Univalence for inverse diagrams and homotopy canonicity. To appear in *Mathematical Structures in Computer Science*; arXiv:1203.3253, 2013.

[SS98] G. Sambin and J. Smith, editors. *Twenty-Five Years of Constructive Type Theory*, volume 36 of *Oxford Logic Guides*, Oxford, 1998. Oxford University Press.

[SS12] Urs Schreiber and Michael Shulman. Quantum gauge field theory in cohesive homotopy type theory. In *QPL'12*, 2012. `http://ncatlab.org/schreiber/files/QFTinCohesiveHoTT.pdf`.

[Str91] Thomas Streicher. *Semantics of type theory: correctness, completeness, and independence results*. Birkhauser Boston Inc., Cambridge, MA, USA, 1991.

[TV05] Bertrand Ton and Gabriele Vezzosi. Homotopical algebraic geometry i: topos theory. *Advances in Mathematics*, 193(2):257 – 372, 2005.

[Voe06] V. Voevodsky. A very short note on the homotopy λ-calculus. Unpublished note, available at `http://www.math.ias.edu/ vladimir/Site3/UnivalentFoundations.html`, 2006.

[Voe10] V. Voevodsky. Univalent foundations project. Unpublished note, available at `http://www.math.ias.edu/ vladimir/Site3/UnivalentFoundations.html`, 2010.

[Wad87] Philip Wadler. Views: a way for pattern matching to cohabit with data abstraction. In *Principles of Programming Languages*, 1987.

[Wad90] Philip Wadler. Deforestation: transforming programs to eliminate trees. *Theoretical Computer Science*, 73(2):231 – 248, 1990.