An introduction to univalent foundations for mathematicians

DANIEL R. GRAYSON

Abstract. We offer an introduction for mathematicians to the univalent foundations of Vladimir Voevodsky, aiming to explain how he chose to encode mathematics in type theory and how the encoding reveals a potentially viable foundation for all of modern mathematics that can serve as an alternative to set theory.

Contents

Introduction

The traditional foundation for mathematics, chosen more than a century ago, pins down basic issues, such as what numbers "really are", by giving them a specific arbitrary internal structure based on sets, which is irrelevant to modern daily mathematical discourse. Bertrand Russell's contemporaneously proposed alternative foundation [32] was

Date: Nov 4, 2017.
2010 *Mathematics Subject Classification.* 03B15.
Key words and phrases. homotopy type theory, type theory, identity type, univalence axiom, univalent mathematics.
MR Author ID: 76410

based on the theory of "types", in which each variable is to be accompanied by a type, which is drawn from a hierarchy of types of increasing complexity and which provides the variable's range of values, the aim being to prevent the formulation of paradoxical concepts, such as the set of all sets. Over the intervening decades type theory has been developed by computer scientists into a useful tool for verifying security of computer languages and by mathematicians into a useful tool for computer verification of mathematical theorems, such the Four Color Theorem (the original 1976 Appel-Haken proof had gaps, but a proof was finally verified by computer using Coq [35] in 2005 [18]).

In this brief introduction, we'll take a glance at the world of mathematics as viewed through the *univalent foundations* of Voevodsky, which is based on type theory, is part of the world of *homotopy type theory*, is formally and precisely specified, and has been under development over the last several years. The most fundamental novelty for mathematicians is that the notion of equation is recast using (dependent) types, as originally used by de Bruijn [17] in the proof checker *Automath*, and as formulated and augmented by Per Martin-Löf [28], so that an equation is no longer necessarily a proposition. Building on that, harking back to a conjecture of Grothendieck about ∞-groupoids, Voevodsky singles out the types at *h-level* n, for a natural number n, with the true "propositions" being the types at h-level 0, the "propositions" being the types at h-level 1, the "sets" being the types at h-level 2, the objects of a category being the elements of a type at h-level 3 that captures the groupoid of isomorphisms of the category, and so on. (The quotation marks in the previous sentence are intended to indicate initially that these "propositions" and "sets" are intended to play the role that propositions and sets play in set theory, but are not the same: rather, they are alternative ways to formalize a common intuitive understanding of the terms[1].) Thus propositions and their proofs become part of the language of mathematics, rather than part of the language of logic, and sets become a special case of something more fundamental, *types*, worthy of independent study.

The formal mathematical language, together with univalence, fulfills the mathematicians' dream: a language for mathematics invariant under "equivalence" and thus freed from irrelevant details and able to merge the results of mathematicians taking different but equivalent approaches. Voevodsky called this invariance property of the language

[1]Perhaps one would prefer to use three terms, when needed to avoid ambiguity: *set* for the intuitive notion, *Zermelo-Fraenkel structure* for the set theory notion (as Voevodsky preferred to call it), and *h-set* for the univalent foundations notion.

"univalence". It offers the hope that formalization and verification of today's mathematical knowledge may be achievable, relieving referees of mathematics articles of the tedious chore of checking the details of proofs for correctness, allowing them to focus on importance, originality, and clarity of exposition. Feasibility and practicality of the approach were demonstrated by Voevodsky in his *Foundations* [37, 41], based on Coq, and various teams have continued formalization efforts based on it. In section 5 we describe how the notions of *group* and *torsor* are encoded in the system, thereby motivating the definitions of *proposition* and *set* and aiming to expose the aspects of the system that make the formalization succeed.

Further useful expositions of the subject include these: [16, 36].

I turn now to a bit of speculation. The beauty of mathematics, as appreciated by human mathematicians, can be only enhanced by formalization, for a beautiful proof known to be correct is more beautiful than the same proof in unverified form. It will still be humans who decide what to prove, how to prove it, which proofs are better than others, and which proofs are worth publishing in journals. The shortest proofs will be easiest to enter into the computer – no one will have the energy to formalize the long complicated ones, until later on, if ever. A beautiful incomplete or incorrect proof can benefit from formalization, too – then anyone will know precisely what has been done, that it has been done correctly, and what remains to be done.

The adoption of proof formalization by a broad segment of practicing mathematicians will await the day when mathematicians feel more successful with formalization than without, as happened with *TeX*, starting in the 1980's. There is much development to be done to make proof assistants more powerful and easier to use. Formalization of a result also depends on formalization of the results cited by it, so another prerequisite is the formalization of a large body of existing mathematics.

Current proof assistants are oriented toward enabling humans to enter proofs efficiently, so perceiving the beauty of a proof by reading its formalization will not be as enlightening as by reading human-written prose about it, except for those who become proficient with the proof-entering mechanism. Eventually, when proof-entering technology is sufficiently developed, we can expect proof-reading technology to be developed further, and, ultimately, the optimal way to read a proof and to learn it will be with the active assistance of a proof assistant, so that details can be clarified upon demand, right down to the bottom. Anyone who has worked their way through even a fragment of EGA, with its numerous cross-references to earlier results, knows how

desirable that is in a text, and how hard it is to achieve outside a unified framework. In an automated comprehensive system, it would be effortless.

Alexander Beilinson has written [10], perhaps expressing a thought of I. M. Gelfand: "Modern mathematics is a unique thrust of conceptual thought: once the right concept (a mathematical structure) and a language to deal with it are found, a whole new world unfolds." This is what has happened with univalent foundations, and in this new world mathematicians and the beauty of mathematics will prosper.

Dedication

I dedicate this article to Vladimir Voevodsky, an ingenious mathematician with an immense drive to create, who died suddenly in September, 2017. Our friendship began perhaps in 1994, when I first became aware of his work on motivic cohomology, and he visited me in Urbana to explain what it was about. Some mathematicians had suspected that there was a larger role for homotopy theory to play in motivic cohomology, but it was up to Vladimir to show the way, which he did with immense energy, and a detailed plan for the future covering many components of his program. His work, including a proof of the Milnor Conjecture (1970), won him the Fields Medal in 2002, went most of the way toward the ultimate solution of the Bloch-Kato Conjecture (of which the Milnor Conjecture is a special case), the Beilinson-Lichtenbaum Conjecture, and the conjecture of Milnor about the Witt ring of quadratic forms. The field of motivic homotopy theory remains a vibrant one to this day, although Vladimir stopped participating in it about ten years ago or so. He started thinking about the use of computers for formal representation and verification of mathematical statements and proof in 2002. We worked together for a week in Spring, 2004, brainstorming about that, dreaming about what an ideal system would be like, and I even wrote some code. I soon became occupied with other matters, but he surveyed the field, and by 2006 had chosen type theory as the proper language. By 2010 he had chosen a suitable encoding of mathematics in type theory (described in this article) and had spent three months writing some brilliant, beautiful, and instructive proofs in Coq, which he dubbed *Foundations*, later incorporated into *UniMath*. Always eager to participate in his project, I finally found the time to start collaborating with him in 2011, and in 2012-2013 I attended the special year on the topic at the Institute for Advanced Study and found it bracing and enlightening. His final project was to establish the soundness of the univalent foundations in a series of

papers, eight of which he managed to write (see section 7). His dream to establish the univalent foundations as the preferred and practical framework for formalizing the world's mathematical knowledge seems feasible, but much work remains to be done by the community. A fitting memorial for Vladimir would be to formalize his work on motivic homotopy theory in the univalent foundations.

Acknowledgments

I thank the Oswald Veblen Fund and the Friends of the Institute for Advanced Study for supporting my stay at the Institute for Advanced Study in Spring, 2017, where I worked on part of this paper. I thank Benedikt Ahrens, Guillaume Brunerie, and Thierry Coquand for useful comments on earlier drafts of this paper.

1. What is a type?

In some computer programming languages, all variables are introduced along with a declaration of the type of thing they will refer to. For example, one may encounters types such as *bool*, *string*, *int*, and *real*, describing Boolean values, character strings, 32 bit integers, and 64 bit floating point numbers. The types are used to determine which statements of the programming language are grammatically well-formed. For example, if s of type *string* and x is of type *real*, we may write $1/x$, but we may not write $1/s$.

Types occur in traditional mathematics, as expressed in informal mathematical speech, and are used in the same way: all variables are introduced along with a declaration of the type of thing they will refer to. For example, one may say "consider a group G", "consider a ring R", or "consider an algebraic variety X over a field k". One does not see circumlocutions such as "consider a thing X: if X is a group then ...".

This informal use of types is not supported by Zermelo-Fraenkel set theory, where there are just two types of mathematical object, propositions and sets.[2] Nevertheless, as with computer languages, here the types are used to determine which statements of the theory are grammatically well-formed. For example, if X and Y are sets, then we may

[2]Perhaps it is more traditional to say that in set theory there is just one type of mathematical object, sets, because one may not introduce a quantifier whose variable ranges over all propositions. The indulgent reader will perhaps permit this departure from tradition, for the sake of anticipating the emergence of the type of propositions in type theory.

write the proposition $X \in Y$, and if P and Q are propositions we may write the proposition $P \wedge Q$, but we may not write $P \in Q$ or $X \wedge Y$.

In type theory, there are many more types and they are used for everything. They may be used to support the traditional use of types in informal mathematical speech.

One expresses the statement that an "element" a is of "type" X by writing $a : X$. All variables are introduced along with a declaration of the type of thing they will refer to, using that notation, and the types are used to determine which statements of the theory are grammatically well-formed. Since sets are to be re-implemented as types with a certain property, we will have no use for the set membership operator $\in$.

When using type theory to formalize mathematics, one encounter types such as $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $Group$, $Ring$, $AbelianCategory$. Their elements are, respectively, the natural numbers, the integers, the rational numbers, the groups, the rings, and the abelian categories.

There will be enough "primitive" ways to form new types from old ones to provide everything we need to formalize mathematics: logical quantifiers, functions, families, pairs, products, sums, and equality.

For example, if X and Y are types, there will be a type whose elements serve as *functions* from X to Y; the notation for it is $X \to Y$. This allows us to introduce the surprising mathematical pun $f : X \to Y$, which says that f is an element of the type $X \to Y$, and which can be read traditionally as saying that f is a function from X to Y.[3]

Functions behave as one would expect, and one can make new ones in the usual way.[4]

[3]This new interpretation of the notation may appear jarring to those who are accustomed to writing something like "consider a function $X \xrightarrow{f} Y$", regarding the arrow as a pictorial representation of the function itself; we won't do that.

[4] To provide an example of making new functions in the usual way, consider functions $f : X \to Y$ and $g : Y \to Z$. We define their composite $g \circ f : X \to Z$ by setting $g \circ f := (a \mapsto g(f(a)))$. Such definitions are to be regarded as syntactically transparent in our formal system, in the sense that two formal expressions will be regarded as being *the same by definition* if they yield the same formal expression after the definitions of all the symbols within them are completely expanded, i.e., we may replace one with the other in any other expression, at will. For example, consider functions $f : X \to Y$, $g : Y \to Z$, and $h : Z \to W$. Then $(h \circ g) \circ f$ and $h \circ (g \circ f)$ are the same by definition, since applying the definitions within expands both to $a \mapsto h(g(f(a)))$.

One may define the identity function $id_X : X \to X$ by setting $id_X := (a \mapsto a)$. Application of definitions shows that $f \circ id_X$ is the same as $a \mapsto f(a)$, which, by a standard convention, is to be regarded as the same as f. A similar computation applies to $id_Y \circ f$.

In the following sections we will expose various other elementary sorts of types, focusing the most attention on those with novel aspects.

2. The type of natural numbers

The inductive definition of the natural numbers provided by Giuseppe Peano in 1889 [30] is mirrored almost exactly in type theory.

Here are Peano's rules[5] for constructing the natural numbers in the form that is used in type theory.

- P1: there is a type called $\mathbb{N}$ (whose elements will be called natural numbers);
- P2: there is an element of $\mathbb{N}$ called 0;
- P3: if m is a natural number, then there is also a natural number $S(m)$, called the *successor* of m;
- P4: given a family of types $X(m)$ depending on a parameter m of type $\mathbb{N}$, in order to define a family $f(m) : X(m)$ of elements of each of them it suffices to provide an element a of $X(0)$ and to provide, for each m, a function $g_m : X(m) \to X(S(m))$. (The resulting function f may be regarded as having been completely defined by the two definitions $f(0) := a$ and $f(S(m)) := g_m(f(m))$.)

Mathematicians will recognize rule P4 as "the principle of mathematical induction" when each $X(m)$ is a proposition, and as "defining a function by recursion" if each $X(m)$ is a fixed set Y. We will refer to it simply as "induction for $\mathbb{N}$".[6] Notice that the two cases in an inductive definition correspond to the two ways of introducing elements of $\mathbb{N}$ via the use of rules P2 and P3. Intuitively, induction for $\mathbb{N}$ corresponds to the statement that, in the absence of variables and axioms and after expansion of definitions, every formal expression representing an element n of $\mathbb{N}$ has the form 0 or the form $S(m)$ for some m, so those are the only two cases that an algorithm corresponding to the definition of f needs to handle.

[5]There is a reason for not calling them Peano's Axioms in this context, since we wish to reserve the word "axiom" for decrees that a proposition has a proof.

[6]Here is an example of defining a function by recursion using induction for $\mathbb{N}$. We define the factorial function $f : \mathbb{N} \to \mathbb{N}$ by setting $f(0) := 1$ and setting $f(S(m)) := (m + 1) \cdot f(m)$. One can infer that the function g_m of rule (P4) is $n \mapsto (m + 1) \cdot n$.

We introduce the following definitions.

$$1 := S(0)$$
$$2 := S(1)$$
$$3 := S(2)$$
$$4 := S(3)$$

We may use induction to define the sum $m + n$ of two natural numbers, as a natural number. We handle the two possible cases for the argument m as follows: we define $0 + n := n$, and we define $(S(m)) + n := S(m + n)$. Application of definitions shows, for example, that $2 + 2$ and 4 are the same by definition, because they both reduce to $S(S(S(S(0))))$.[7]

Before we can write an equation such as $2+2 = 4$, we must introduce a formal treatment of equality in type theory. We do that in the next section.

3. Identity types

The most important type is the *identity type*, which implements the intuitive notion of equality, but in a novel way; the mathematical reader will be more comfortable if we call it the *equality type*, at least initially. Its novelty derives from three aspects: equality between two elements may be considered only when the two elements are of the same type; equality may fail to be a proposition (unless it's an equality between two elements of a set); and a proof of equality between two sets will amount to a bijection between them.

We can anticipate the consequences of the first of those three aspects immediately. For example, let $2_{\mathbb{Z}}$ denote the integer corresponding to 2 and let $2_{\mathbb{Q}}$ denote the rational number corresponding to 2. Then there is no way to translate the valid but mathematically irrelevant equation

[7]The reduction of the expression $2 + 2$ to $S(S(S(S(0))))$ by applying the definitions involved can be regarded as a computation, which is "complete". Computation succeeds in other cases involving natural numbers, too. That is why we don't refer to Peano's rules as axioms, as we mentioned in a previous footnote, for we wish to reserve the word "axiom" for statements unaccompanied by effective methods for computation. A good example of such an axiom is the law of the excluded middle, which states, in our context, that any proposition P either has a proof or its negation has a proof. The law provides no effective way to decide which case we are in nor what the proof would be in that case. Nevertheless, it is consistent with type theory and all the axioms we intend to use, so its use as a hypothesis in a theorem is permissible. If one wishes to define a discontinuous function of a real variable, it is needed, because there is no effective way to decide whether a real number is, for example, positive.

$2_{\mathbb{Z}} = 2_{\mathbb{Q}}$ of set theory into type theory, because $2_{\mathbb{Z}}$ and $2_{\mathbb{Q}}$ are elements of distinct types: $\mathbb{Z}$ and $\mathbb{Q}$. Beginning mathematics students would be completely comfortable with this restriction. For another example, consider two sets (or types) X and Y. Then there is no way to state a condition that says the X is a subset (or subtype) of Y. Some other circumlocution, analogous to the one used in category theory to speak of subobjects of an object, will be used to formalize a proof that speaks of subsets.

Observe that the smallest reflexive relation on the elements of a set X is equality. Per Martin-Löf realized in the 1970's that one can use this observation to provide an inductive definition of equality[8] analogous to the inductive definition of the natural numbers presented in section 2.

Here are Martin-Löf's rules for constructing equality types.

E1: for any type X and for any elements a and b of it, there is a type $a = b$;[9]

E2: for any type X and for any element a of it, there is an element $\mathit{refl}(a)$ of type $a = a$ (the name refl comes from the word "reflexivity")

E3: for any type X and for any element a of it, given a family of types $P(b, e)$ depending on parameters b of type X and e of type $a = b$, in order to define elements $f(b, e) : P(b, e)$ of all of them it suffices to provide an element p of $P(a, \mathit{refl}(a))$. The resulting function f may be regarded as having been completely defined by the single definition $f(a, \mathit{refl}(a)) := p$.

An element of $a = b$ can be thought of, for now, as a proof that a is equal to b.

We see from rule E2 that $\mathit{refl}(S(S(S(S(0)))))$ serves as a proof of $2 + 2 = 4$, as do $\mathit{refl}(4)$ and $\mathit{refl}(2 + 2)$. A beginning student might wish for a more detailed proof of that equation, but as a result of our convention above that definitions are syntactically transparent, the application of definitions, including inductive definitions, is regarded as a trivial operation.

We will refer to rule E3 as "induction for equality". It says that to prove something about (or to construct something from) every proof

[8] See [27, section 1.7]. An antecedent was published in [26, section 3.8.2, p. 190].

[9] We point out the order of quantifiers in rule E1 is grammatically required. Suppose one tried to rephrase rule E1 to have the form "for any elements a and b, ...". Then the puzzle would be how to formulate the condition that a and b have the same type, and there is no way to do that in our formal language, as currently designed. An arbitrary element of an arbitrary type is always introduced by introducing a quantifier first for its type, so the quantifier for the element can state the type of the element, as required.

that a is equal to something else, it suffices to consider the special case where the proof is the trivial proof that a is equal to itself, i.e., the proof is $\mathit{refl}(a) : a = a$. Notice that the single case in such an induction corresponds to the single way of introducing elements of equality types via rule E2, and compare that with P4, which dealt with the two ways of introducing elements of $\mathbb{N}$. Intuitively, induction for equality corresponds to the statement that, in the absence of variables and axioms, any formal expressions representing a type X, a pair of elements a and b of X, and a proof p of the equality $a = b$, reduce, after expansion of definitions, to the proof $\mathit{refl}(a)$ of $a = a$, so that is the only case that an algorithm corresponding to the definition of f needs to handle.

The mathematical reader who accepts Peano's rules as valid may feel uneasy about the validity of Martin-Löf's rules for equality types. Peano's rules are easily accepted as valid, because they describe things we know about, natural numbers, and they posit inference rules which appear to be valid. Proofs of equality, on the other hand, are not things we know about, at least, not as mathematical objects in their own right. The informal way to justify them is to repeat that the smallest reflexive relation on the elements of a set X is equality, so everything true about equality ought to flow from reflexivity. Formally, one way to proceed is to justify the consistency of the formal system by providing an interpretation of the entire formal system in a suitable mathematical structure. The justification of type theory with the univalence axiom in that way one of the goals of a project of Voevodsky; we cite references in section 7.

For now, until we encounter univalence and its consequences, the reader may consider the following hints pointing toward an interpretation (in classical mathematics) of this formal system. In that interpretation, every type X is interpreted as a set $|X|$, and an element $a : X$ is interpreted as an element $|a| \in |X|$. A function type $X \to Y$ is interpreted as the set of functions from $|X|$ to $|Y|$. The type $\mathbb{N}$ is interpreted as the set $|\mathbb{N}|$ of natural numbers. The type $a = b$ is interpreted as the one-point set $\{*\}$ if $|a| = |b|$, and is interpreted as the empty set $\{\}$ if $|a| \neq |b|$. The element $\mathit{refl}(a) : a = a$ is interpreted as the element $*$ of the one-point set. Using this visualization may provide some temporary intuition for Martin-Löf's rules. The interpretation is not required to be faithful, except in the sense that the empty type (to be introduced below) should be interpreted as the empty set. Then if the formal system allowed an absurdity to be proved, the corresponding element a of the empty type would give an element $|a|$ of the empty set, providing a contradiction in set theory.

Later, when univalence is introduced, the mathematical interpretation will have to be modified so types are interpreted not as sets, but as topological spaces, to handle the types that are not sets.

We may use induction to prove symmetry of equality. In accordance with our discussion of implication above, we show how to produce an element of $b = a$ from an element e of $a = b$, for any b and e. By induction (letting $P(b, e)$ be $b = a$ in rule E3 above), it suffices to produce an element of $a = a$; we choose $\mathit{refl}(a)$ to achieve that.

Transitivity of equality is established the same way. For each $a, b, c : X$ and for each $p : a = b$ and for each $q : b = c$ we want to produce an element of type $a = c$. By induction on q we are reduced to the case where c is b and q is $\mathit{refl}(b)$, and we are to produce an element of $a = b$. The element p serves the purpose. Notice the similarity of this inductive definition with the definition given above of the sum $m + n$.

Associativity of transitivity is established the same way. We leave its proof as an exercise.

Now let's consider how we might formulate our symmetry result. One way would be to assert that we have proved the following lemma.

Lemma 3.1. *For any type X and for any $a, b : X$, $(a = b) \to (b = a)$.*

Aside from using $\to$ instead of $\Longrightarrow$ for the "implication", it looks traditional, but there is a drawback that is perhaps not immediately apparent: standard practice in current mathematical prose is to regard the statement of a lemma as sufficient for application of the lemma, and to regard the proof (once it has been provided and verified) as irrelevant for the application of the lemma[10]. In this case, the construction of the symmetry function that we have provided above is not revealed in the statement of the lemma, but only in the proof of the lemma. Having the precise function available for perusal might be important if $b = a$ has more than one proof. Thus a better formulation would be as a definition, as follows.

Definition 3.2. For any type X and for any $a, b : X$, let $\mathit{symm}_{a,b} : (a = b) \to (b = a)$ be the function defined by induction by setting $\mathit{symm}_{a,a}(\mathit{refl}(a)) := \mathit{refl}(a)$.

Similarly, transitivity is best formulated as an inductive definition $\mathit{trans}_{a,b,c} : (a = b) \to ((b = c) \to (a = c))$. We may abbreviate $(\mathit{trans}_{a,b,c}(p))(q)$ as $p * q$.

Given 4 points $a, b, c, d : X$ and 3 proofs $p : a = b$, $q : b = c$, $r : c = d$ of equality, we may produce two proofs that $a = d$, namely $(p * q) * r$

[10] I don't mean to imply that the proof is useless for the mathematical reader, who may learn something from it, or who may modify it to prove something else.

and $p*(q*r)$. As elements of the same type, we may consider the type $(p*q)*r = p*(q*r)$. An element of it is a proof of associativity, and it may be constructed by induction.

One frequent use of equality proofs is in *substitution*. Let X be a type, and let $P(x)$ be a family of types depending on a parameter $x : X$. Suppose $a, b : X$ and $e : a = b$. Then there is a function of type $P(a) \to P(b)$. To prove that, it suffices, by induction, to consider the case where e is $\mathit{refl}(a)$ and to provide a function $P(a) \to P(a)$. To achieve that, we provide the identity function.

In the case where each $P(a)$ is a proposition (to be defined later), we may say that the truth of $P(a)$ is invariant under substitution. In other words, equal elements have the same properties. In the case where each $P(a)$ is not assumed to be a proposition, we may say that the statements of type theory are invariant under equality, i.e., whether an element of a type exists is unchanged by replacing one of the parameters in the expression for the type by something equal to it.

4. Other types

There are other examples of types that are conveniently presented as inductive definitions, in the style we have seen with the natural numbers and the equality types. We present three examples.

Firstly, there will be the "empty" type, called $\emptyset$, defined inductively, with no way to construct elements provided in the inductive definition. The inductive principle for $\emptyset$ says that to prove something about (or to construct something from) every element of $\emptyset$, it suffices to consider no special cases (!). Hence, every statement about an arbitrary element of $\emptyset$ can be proven. As an example, we may prove that any two elements x and y of $\emptyset$ are equal by using induction on x.

An element of $\emptyset$ will be called an *absurdity*, and the negation $\neg P$ of a proposition P will be implemented as the function type $P \to \emptyset$. This is sensible, because an element of $\neg P$ could be applied to an element of P to produce an element of $\emptyset$, i.e., an absurdity.

Another appropriate name for $\emptyset$ is *False*.

We may also construct a function $\mathit{False} \to X$, for any type X, by induction, showing that from an absurdity anything follows.

To encode the property that X has no elements we use the type $X \to \emptyset$. To encode the property that elements $a, b : X$ are not equal, we use the type $(a = b) \to \emptyset$, and we let $a \neq b$ denote it.

Secondly, there will also be a type called *True*, defined inductively and provided with a single element *triv*; (the name *triv* comes from the word "trivial"). Its induction principle states that, in order to prove

something about (or to construct something from) every element of *True*, it suffices to consider the special case where the element is *triv*. As an example, we may prove, for any element $u : \mathit{True}$, that $u = \mathit{triv}$, by using induction to reduce to proving $\mathit{triv} = \mathit{triv}$, a proof of which is provided by $\mathit{refl}(\mathit{triv})$. One may also prove that any two elements of *True* are equal by using induction twice.

There is a function $X \to \mathit{True}$, for any type X, namely: $a \mapsto \mathit{triv}$. This corresponds, for propositions, to the statement that an implication holds if the conclusion is true.

The name of *True* is appropriate, because if P is a proposition, a function of type $\mathit{True} \to P$ could be applied to the element *triv*, yielding an element of P, thus proving P.

Thirdly, there will be a type called *Bool*, defined by induction and provided with two elements, *yes* and *no*. One may prove by induction that any element of *Bool* is equal to *yes* or to *no*.

We may use substitution to prove $\mathit{yes} \neq \mathit{no}$. To do this, we introduce a family of types $P(b)$ parametrized by a variable $b : \mathit{Bool}$. Define $P(\mathit{yes}) := \mathit{True}$ and define $P(\mathit{no}) := \mathit{False}$. The definition of $P(b)$ is motivated by the expectation that we will be able to prove that $P(b)$ and $b = \mathit{yes}$ are equivalent. If there were an element $e : \mathit{yes} = \mathit{no}$, we could substitute *no* for *yes* in $\mathit{triv} : P(\mathit{yes})$ to get an element of $P(\mathit{no})$, which is absurd. Since e was arbitrary, we have defined a function $(\mathit{yes} = \mathit{no}) \to \emptyset$, establishing the claim.

In the same way, we may use substitution to prove that successors of natural numbers are never equal to 0, i.e., for any $n : \mathbb{N}$ that $0 \neq S(n)$. To do this, we introduce a family of types $P(i)$ parametrized by a variable $i : \mathbb{N}$. Define P recursively by specifying that $P(0) := \mathit{True}$ and $P(S(m)) := \mathit{False}$. The definition of $P(i)$ is motivated by the expectation that we will be able to prove that $P(i)$ and $i = 0$ are equivalent. If there were an element $e : 0 = S(n)$, we could substitute $S(n)$ for 0 in $\mathit{triv} : P(0)$ to get an element of $P(S(n))$, which is absurd. Since e was arbitrary, we have defined a function $(0 = S(n)) \to \emptyset$, establishing the claim.

Type theory provides *sums* of types. By this we mean if X is a type and $Y(x)$ is a family of types indexed by a parameter x of type X, then there will be a type $\sum_{x:X} Y(x)$ whose elements are the pairs (a, b), where $a : X$ and $b : Y(a)$. This is reminiscent of the sum in set theory, which is implemented by the disjoint union $\amalg_{x:X} Y(x)$. Sums may be implemented by an inductive definition.

The sum $\sum_{x:X} Y(x)$ behaves as one would expect (except when X is a type that is not a set, because then the type X behaves like a topological space, the family $Y(x)$ behaves like a family of spaces varying

continuously in the parameter x, and the sum behaves like the total space of the family.)

Using sums we may encode the fibers of a function. Given a function $f : X \to Y$ and an element $y : Y$, the fiber $f^{-1}(y)$ consists of points x such that $f(x) = y$. This is encoded by defining $f^{-1}(y) := \sum_{x:X}(f(x) = y)$. In other words, a point of the fiber is a pair (x, e) consisting of the point x and a proof e of the equation $f(x) = y$.

There is a binary sum operation on types: for any types X and Y, there is a type $X \amalg Y$, which is analogous to the disjoint union of two sets. From an element of X or an element of Y we can produce an element of $X \amalg Y$. The binary sum can be implemented as a sum where the index type is *Bool* and the family of types sends *yes* to X and sends *no* to Y, or it may be implemented as an inductive definition. Binary sums behave as one would expect.

Our type theory will also contain *products* of types. By this we mean if X is a type and $Y(x)$ is a family of types indexed by a parameter x of type X, then there will be a type $\prod_{x:X} Y(x)$ whose elements serve as *families* of elements $b : Y(a)$, one for each $a : X$. A family is much like a function, but without a single codomain. This is reminiscent of the product in set theory, which uses the same notation.

Products behave as one would expect (except when X is a type that is not a set, because then the type X behaves like a topological space, the family $Y(x)$ behaves like a family of spaces varying continuously in the parameter x, and the product behaves like the space of continuous sections of the family.)

There is a binary product operation on types: for any types X and Y, there is a type $X \times Y$. From an element of X and an element of Y we can produce an element (x, y) of $X \times Y$. The binary product can be implemented as an inductive definition, as a special case of products, or as a special case of sums. Binary products behave as one would expect.

5. Formalization of mathematics

With equality available, with all its expected properties, we may encode some elementary mathematical properties as types, to show how such encoding goes in practice, as implemented (approximately) in the *UniMath* project at `http://UniMath.org/` and as exposed by Voevodsky in [41], as in the *HoTT* project [4] as exposed in [9], as in the *HoTT-Agda* project [1], and as in the *Lean 0.2* theorem prover[2].

Let X be a type. The property that X has at most one element is equivalent to the property that any two elements are equal, so is

encoded by $\prod_{a:X} \prod_{b:X} (a = b)$. The property that X has exactly one element is equivalent to having an element such that every other element is equal to it; hence it is encoded by $\sum_{a:X} \prod_{b:X} (a = b)$.

Now let us consider how to formalize the mathematical notion of group. More precisely, we adopt as our goal the definition of a type, $Group$, whose elements are the groups, whose existence was advertised in the introduction.

Following a traditional approach, one may choose to define a group as a tuple (G, e, i, m), where G is a set, e is the unit element, i is the inverse operation, and m is the multiplication operation; one also asserts the truth of various properties of the operations, such as associativity.

Using sums, we may define a 4-tuple (G, e, i, m) as an iterated pair of the form $(G, (e, (i, m)))$, and we may consider defining $Group$ to be the type of such 4-tuples.

Well, almost: we need to specify a type whose elements will provide the candidates for G, as the notion of sum requires. For that purpose, we introduce a *universe*, U_0; its elements will correspond to "small" types. The notion of "universe" seems to have first arisen in the formalization of set theory by Bernays [11], building on work of von Neumann, in which there are three types of thing: propositions, sets, and classes. One of the classes has all of the sets as its elements, and it is analogous to our U_0. One could extend their system by introducing *hyperclasses*, one of which would have all of the classes as its elements, and we could regard it as analogous to a bigger universe U_1, one of whose elements is U_0. And so on. Indeed, such a chain of universes is postulated in our type theory, and the concept turns out to be of fundamental importance. Each one is a "universe" in the sense that it is closed under the operations of type theory that make new types from old ones.[11]

Having introduced universes, we may say that our tuples (G, e, i, m) will be the elements of type $\sum_{G:U_n} \sum_{e:G} \sum_{i:G\to G} (G \times G \to G)$. Here n is a fixed universe level, and we are engaged in specifying a type to encode the groups that live in U_n.

In order to formalize the assertion that the operations satisfy various properties, we recall that all such propositions are to be re-implemented as types and their proofs are to be re-implemented as elements of those types. So we extend our 4-tuple to a 9-tuple $(G, e, i, m, \alpha, \lambda, \rho, \lambda', \rho')$,

[11]The interpretation in classical mathematics that we discussed above for verifying consistency of our formal system will need to be adjusted to provide interpretations of the universes. For that purpose one assumes that there is an ascending chain of Grothendieck universes $|U_0| \in |U_1| \in |U_2| \in \cdots$, and one trusts that doesn't add an inconsistency to set theory.

where α is a proof of associativity, λ is a proof of the left unit law, ρ is a proof of the right unit law, λ' is a proof of the left inverse law, and ρ' is a proof of the right inverse law.

Well, almost. The associativity property is formulated in set theory as

$$\forall_{a:G}\forall_{b:G}\forall_{c:G} m(m(a,b),c) = m(a,m(b,c)),$$

so we need a type that can represent universal quantifiers. The product type will serve that purpose, so we can use

$$\prod_{a:G}\prod_{b:G}\prod_{c:G} m(m(a,b),c) = m(a,m(b,c)).$$

We will take α to be an element of that type; in other words, it will be a function that provides, for each a, b, c a proof of the equation $m(m(a,b),c) = m(a,m(b,c))$. The other four properties are implemented in a similar way.

Now consider the possibility that we have two unequal proofs, α and α', of associativity. Then we would get two unequal groups,

$$(G, e, i, m, \alpha, \lambda, \rho, \lambda', \rho')$$

and

$$(G, e, i, m, \alpha', \lambda, \rho, \lambda', \rho'),$$

and that would be an unintended departure from traditional mathematics. This is a consequence of the necessity to include the proofs of the properties in the tuple, because we are trying to define the type whose elements are the groups in our formal language, which provides no way to assert the properties of a group "on the side". Thus we are led to insist that there is a proof of $\alpha = \alpha'$, as a consequence of which it would follow that our two groups are equal. Each of the functions α and α' assigns to any $a, b, c : G$ a proof of $m(m(a,b),c) = m(a,m(b,c))$, and functions with unequal values can't be equal, so we will need to be able to prove that any two proofs of $m(m(a,b),c) = m(a,m(b,c))$ are equal. For this purpose it would be most convenient if it were true more generally that any two proofs of an equation $x = y$ between any two elements x and y of G are equal.

Thus we are led to embark on a diversion about types any two elements of which are equal. In informal mathematical speech and in set theory, any proof of a proposition is irrelevant, as it tells us only that the proposition is true: it provides no information that is needed later on. Good mathematical prose is formulated to follow that practice by relocating information needed later on from the proof to the statement being proven. That and the preceding paragraph motivate our definition of "proposition" in this context. A *proposition* is a type such that

any two elements of it are equal. The statement that a type P is a proposition is encoded by the type $isProp(P) := \prod_{a:P} \prod_{b:P} (a = b)$.

Statements proven previously can be rephrased as saying that *False* and *True* are propositions.

Suppose P and Q are propositions. The implication $P \implies Q$ can be implemented as the type $P \to Q$. An element of it will transform any proof of P into a proof of Q. The conjunction $P \wedge Q$ can be implemented as the product $P \times Q$. However, the disjunction $P \vee Q$ cannot be implemented as the sum $P \amalg Q$, because that type will have two different proofs if both P and Q have proofs. We will return to this issue later on.

We may define the universal quantifier

$$\forall_{x:X} P(x) := \prod_{x:X} P(x)$$

for a family of propositions $P(x)$ depending on a parameter $x : X$. It is a proposition, and it has the properties one would expect.

We continue our diversion with a discussion of types X with the property that any two proofs of an equation $a = b$ between any two elements a and b of X are equal, motivated by our need for that to be true of G. With our definition of proposition in hand, we can rephrase the condition on X as positing that the equality type $a = b$ between any two elements $a, b : X$ is a proposition. A type X with that property is defined by Voevodsky to be a "set". Let us use the notation $isSet(X)$ to denote the type encoding this condition.

It can be proven that the propositions are the sets with at most one element, and that many types, including *Bool* and $\mathbb{N}$, are sets.

Ending our diversion and returning to our formalization of the notion of "group", we see that we need G to be a set. Since we can't postulate that G is a set "on the side", we have to add a proof of it to the tuple. Thus our final definition of "group" is that it is a 10-tuple

$$(G, e, i, m, \alpha, \lambda, \rho, \lambda', \rho', \iota),$$

where ι is a proof that G is a set, i.e., it is an element of the type $isSet(G)$.

Now consider the possibility that we have two unequal proofs, ι and ι', that G is a set. Then we would get two unequal groups,

$$(G, e, i, m, \alpha, \lambda, \rho, \lambda', \rho', \iota)$$

and

$$(G, e, i, m, \alpha, \lambda, \rho, \lambda', \rho', \iota'),$$

and that would be an unintended departure from traditional mathematics. If we were able to prove that $\iota = \iota'$, then our two groups would be equal. Luckily, it is a theorem of Voevodsky that, for any type X, the type $isSet(X)$ is a proposition.[12] Applied to G, it shows that $\iota = \iota'$, implying in turn that our two groups are equal. This fortuitous foundational result shows the feasibility of the approach.

We now define $Group$ to be the type consisting of such 10-tuples; its elements are the groups living in U_n. For precision about the universe, one may use a subscript, as in $Group_n$.

It should now be evident to the reader how to follow the pattern established above and formalize the definitions of the types of many sorts of mathematical objects, such as of Abelian groups, monoids, rings, commutative rings, and fields. For example, the type Set of all sets in U_n will consist of the pairs (X, ι), where X is an element U_n, and ι is a proof that X is a set. As above, one sees that two proofs that X is a set yield equal elements of Set.

Now let's consider the problem of formalizing the notion of G-torsor. By definition, it is a nonempty set X with a free transitive[13] left action by G. For example, if G is a subgroup of a group H, then the cosets of G in H are G-torsors. We may represent a G-torsor as a tuple

$$(X, m, \alpha, \lambda, \iota, \tau, \nu),$$

where $m : G \times X \to X$ is the action, and the remaining components represent the associativity property, the left unit property, the proof that X is a set, the proof that G acts freely and transitively on X, and the proof that X is nonempty[14].

Well, almost: what type does the proof ν of nonemptiness of X belong to? The only type available for the task seems to be X itself: suppose we were to use it. Let ν and ν' be unequal elements of X. Then we would get two unequal G-torsors,

$$(X, m, \alpha, \lambda, \iota, \tau, \nu)$$

[12]Voevodsky's construction of that element uses an axiom called "functional extensionality". Luckily, as he proves, that axiom is consistent with the rest of theory, so it may safely be used in formalizations. It is also consistent with the univalence axiom; indeed, it is a consequence of it.

[13]One says that G acts *freely* on X if $gx = x$ implies $g = e$. One says that G acts *transitively* on X if, for all $x, y \in X$, there is some $g \in G$ such that $gx = y$. Thus G acts freely and transitively on X if, for every $x \in X$, the function of type $G \to X$ given by $g \mapsto gx$ is a bijection. The conjunction of these two conditions is equivalent to the map $G \times X \to X \times X$ provided by $(g, x) \mapsto (gx, x)$ being a bijection, which is why these two conditions are usually stated together.

[14]Nonemptiness of X does not follow from the other properties.

and

$$(X, m, \alpha, \lambda, \iota, \tau, \nu'),$$

and that would be a departure from traditional mathematics: two proofs of nonemptiness of the set underlying a torsor should not give two different torsors. Moreover, equipping a torsor with a point renders the torsor canonically trivial. In set theory there is a difference between a pointed set and a nonempty set, and we need a formal way to express that difference in type theory.

In order to ensure that two proofs of nonemptiness (such as ν and ν') are always equal, we need them to be elements of a *proposition*, to be called $\|X\|$, say, and to be called the *propositional truncation* of X. Any element of X should provide a proof of nonemptiness, so there should be a map $\mu : X \to \|X\|$. One way to implement propositional truncation is to add it to the language and justify it later, as is done in [8]. One could provide the following more economical definition of it, as is done in [49].

For any type X in U_n, Voevodsky defines

$$\|X\| := \prod_{P:U_n} (isProp(P) \to ((X \to P) \to P)).$$

It will follow from univalence that it is a proposition. The map μ can be implemented as $x \mapsto P \mapsto i \mapsto f \mapsto f(x)$. Moreover, it has a universal property: any map $g : X \to Q$ to a proposition Q factors through μ; the factorization map is provided by $w \mapsto ((w(P))(j))(g)$, where j is the proof that Q is a proposition.[15]

Returning to our formalization above of the type of all G-torsors, we may now specify that the final component ν of the tuple is to be of type $\|X\|$, thereby completing the formalization.

We may use propositional truncation to define the disjunction of two propositions as $P \vee Q := \|P \amalg Q\|$. It is a proposition and has the properties one would expect.

[15]The definition is quantified over a variable P in the universe U_n, so $\|X\|$ lies not in U_n, but in U_{n+1}. Voevodsky's proposed system includes two "resizing" axioms to fix this problem, one of which ensures that, for any proposition Q, $Q : U_n$ follows from $Q : U_{n+1}$. He believes it likely that the two axioms are consistent with the rest of the system, but the interpretation described below does not validate them. If such validation is unavailable, one may instead implement propositional truncation using higher inductive types, as in [36, Section 6.9], at the expense of providing a proof that the interpretation described below validates them.

We may use propositional truncation to define the existential quantifier

$$\exists_{x:X} P(x) := \| \sum_{x:X} P(x) \|$$

for a family of propositions $P(x)$ depending on a parameter $x : X$. It expresses the statement that there is some element x so that $P(x)$ is true, without providing x. It is a proposition and has the properties one would expect.

We encode surjectivity of a function $f : X \to Y$ as the type $\forall_{y:Y} \| f^{-1}(y) \|$. The type $\prod_{y:Y} f^{-1}(y)$ would not provide a suitable encoding, since it is the type of sections of f.

6. Univalence

At this point, we are faced with a design decision: whether to arrange for some of our identity types $a = b$ to have multiple elements. Additional equalities would be useful, because they can be used in substitutions. Alternatively, if classical mathematics were our only guide, we would be led to introduce an axiom that asserts that every type is a set. That wouldn't lead to a contradiction, but it wouldn't be essential, because the classical constructions of new sets that don't involve Grothendieck universes, when formalized in type theory, yield types that can be proved to be sets anyway, as Voevodsky has shown.

Let us consider the identity types $X = Y$ corresponding to types X and Y in the same universe, U_n say. The only obvious statement about such types is that $X = X$ has an explicit and trivial proof: *refl*(X).

We motivate a desire for further proofs of equality between types as follows.

In set theory, the equation $\mathbb{N} = \{x \in \mathbb{Z} \mid x \geq 0\}$ is false, if one uses the usual definition that $\mathbb{Z}$ is a certain quotient set of $\mathbb{N} \times \mathbb{N}$. The statement is false because the elements of the two sets are different. The strongest thing that can be said is that there is an isomorphism $\mathbb{N} \simeq \{x \in \mathbb{Z} \mid x \geq 0\}$. Nevertheless, a mathematician may *identify* the two sets and expect not to get into trouble.

In set theory, the associativity $(X \amalg Y) \amalg Z = X \amalg (Y \amalg Z)$ of binary sums is false, if one uses the usual definition that $X \amalg Y$ is the set of pairs (i, z) where $i = 0$ or $i = 1$, $z \in X$ if $i = 0$, and $z \in Y$ if $i = 1$. The statement is false because the elements of the two sets are different. The strongest thing that can be said is that there is a natural isomorphism $(X \amalg Y) \amalg Z \simeq X \amalg (Y \amalg Z)$. Nevertheless, a mathematician may *identify* the two sets and expect not to get into trouble.

The two examples above can also be formulated in type theory, and the two types in either example might be equal, because the formal language of type theory is restricted, preventing us from forming the statement that the elements of one type are unequal to all the elements of another type. Hence we may entertain the possibility that $\mathbb{N}$ and $\{x \in \mathbb{Z} \mid x \geq 0\}$ are equal, and that binary sums are associative. We may even try to re-engineer our formal system to make it happen.

Here's how Voevodsky made it happen.

A function $f : X \to Y$ between two types is called an *equivalence* if, for each $y : Y$, the fiber $f^{-1}(y)$ has exactly one element. It can be proven that being an equivalence is a proposition. The interested reader may encode this property as a type by applying encodings previously introduced. Thus the equivalences are the elements of a type, which is denoted by $X \simeq Y$. This definition has almost all of the properties one would expect for the appropriate notion of isomorphism between two types. For example, one can prove that the identity function $X \to X$ is an equivalence, that an equivalence has an inverse function that is also an equivalence, and that a function with an inverse function is an equivalence.

For types $X, Y : U_n$, we may define a function $\Phi_{X,Y} : (X = Y) \to (X \simeq Y)$ by induction: in the case where Y is X, it sends $\mathit{refl}(X)$ to the identity equivalence. This gives a relationship between equalities and equivalences.

Voevodsky's Univalence Axiom is stated as follows.

Axiom 6.1 (Univalence)**.** *The map* $\Phi_{X,Y} : (X = Y) \to (X \simeq Y)$ *is an equivalence.*

This axiom provides a way to promote equivalences into equalities, by application of the inverse of $\Phi_{X,Y}$, thereby satisfying the desires motivated above.

Now we address an issue of terminology. From this point on, the reader may prefer to think of an element of $X = Y$ not as providing a proof that the types X and Y are *equal*, but as providing a way to *identify* X with Y. Different elements will provide different ways. This alternative reading may relieve some discomfort for readers who prefer to preserve the word "equality" for something that can happen in just one way. In line with that consideration, we will refer to the type $X = Y$ as an *identity type*, rather than as an *equality type*, and we will refer to its elements as *identities* or *identifications*.

For example, a bijection $\theta : \mathbb{N} \simeq \{x \in \mathbb{Z} \mid x \geq 0\}$ provides, via univalence, a proof θ' that $\mathbb{N} = \{x \in \mathbb{Z} \mid x \geq 0\}$, which in turn allows us to use θ' to *identify* $\mathbb{N}$ with $\{x \in \mathbb{Z} \mid x \geq 0\}$. This way of putting

it is common mathematical practice. What's new is support for the practice in the language of logic.

Let us pause to infer an important foundational consequence. We know that the statements of type theory are invariant under identity (by substitution). With the ability to promote equivalences to identities, we see that the statements of type theory are invariant under equivalence, thereby fulfilling the mathematicians' dream: one cannot express a property in our formal language that fails to be invariant under equivalence. That invariance is a strong and useful principle, which flows from the precise way that the formal language has been restricted to the minimum required for formalization of mathematics.

Pairs of sets often have more than one isomorphism between them. Hence, in the presence of the univalence axiom, pairs of types often will have more than one proof of identity, and thus the universes are not sets. Moreover, in the absence of univalence and of axioms contradicting it the universes cannot be proved to be sets. This is a major difference from set theory.

A consequence of the Univalence Axiom is that isomorphism between groups is the same thing as identity. In other words, given two groups $G, H : Group$, let $G \cong H$ denote the type of isomorphisms between G and H. Then the map $G = H \to G \cong H$ is an equivalence. An analogous statement can be proved for all the other standard mathematical structures: partially ordered set, G-torsor, monoid, ring, etc. This general principle is called the *structure identity principle* in the book [36, section 9.8]. The main point in the proof of it is that the notion of isomorphism between two instances of an algebraic structure pays attention to every element of the structure, so the elements of either structure are uniquely determined by the equivalence on the underlying types and the elements of the other structure.

The structure identity principle can be used to prove, for example, that if X and Y are partially ordered sets, x is a minimal element of X, and $f : X \simeq Y$ is an isomorphism of partially ordered sets, then $f(x)$ is a minimal element of Y. The same proof works for maximal elements: neither proof needs to appeal to the definition of "minimal element" or "maximal element", because the principle applies to all properties expressible in our formal language. By contrast, in set theory, to prove those assertions, the definitions of minimal element and of maximal element must be examined and used, since the more permissive language of set theory permits statements to be written that contain mathematical irrelevancies; for an example, consider the proposition $x = 2$, which does not imply $f(x) = 2$.

Traditionally, the objects of a category have constituted a set, but the natural way to formalize the *category* of groups is with its objects being the elements of the *type* $Group$. The best categories are the ones (such as this one) where the identities in the type of objects are equivalent to the isomorphisms, for it is such categories that fulfill the category theorists' dream: to work in a mathematical language where one cannot express a property that fails to be invariant under isomorphism[16]; such categories are called *univalent*. See [5] for a procedure that renders categories univalent in a universal way.

On the other hand, sometimes one needs a category whose objects form a set; for example, one may wish to construct its geometric realization as a topological space. If so, then one may incorporate enough extraneous data in each object of the category to make proofs of identity between two objects unique. For example, one may equip each object in the category of finite dimensional vector spaces with an ordered basis – that is enough, because isomorphisms respecting the bases are unique.

As we said above, in the presence of univalence the type $Group$ is not a set, but there is something positive to be said about it. Given two groups $G, H : Group$, the type $G = H$ is a set, because the map the map $G = H \to G \cong H$ is an equivalence of it with a set.

This motivates another fundamental definition of Voevodsky's, which unifies and simplifies the formalization of the basic theorems of the subject. We define by induction on a natural number n what it means to say that a type X has *h-level (at most)* n: (1) it has h-level 0 if it has exactly one element; (2) it has h-level $n+1$ if for any elements a and b of X, the type $a = b$ has h-level n. Voevodsky has proven that the types of h-level 1 are the propositions, and then one can see from the definition that the types of h-level 2 are the sets. We see also that $Group$ is of h-level 3, which is the natural level for the type of objects of any groupoid. Properly defined, the type of categories will be of h-level 4, because the groupoid of categories is a 2-groupoid: an equivalence between two categories may have automorphisms. In general one expects the objects of an *n-groupoid* to be of h-level $n+2$.

Voevodsky points out a stratification of all of mathematics into levels that follows from the stratification of types by h-level: *element-level*

[16]For example, Makkai [25] has formulated the dream this way, as a requirement to be satisfied by a future "Structuralist Foundation of Abstract Mathematics" (SFAM): "SFAM adopts isomorphism of objects in a category ... to play the role of equality for those objects, in the sense that ... all grammatically correct properties of objects of a fixed category are to be invariant under isomorphism."

mathematics concerns equations between elements of sets, or properties of elements of sets; *set-level* mathematics concerns isomorphisms between algebraic structures, or properties of algebraic structures invariant under isomorphism; *groupoid-level* mathematics concerns equivalences of groupoids, or properties of groupoids invariant under equivalence of groupoids; and so on [41].

Another common mathematical practice is to speak of "natural" or "canonical" constructions; for example, one often states that there is no natural isomorphism between a vector space V and its dual vector space V^*. This practice is also directly supported now by logic, provided we consider naturality only with respect to isomorphisms $V \cong V'$, which, by the structure identity principle, are captured by identities, which in turn may be used in substitutions. We may formalize the type of isomorphisms between a vector space and its dual as the type $\prod_{V:Vect}(V \cong V^*)$, where $Vect$ is the type of vector spaces over a field k in universe U_n. In the presence of univalence, one may prove that the type of such families of isomorphisms is empty.[17]

7. The interpretation

The possibility that an identity type $a = b$ has multiple elements presents a problem for the mathematical reader who wants to understand the situation, as well as a problem for the reader who has perused the interpretation for our formal system in classical mathematics that we sketched above. That interpretation is incompatible with the Univalence Axiom, because, roughly speaking, for a set $X : U_n$ with more than one element it interprets the map $\Phi_{X,X} : (X = X) \to (X \simeq X)$ as a function from the one point set to the set of permutations of $|X|$; the function sends the single point $*$ to the identity bijection. The function is not a bijection, as would be required to interpret the Univalence Axiom.

The idea for repairing this is to interpret each type X as a topological space[18] $|X|$, to interpret an element $a : X$ as a point $|a|$ of the space $|X|$, to interpret a proof of $a = b$ as a path connecting $|a|$ to $|b|$ in $|X|$, and to interpret the type $a = b$ as the space of all such paths. Symmetry of identity passes to reversal of paths in the interpretation, and transitivity of identity passes to concatenation of paths. A family $Y(x)$ of types parametrized by a variable $x : X$ will be interpreted by

[17] Compare with the discussion in [29] or the use of the word "natural" in the proof of [36, Theorem 3.2.2].

[18] The topological spaces used in the interpretation are actually fibrant simplicial sets.

a fibration[19] $E(Y) \to |X|$ whose fibers over the points $|a| \in |X|$ arising from elements $a : X$ are the spaces $|Y(a)|$. Induction for identity will be interpreted by the lifting property that characterizes fibrations, as observed in [7]. One assumes one is given an ascending sequence of Grothendieck universes $V_0 \in V_1 \in \ldots$ and one interprets the universe U_n by the space $|U_n|$ of all spaces in V_n; it is a space in V_{n+1}. For $n \geq 0$, types of h-level $n+2$ are interpreted by spaces whose homotopy groups vanish above dimension n. Validity of the Univalence Axiom in this interpretation is a theorem of Voevodsky that shows the path lifting map from the space of paths between two points of $|U_n|$ to the space of homotopy equivalences between the corresponding fibers of the universal space over $|U_n|$ is a homotopy equivalence. The expository paper [21] gives the main ideas of the proof of the theorem. Full details of the correctness of the interpretation, and thus of the soundness of the theory, will appear in a series of papers planned by Voevodsky, aimed at treating a large class of formal languages, some of which are available: [46, 39, 38, 43, 42, 40, 44, 45, 48, 47]. What necessitates so many details to be expressed in print is the abundance of grammatical rules in a formal language such as this one, each one of which has an incarnation in the interpretation that must be carefully considered.

The Law of Excluded Middle and the Axiom of Choice are also validated by this interpretation, so classical mathematics is supported soundly by the univalent foundations.

8. Further developments

In addition to the focus of many on establishing the soundness of the system and on paving the way for widespread adoption of univalent foundations and the use of proof assistants by mathematicians, there are other interesting developments focusing such things as computability and the study of types in their own right.

The phrase "synthetic homotopy theory" refers to the enterprise where one regards a type as a good substitute for the classical notion of topological space, and one regards a proof of identity as a good substitute for the notion of a path between two points of a space: the goal is to see which theorems of homotopy theory have analogues that remain provable in this context. This context is a primitive one, because so few axioms are assumed to set up the theory, as we have seen, so the theorems that hold in this context will the most fundamental

[19] In topology, a continuous family of topological spaces is one that is provided by the fibers of a continuous map that is a "fibration".

ones, capable of the most generalization. A pleasant surprise is that so many theorems hold in this context.

Given a type X we define the type $\pi_0 X$ of connected components as the quotient of X by the equivalence relation $b = c$, where b and c are elements of X. Voevodsky has proved that $\pi_0 X$ is a set (as is the case for the type of equivalence classes of any equivalence relation on X).

Given a type X and a basepoint $a : X$ the loop space ΩX to be the type $a = a$, and we equip it with the basepoint $\mathit{refl}(a) : \Omega X$. Iterating n times yields a type $\Omega^n X$, and we may define $\pi_n X := \pi_0 \Omega^n X$. The proofs of transitivity and symmetry in section 3 provide $\pi_1 X$ with a group structure, and thus provide $\pi_n X$ with n group structures when $n > 0$. A basic fact of homotopy theory is that the group $\pi_n X$ is abelian for $n \geq 2$, and that the various group structures are the same. To prove that, it suffices to show that the two composition operations on the type $\Omega^2 X$ (coming from transitivity) agree and are commutative. (We don't have to say "commutative up to homotopy" here, because in our context, homotopy and identity are the same.) One standard argument, due to Eckmann and Hilton, involves showing that a monoid object in the category of monoids is commutative and the two operations coincide. That argument is formal enough that it applies here [36, Theorem 2.1.6], and gives the first indication that something wonderful is going on. The interested reader may refer to [36, Chapter 8] or to [34] for more leisurely expositions of this pursuit.

It is not yet known how to construct types that correspond to the spheres S^n in the univalent foundations (except for S^1, which can be defined as the type $B\mathbb{Z}$ of $\mathbb{Z}$-torsors). The book [36] introduces spheres by using additional basic types constructors in the formal language – they are called "higher inductive types". For example, the circle S^1 is defined inductively by declaring that there is a basepoint $s : S^1$ and that there is a loop $\ell : s = s$. The higher spheres are done in a similar way. Voevodsky's program for proving soundness of univalent foundations doesn't include higher inductive types, but others are pursuing it: see [24]. Much of the work cited below uses higher inductive types.

In [23] one may find a proof that the fundamental group of S^1 is $\mathbb{Z}$, and in [19] one may find a formalization of the Seifert–van Kampen theorem. In [20] one may find a formalization of the Blakers-Massey connectedness theorem, using higher inductive types to construct homotopy pushouts. A translation of the formal proof into classical homotopy theory is given in [31], with the expectation that it will go through for any ∞-topos. Finally, the proof is translated into the language of ∞-topoi in the paper [6]. Even better would be a meta-theorem that says that type theory serves as an internal language for higher topoi, so

manually rewriting and rechecking the proof in the new context would not be required. Progress on that dream has recently been made in the paper [22].

A type-theoretic proof that $\pi_4(S^3) \cong \mathbb{Z}/2$ is offered by the thesis [13]. The first part of the proof demonstrates the existence of a natural number n satisfying $\pi_4(S^3) \cong \mathbb{Z}/n$, and the second part demonstrates that $n = 2$. The proofs are constructive, in the sense that the use of axioms (such as the Law of Excluded Middle or the Axiom of Choice) is avoided, aside from the use of the Univalence Axiom. Although normally the use of any axiom interferes with computability, Voevodsky conjectured (albeit for a system not including higher inductive types) that the interference arising from the use of the Univalence Axiom can be bypassed. Thus one may hope to design a proof assistant that can produce the value of n by performing a computation. Fundamental progress has been made in that direction through the development of "cubical type theory" [12, 15] and of a proof assistant `cubicaltt` [14] based on it; see also the development of *RedPRL* [3]. A formal expression for the number n has been submitted to `cubicaltt` for evaluation, but the computation ran out of memory after running for 6 hours on a machine with plenty of memory, according to Brunerie. Work in that direction continues.

In [33] one may find a type theoretic proof of the Brouwer fixed point theorem, accomplished by an enhanced type theory called *real-cohesive homotopy type theory*, which supports synthetic topology side by side with synthetic homotopy theory.

References

1. *Homotopy Type Theory in Agda*, an Agda library of formalized proofs, available at `https://github.com/HoTT/HoTT-Agda`.
2. *Lean theorem prover version 0.2*, a proof assistant together with a library of formalized proofs, available at `https://github.com/leanprover/lean2/`.
3. *RedPRL*, a proof assistant for Computational Cubical Type Theory together with a library of formalized proofs, available at `http://www.redprl.org/`.
4. *The HoTT Library*, a Coq library of formalized proofs, available at `https://github.com/HoTT/HoTT`.
5. Benedikt Ahrens, Krzysztof Kapulkin, and Michael Shulman, *Univalent categories and the Rezk completion*, Math. Structures Comput. Sci. **25** (2015), no. 5, 1010–1039. MR 3340533

6. Mathieu Anel, Georg Biedermann, Eric Finster, and André Joyal, *A Generalized Blakers-Massey Theorem*, (2017), (preprint available at arXiv:1703.09050).
7. Steve Awodey and Michael A. Warren, *Homotopy theoretic models of identity types*, Math. Proc. Cambridge Philos. Soc. **146** (2009), no. 1, 45–55, (preprint available at arXiv:0709.0248). MR 2461866
8. Steven Awodey and Andrej Bauer, *Propositions as [types]*, J. Logic Comput. **14** (2004), no. 4, 447–471. MR 2081047
9. Andrej Bauer, Jason Gross, Peter LeFanu Lumsdaine, Mike Shulman, Matthieu Sozeau, and Bas Spitters, *The HoTT Library: A formalization of homotopy type theory in Coq*, (2016), (preprint available at arXiv:1610.04591).
10. A. Beilinson, *I. M. Gelfand and his seminar—a presence*, Notices Amer. Math. Soc. **63** (2016), no. 3, 295–298, (preprint available at arXiv:1505.00710). MR 3445168
11. Paul Bernays, *A system of axiomatic set theory. Part I*, J. Symbolic Logic **2** (1937), 65–77.
12. Marc Bezem, Thierry Coquand, and Simon Huber, *A model of type theory in cubical sets*, 19th International Conference on Types for Proofs and Programs, LIPIcs. Leibniz Int. Proc. Inform., vol. 26, Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2014, pp. 107–128. MR 3281415
13. Guillaume Brunerie, *On the homotopy groups of spheres in homotopy type theory*, (2016), (preprint available at arXiv:1606.05916).
14. Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg, *Experimental implementation of cubical type theory*, the proof assistant `cubical`, written in Haskell, available at `https://github.com/mortberg/cubicaltt`.
15. ———, *Cubical Type Theory: a constructive interpretation of the univalence axiom*, (2016), (preprint available at arXiv:1611.02108).
16. Thierry Coquand, *Théorie des types dépendants et axiome d'univalence*, Astérisque (2015), no. 367-368, Exp. No. 1085, x, 367–386. MR 3363596
17. N. G. de Bruijn, *The mathematical language AUTOMATH, its usage, and some of its extensions*, Symposium on Automatic Demonstration (Versailles, 1968), Lecture Notes in Mathematics, Vol. 125. Springer, Berlin, 1970, pp. 29–61. MR 0274219
18. Georges Gonthier, *Formal proof—the four-color theorem*, Notices Amer. Math. Soc. **55** (2008), no. 11, 1382–1393. MR 2463991
19. Kuen-Bang Hou and Michael Shulman, *The Seifert–van Kampen theorem in homotopy type theory*, Computer science logic 2016, LIPIcs. Leibniz Int. Proc. Inform., vol. 62, Schloss Dagstuhl.

Leibniz-Zent. Inform., Wadern, 2016, pp. Art. No. 22, 16. MR 3566711
20. Kuen-Bang Hou (Favonia) Hou, Eric Finster, Dan Licata, and Peter LeFanu Lumsdaine, *A mechanization of the Blakers-Massey connectivity theorem in Homotopy Type Theory*, (2016), (preprint available at arXiv:1605.03227).
21. Chris Kapulkin and Peter LeFanu Lumsdaine, *The Simplicial Model of Univalent Foundations (after Voevodsky)*, (2012), (preprint available at arXiv:1211.2851).
22. Chris Kapulkin and Karol Szumio, *Internal Language of Finitely Complete* $(\infty, 1)$*-categories*, (2017), (preprint available at arXiv:1709.09519).
23. Daniel R. Licata and Michael Shulman, *Calculating the fundamental group of the circle in homotopy type theory*, 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2013), IEEE Computer Soc., Los Alamitos, CA, 2013, pp. 223–232. MR 3323808
24. Peter LeFanu Lumsdaine and Mike Shulman, *Semantics of higher inductive types*, (2017), (preprint available at arXiv:1705.07088).
25. M. Makkai, *Towards a categorical foundation of mathematics*, Logic Colloquium '95 (Haifa), Lecture Notes Logic, vol. 11, Springer, Berlin, 1998, pp. 153–190. MR 1678360
26. Per Martin-Löf, *Hauptsatz for the intuitionistic theory of iterated inductive definitions*, Proceedings of the Second Scandinavian Logic Symposium (Univ. Oslo, Oslo, 1970), North-Holland, Amsterdam, 1971, pp. 179–216. Studies in Logic and the Foundations of Mathematics, Vol. 63. MR 0387023 (52 #7870)
27. ———, *An intuitionistic theory of types: predicative part*, Logic Colloquium '73 (Bristol, 1973), North-Holland, Amsterdam, 1975, pp. 73–118. Studies in Logic and the Foundations of Mathematics, Vol. 80. MR 0387009 (52 #7856)
28. Per Martin-Löf, *Constructive mathematics and computer programming*, Logic, methodology and philosophy of science, VI (Hannover, 1979), Stud. Logic Found. Math., vol. 104, North-Holland, Amsterdam, 1982, pp. 153–175. MR 682410
29. David McAllester, *Morphoid type theory, a typed platonic foundation for mathematics*, (preprint available at arXiv:1407.7274).
30. Ioseph Peano, *Arithmetices principia*, Fratres Bocca, 1889.
31. Charles Rezk, *Proof of the Blakers-Massey Theorem*.
32. Bertrand Russell, *Mathematical Logic as Based on the Theory of Types*, Amer. J. Math. **30** (1908), no. 3, 222–262. MR 1506041

33. Michael Shulman, *Brouwer's fixed-point theorem in real-cohesive homotopy type theory*, (2015), (preprint available at arXiv:1509.07584).
34. Michael Shulman, *Homotopy type theory: the logic of space*, (2017), (preprint available at arXiv:1703.03007).
35. The Coq Development Team, *The coq proof assistant*, Available at `https://coq.inria.fr/`.
36. The Univalent Foundations Program, *Homotopy type theory: Univalent foundations of mathematics*, `https://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.
37. Vladimir Voevodsky, *Foundations: Development of the univalent foundations of mathematics in Coq*, a Coq library of formalized proofs, available at `https://github.com/vladimirias/Foundations`.
38. ———, *B-systems*, (2014), (preprint available at arXiv:1410.5389).
39. ———, *C-system of a module over a monad on sets*, (2014), (preprint available at arXiv:1407.3394).
40. ———, *A C-system defined by a universe category*, Theory Appl. Categ. **30** (2015), no. 37, 1181–1215, (preprint available at arXiv:1409.7925). MR 3402489
41. ———, *An experimental library of formalized mathematics based on the univalent foundations*, Math. Structures Comput. Sci. **25** (2015), no. 5, 1278–1294. MR 3340542
42. ———, *Martin-Löf identity types in the C-systems defined by a universe category*, (2015), 1–51, (preprint available at arXiv:1505.06446).
43. ———, *Products of families of types in the C-systems defined by a universe category*, (2015), (preprint available at arXiv:1503.07072).
44. ———, *Lawvere theories and Jf-relative monads*, arXiv:1601.02158 (2016), 1–21, (preprint available at arXiv:1601.02158).
45. ———, *Products of families of types and* (Π, λ)*-structures on C-systems*, Theory Appl. Categ. **31** (2016), Paper No. 36, 1044–1094. MR 3584698
46. ———, *Subsystems and regular quotients of C-systems*, A panorama of mathematics: pure and applied; Conference on Mathematics and its Applications, (Kuwait City, 2014), Contemp. Math., vol. 658, Amer. Math. Soc., Providence, RI, 2016, pp. 127–137. MR 3475277
47. ———, *C-systems defined by universe categories: presheaves*, Theory Appl. Categ. **32** (2017), Paper No. 3, 53–112. MR 3607209

48. ———, *The* (Π, λ)*-structures on the C-systems defined by universe categories*, Theory Appl. Categ. **32** (2017), Paper No. 4, 113–121. MR 3607210
49. Vladimir Voevodsky et al., *UniMath: a unified approach to formalization of mathematical knowledge based on Univalent Foundations*, a Coq library of formalized proofs, available at `http://unimath.org/`.

2409 S. Vine St., Urbana, IL 61801, USA
E-mail address: `drg@illinois.edu`
URL: `http://dangrayson.com/`