

Some Wellfounded Trees in UniMath

Extended Abstract

Benedikt Ahrens^(✉) and Anders Mörtberg

Institute for Advanced Study, Princeton, USA
{ahrens,amortberg}@ias.edu

Abstract. UniMath, short for “Univalent Mathematics”, refers to both a language (a.k.a. a formal system) for mathematics, as well as to a computer-checked library of mathematics formalized in that system. The UniMath library, under active development, aims to coherently integrate machine-checked proofs of mathematical results from many different branches of mathematics.

The UniMath language is a dependent type theory, augmented by the univalence axiom. One goal is to keep the language as small as possible, to ease verification of the theory. In particular, general inductive types are not part of the language.

In this work, we partially remedy this lack by constructing some inductive (families of) sets. This involves a formalization of a standard category-theoretic result on the construction of initial algebras, as well as a mechanism to conveniently use the inductive sets thus obtained.

The present work constitutes one part of a construction of a framework for specifying, via a signature, programming languages with binders as nested datatypes. To this end, we are going to combine our work with previous work by Ahrens and Matthes (itself based on work by Matthes and Uustalu) on an axiomatisation of substitution for languages with variable binding. The languages specified via the framework will automatically be equipped with the structure of a monad, where the monadic operations and axioms correspond to a well-behaved substitution operation.

Keywords: Proof assistant · Univalent type theory · Inductive datatypes · UniMath · Initial algebras

1 Introduction

The term “UniMath”, short for “Univalent Mathematics”, refers to both a language (a.k.a. a formal system) for mathematics, as well as to a computer-checked library of mathematics formalized in that system.

The UniMath language is meant to be a “core” univalent type theory, in the sense that its set of basic, primitive, constructions is minimal. In short, it is an intensional Martin-Löf type theory with dependent sums, dependent products, identity types, natural numbers, and a universe \mathcal{U} of types for which

the univalence axiom is assumed. For a precise description, we refer to [12] and [4, Sect. 2.1]. We only point out one important definition: among all types of \mathcal{U} , some can be shown to satisfy a principle called “uniqueness of identity”. This principle can be defined (and, for some types, proved) within type theory as

$$\text{uip}(X) := \prod_{(x:X)} \prod_{(p:x=x)} p = \text{refl}(x) \ .$$

The type of all types that satisfy `uip` forms a “subuniverse” of the universe \mathcal{U} . This subuniverse is the type of objects of a category `Set` where arrows are given by type-theoretic functions; this category will be of interest in the following.

The purpose of minimality of the underlying theory of `UniMath` is twofold:

- any construction in the `UniMath` language should embed directly into any other computer proof assistant implementing a univalent type theory, in particular, into the newly developed *cubical type theory* [6, 8];
- the verification of the consistency of `UniMath` should be as easy as possible.

One language feature that has been deliberately omitted from `UniMath`, but which is usually present in other proof assistants, is *general inductive and coinductive datatypes*.

Inductive datatypes are finite data structures that can be used to store information in a systematic manner. Examples include the data structures of lists (over a given base type) and n -ary trees, but also more complicated, *heterogeneous* families of data types, used to represent programming languages with variable binding such as the lambda calculus—see [7] for such a representation.

Coinductive datatypes are *not necessarily finite* data structures. An example of such a datatype is the type of streams (a.k.a. infinite lists) of elements of a given base type.

In the semantics of type theory, inductive and coinductive types correspond to initial algebras and terminal coalgebras in a suitable sense: the usual 1-categorical notions only apply to types that are *discrete*, that is, to types that satisfy uniqueness of identity, or, equivalently, Axiom K. Indeed, the “subuniverse” of discrete types of a fixed universe of types constitutes a category (as defined in [3]), and inductive and coinductive sets are reasonably defined as initial algebras and terminal coalgebras of endofunctors on that category.

The present work is only concerned with such discrete types, and we do not attempt to construct types that are not sets.

The characterisation of inductive and coinductive types as initial algebras and terminal coalgebras is completely dual; this suggests that their syntactic behaviour is dual as well. However, that impression is somewhat deceptive:

As for *coinductive* datatypes—in the principled formulation of (indexed) `M`-types—, those are *derivable* in the `UniMath` language [2]. However, the expected computation rules only hold up to propositional identity, not judgmentally, due to the implementation of function extensionality as an axiom. We expect the computation rules to hold judgmentally for the same construction done in cubical

type theory, a type theory where function extensionality is provable instead of an axiom.

As for *inductive* types, the construction of a class of them is the subject of the work we report on in this extended abstract: In the present work, we show that a class of inductive “sets” is derivable in **UniMath** from just the “prototypical” inductive set of natural numbers. This is done by combining two results:

- a classical category-theoretic result saying that an initial algebra of an ω -cocontinuous functor can be constructed as colimit of a certain chain [1]
- the constructibility of colimits in the category of sets (a.k.a. discrete types) in **UniMath** as a consequence of the constructibility of set quotients.

The construction of set level quotients was done by Voevodsky [12]. It is a prime example of the new possibilities that the univalence axiom and its consequences provide for the formalization of (set-level) mathematics compared to the type theories implemented by **Coq** or **Agda** without the univalence axiom.

Our contribution consists of a formalization of

- some well-known theorems in category theory about existence and preservation of (co)limits
- the construction of colimits from quotients in the category of sets
- proofs that various functors are ω -cocontinuous.

and combining those results in order to obtain some inductive sets in **UniMath**.

As such, the results presented in this article are *not surprising at all*—it is our hope, however, that their formalization will be *useful*.

1.1 Related work

We do not give, in this extended abstract, a full description of the univalent foundations; instead, we point to Voevodsky’s article [12]. A brief introduction to univalent foundations is also given in [4, Sect. 2.1].

Most of the mathematics formalized in this project concerns category theory; category theory in univalent foundations has been studied in [3]. The distinction between precategories and (univalent) categories emphasized there is not of importance in our project.

A characterisation of inductive types in type theory with function extensionality (not necessarily with the univalence axiom) has been studied in [5]. In contrast to the present work, their goal is not the construction of inductive sets, but a proof of equivalence of several different definitions, given within type theory, of inductive types.

In [4], the authors formalize, in **UniMath**, the notion of “heterogeneous substitution system”, a categorical notion axiomatizing substitution for programming languages with variable binding. One of the main results formalized there says that any initial algebra of a given functor gives rise to a monad. The existence of inductive sets (or, more generally, existence of initial algebras for a given functor) is taken as a hypothesis in that result. We can plug our construction of inductive sets into that theorem and thus, e.g., obtain a monad structure on the lambda calculus, see the example below.

2 Overview of the Mathematics Formalized in This Project

In this section we give an overview of the mathematical results that we have formalized in the course of this project. It is outside the scope of this extended abstract to give a complete list; for details follow the pointers given in Sect. 3.

We also explain how those results help in the construction of inductive sets.

We are not concerned here with giving a general definition of inductive types, nor do we give a precise specification of the class of inductive types that we construct. This is due to the fact that the class is easily extensible by extending the library of mathematical theorems formalized in our library, a statement that we illustrate with the examples in this section.

2.1 Inductive Sets as Initial Algebras

In the present work, we do not attempt to construct general inductive types, but only inductive *sets*. We thus define

Definition 1. *The inductive set generated by an endofunctor $F : \mathbf{Set} \rightarrow \mathbf{Set}$ is the initial algebra of F .*

This definition is expressible within type theory, we refer to [3] for details. The use of the definite article (“*the* initial algebra”) is justified by the fact that any two initial algebras are isomorphic, and hence their carriers are propositionally equal as a consequence of the univalence axiom.

Here, the category \mathbf{Set} is the category of objects of which are discrete types of a fixed universe. It would thus be more precise to call the category $\mathbf{Set}(\mathcal{U})$, where \mathcal{U} is the fixed universe.

An example of an inductive set is the set of natural numbers, which is an initial algebra of the functor $F(X) := 1 + X$.

Another example is the inductive set of lists of elements of a fixed base set A . The defining endofunctor is given by

$$F_A(X) := 1 + A \times X .$$

There are more complicated inductive definitions, examples of which are used to model programming languages with variable binding [7]. We call those inductive families:

Definition 2. *The inductive family of sets generated by an endofunctor $F : [\mathbf{Set}, \mathbf{Set}] \rightarrow [\mathbf{Set}, \mathbf{Set}]$ is the initial algebra of F .*

An example of an inductive family is given by the datatype of untyped lambda terms with an algebraic variant of De Bruijn variables—see [7] for an explanation. The functor $A : [\mathbf{Set}, \mathbf{Set}] \rightarrow [\mathbf{Set}, \mathbf{Set}]$ specifying this datatype is given by

$$A(G) := \text{Id} + G \times G + G \circ \text{option} .$$

Here, the functor $\text{option} : \mathbf{Set} \rightarrow \mathbf{Set}$ is defined on objects by $\text{option}(X) := 1 + X$.

2.2 Existence of Initial Algebras

Obviously, not every functor as above gives rise to an initial algebra. The following theorem, due to Adámek [1], is our main tool for the construction of initial algebras.

Theorem 1. (existence of initial algebras of ω -cocontinuous functors). *Let $F : \mathcal{C} \rightarrow \mathcal{C}$ be an ω -cocontinuous endofunctor, i.e., an endofunctor that preserves colimits of chains $A_0 \rightarrow A_1 \rightarrow A_2 \rightarrow \dots$. Then, provided its existence, the colimit of*

$$0 \xrightarrow{!} F0 \xrightarrow{F!} F^2 0 \xrightarrow{F^2!} \dots$$

is an initial algebra of F .

The above theorem is applicable to construct inductive sets as in Definition 1 as per the following lemma:

Lemma 1. *The category **Set** has all colimits.*

The colimits in the category **Set** are given by the usual formula: For a diagram D , its colimit is given by

$$\left(\sum_{g:G} D(g) \right) / \sim$$

with \sim being the smallest equivalence relation containing the relation

$$(g, x) \sim' (g', y) \quad \text{iff} \quad \exists e : g \rightarrow g' \text{ with } D(e)(x) = y .$$

It is in the definition of this colimit that the univalence axiom helps: as shown by Voevodsky [12], set-level quotients of arbitrary types, but in particular of sets, modulo an equivalence relation, can be constructed in univalent type theory.

The construction of inductive families as in Definition 2 is helped by the lifting of colimits to functor categories:

Lemma 2. *If \mathcal{A} has all colimits, then also $[\mathcal{C}, \mathcal{A}]$ has all colimits.*

However, Theorem 1 only guarantuees the existence of initial algebras of ω -cocontinuous functors. This is discussed in the next section.

2.3 Preservation of Colimits

While Theorem 1 is crucial for us, its formulation and proof do not constitute the bulk of work necessary for the construction of inductive sets. Instead, it is the proof that various functors are ω -cocontinuous—hypothesis of Theorem 1—that requires most of our time and efforts.

Fortunately, there is a class of endofunctors on categories with binary products and coproducts that is built from a small set of constructions, where each

construction is, or preserves, ω -cocontinuity. This class consists of “finite polynomials” of the general form

$$F(X) = B_0 + B_1X + B_2X^2 + \dots + B_nX^n .$$

Our first example, the datatype of natural numbers, is given by such a finite polynomial, as is the second example of lists of elements of a given set A .

In order to show that such functors are ω -cocontinuous, it is sufficient to show that $F + G$ and $F \times G$ are ω -cocontinuous whenever F and G are, and that constant functors and the identity functor are ω -cocontinuous. We omit the precise statements, and pass to the more interesting situation of *heterogeneous* datatypes.

Heterogeneous datatypes are specified by endofunctors on functor categories. More specifically, what makes them heterogeneous is a “summand” of the form

$$F_K(X) := X \circ K$$

for an endofunctor K . In the case of the example of the lambda calculus, $K = \text{option}$ is the addition of a distinguished variable in the “context”.

The ω -cocontinuity of the functor F_K is ensured as long as the target category has limits. More precisely:

Theorem 2 [10, Sect. X.3]. *Let $K : \mathcal{M} \rightarrow \mathcal{C}$ be a functor, and let \mathcal{A} be a category with (specified) limits. Then the functor*

$$\mathcal{A}^K : [\mathcal{C}, \mathcal{A}] \rightarrow [\mathcal{M}, \mathcal{A}] , \quad F \mapsto F \circ K$$

is a left adjoint (a.k.a., global right Kan extensions along K exist).

Theorem 3 [10, Sect. V.5]. *If $F : \mathcal{C} \rightarrow \mathcal{A}$ is a left adjoint, then it preserves any colimit.*

Combining those two results, we obtain that the functor F_K preserves colimits, in particular, it is ω -cocontinuous. This result hence allows the construction of heterogeneous datatypes in **UniMath**.

We note that a more direct proof of the fact that F_K preserves colimits is also possible. However, the construction of right Kan extensions is important also in the context of the related project [4].

Every summand of the functor A above corresponds to one language constructor. Note that a generalization to infinitely many summands is straightforward, so that languages with infinitely many language constructors can also be modeled by our inductive families of sets.

2.4 Connection to Work on Heterogeneous Substitution Systems

Matthes and Uustalu [11] introduce the notion of (*heterogeneous*) *substitution system* in order to give a categorical axiomatisation of *substitution* for languages with variable binding, such as the lambda calculus. Ahrens and Matthes [4]

extend the axiomatisation by devising a notion of morphism of substitution systems, and formalize results of [11] as well as some new results in **UniMath**. One main result formalized in [4] equips a given initial algebra—whose existence is assumed—of an endofunctor on functor categories with a substitution structure [4, Theorem 28], and shows that the resulting substitution system is initial in a suitable category [4, Theorem 29].

We instantiate Theorem 28 of [4] with inductive families of sets, e.g., the lambda calculus specified by the functor Λ , and thus obtain a monad structure on that inductive family for free. Afterwards, we implement a convenient way of specifying languages via a syntactic notion of signature, where, e.g., the lambda calculus is specified by the signature $\{[0,0],[1]\}$. Here, the numbers indicate the number of variables bound in each argument of the corresponding language constructor. We hence obtain a framework which allows to concisely specify a programming language with binders, and which, from such a specification, automatically produces a certified formalization of that language.

While the property of being an initial algebra is for free and automatic for the languages obtained via the machinery we have formalized, we do not know at the moment of an automated way to generate, from initiality, the type-theoretic constructors and suitable recursion schemes. Doing this automatically probably requires some engineering on the meta-level of the proof assistant **Coq**, e.g., writing a plugin to that program.

Details about the project sketched in this section will be explained in a forthcoming article.

3 Some Details on the Formalization

In practice, the **UniMath** language is given by a subset of the language implemented in the proof assistant **Coq**. It is up to the formalizers to make sure that they respect this restriction—there is no mechanic check at the moment. Formalizing in **UniMath** hence consists in writing **Coq** files, using only the constructions mentioned in the introduction as being part of the **UniMath** language. At the time of writing, the **UniMath** library of formalized mathematics consists of about 46k lines of code. The library of formalized mathematics written for this project consists of approximately 4500 lines of code. The code has been integrated into an already existing library of category theory in **UniMath**.

In general, there are two ways of implementing a language construct:

Firstly, it can be *internal* to the language, as in the present work. Then, the construct is merely an abbreviation for its expansion into the primitive notions.

Secondly, the implementation can be *external*, i.e., on the meta-level. This option allows to refer to concepts that cannot be referred to within the language, such as *convertibility* (a.k.a. *judgmental equality*). For instance, the native inductive types of the **Coq** proof assistant are implemented on the meta-level.

Our approach has two disadvantages compared to an implementation on the meta-level of inductive sets:

- the inductive sets we construct will generally not satisfy computation rules *judgmentally*, but only *propositionally*
- the inductive sets we construct will generally not have a normal form.

The lack of judgmental computation rules is reminiscent of the *deliberate* blocking of computation in the `SSReflect` [9] extension of the `Coq` proof assistant.

On the other hand, the advantage of our approach to inductive sets is clear: since it does not extend the language, its consistency is immediate.

4 Conclusions

In this work, we report on a construction of some inductive sets in `UniMath`. The inductive sets we construct suffer from some defects, but have the advantage of not requiring any language extension. It remains to be seen if the defects get in the way when doing proofs involving inductive sets—the fact that `SSReflect` deliberately imposes a similar restriction indicates that the practical implications of the defects will be minor.

Acknowledgments. We thank Dan Grayson, Ralph Matthes, Paige North and Vladimir Voevodsky for helpful discussions on the subject matter.

This material is based upon work supported by the National Science Foundation under agreement Nos. DMS-1128155 and CMU 1150129-338510. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

1. Adámek, J.: Free algebras and automata realizations in the language of categories. *Comment. Math. Univ. Carol.* **15**(4), 589–602 (1974)
2. Ahrens, B., Capriotti, P., Spadotti, R.: Non-wellfounded trees in homotopy type theory. In: Altenkirch, T. (ed.) 13th International Conference on Typed Lambda Calculi and Applications, TLCA, Warsaw, Poland, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 1–3 July 2015. *LIPIcs*, vol. 38, pp. 17–30 (2015)
3. Ahrens, B., Kapulkin, K., Shulman, M.: Univalent categories and the Rezk completion. *Math. Struct. Comput. Sci.* **25**, 1010–1039 (2015)
4. Ahrens, B., Matthes, R.: Heterogeneous substitution systems revisited. *CORR*, abs/1601.04299 (2016)
5. Awodey, S., Gambino, N., Sojakova, K.: Inductive types in homotopy type theory. In: *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, 25–28 June 2012*, pp. 95–104. IEEE Computer Society (2012)
6. Bezem, M., Coquand, T., Huber, S.: A model of type theory in cubical sets. In: Matthes, R., Schubert, A. (eds.) 19th Conference on Types for Proofs and Programs, TYPES 2013, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. *LIPIcs*, vol. 26, pp. 107–128 (2013)
7. Bird, R.S., Paterson, R.: De Bruijn notation as a nested datatype. *J. Funct. Prog.* **9**(1), 77–91 (1999)

8. Cohen, C., Coquand, T., Huber, S., Mörtberg, A.: Cubical Type Theory: a constructive interpretation of the univalence axiom (2015, Preprint)
9. Gonthier, G., Mahboubi, A.: An introduction to small scale reflection in Coq. *J. Formaliz. Reason.* **3**(2), 95–152 (2010)
10. Mac Lane, S.: *Categories for the Working Mathematician*. Graduate Texts in Mathematics, vol. 5, 2nd edn. Springer, New York (1998)
11. Matthes, R., Uustalu, T.: Substitution in non-wellfounded syntax with variable binding. *Theor. Comput. Sci.* **327**(1–2), 155–174 (2004)
12. Voevodsky, V.: An experimental library of formalized mathematics based on the univalent foundations. *Math. Struct. Comput. Sci.* **25**, 1278–1294 (2015). <http://arxiv.org/pdf/1401.0053.pdf>

Mathematical Software – ICMS 2016

5th International Conference, Berlin, Germany, July

11–14, 2016, Proceedings

Greuel, G.-M.; Koch, T.; Paule, P.; Sommese, A. (Eds.)

2016, XXIV, 532 p. 111 illus., Softcover

ISBN: 978-3-319-42431-6