<Isomorphism is equality>

<A Coq formalisation of the proofs Isomorphism is equality by Coquand T. and Danielsson NA. >

<Tiago Greeve[1]>

Supervisor(s): <Benedikt Ahrens[1]>, <Kobe Wullaert[1]>

[1]EEMCS, Delft University of Technology, The Netherlands

Name of the student: <Tiago Greeve>
Final project course: CSE3000 Research Project
Thesis committee: <Benedikt Ahrens>, <Kobe Wullaert>, <Neil Yorke-Smith>

Abstract

This paper will give a formalisation of proofs, given in the paper "isomorphism is equality"((Coquand & Danielsson, 2013), in the proof assistant language Coq. The formalisations will be added to UniMath library. A library containing machine readable proofs in the mathematical field of Homotopy Type theory, a relatively new field which combines Homotopy Theory and Martin-Löf Type Theory. The proofs that have been formalised are the equality pair lemma and the proof that isomorphism is equivalent to equality. We have also constructed a concrete universe on which we defined a notion of isomorphism, as per the same paper.

1 Introduction

Homotopy Type Theory (followingly denoted as HoTT) is a new field of mathematics, which is being considered as a mathematical foundation instead of set theory. HoTT is based on type theory as developed by Martin-Löf with an interpretation based on homotopy theory, where types are spaces and elements of types are points in a space.

HoTT is a particularly interesting field of mathematics, because the type-theoretic basis allows for machine readable proofs, in contrast to other fields of mathematics where proofs are written in plain English and need to be verified by other humans for correctness. The development of HoTT mathematics is thus particularly interesting, because machine readable proofs allow for definitive correctness, machine readable proofs also opens up possibilities for verifying functionality of computer programs and allows proofs to be easily used as lemmas or theorems in other proofs, like one would use a function in a regular computer program.

The foundations for HoTT were laid by Vladimir Voevodsky in his work univalent foundations, which were accompanied by a library of formalisations in the proof assistant coq, namely UNIVALENT FOUNDATIONS. Later, the library named UniMath was set up, with Voevodsky as one of members of the committee overseeing the library. The UniMath library consists of mathematical proofs in HoTT and is based on the univalent foundations of Vladimir Voevodsky, which distinguishes itself by the univalent axiom, which states that there is a function from equality between types to equivalence between types and that this function is itself an equivalence.

For this research project We have selected a paper on the HoTT topic of "Isomorphism is equality", followingly called "the isomorphism paper"(Coquand & Danielsson, 2013), and we will be aiming to formalize the results of this paper in the UniMath library, such that is available for anyone to directly use the lemma's and theorems as part of their own proofs. My research will thus answer the following question: Can we formalise the results of the paper "isomorphism is equality" into code to extend the UniMath library?

1.1 The basis of HoTT

The main concepts of HoTT include, dependent function types, dependent pair types and identity types.

Dependent function types or $\prod -types$, are types which elements are functions, where the type of the applied function depends on the input. Alternatively, this can be interpreted as "for all x of type A we get an element of B(x). We denote them as:

$$\prod x : A, B(x)$$

or as

$$\prod (x : A), B(x)$$

.

Dependent pair types or $\sum -types$, are types which elements are pairs, where the type of the second element of the pair is dependent on the first element. Pairs can be deconstructed into their corresponding elements, by taking a projection of the pair. For $s : \sum (x : A), B(x)$, the first projection, denoted (pr1 s), is an element of type A, and the second projection, denoted (pr2 s), is an element of type B(x). Alternatively one can interpret this as "there exists an x of type A such that we get an element of type B(x). We denote them as:

$$\sum x : A,\ B(x)$$

or as

$$\sum (x : A),\ B(x)$$

An element of such a type can be written in the form: $(x,,\ y$ where the x is of type a and y is of type B(x).

Identity types are how we denote equality in HoTT, for two elements x and y of a type A, we can construct the identity type, denoted as $x = y$. This type can be interpreted as a propositional equality between x and y. Elements of type $x = y$, are witnesses to the equality of type $x = y$. In the homotopy theory interpretation, elements of identity types are so called paths between two points. There is one constructor for identity types, namely idpath. For an element x, this constructor constructs an element of type $x = x$. This is possible because there is always a path from a point to itself. (The Univalent Foundations Program, 2013)

A concise explanation of univalence and the univalence axiom is given by Escardó(Escardó, 2018). For us merely the implications of the univalence axiom are important, which we mention in the section explaining preliminary definitions.

1.2 What is isomorphism?

The paper "Univalent foundations and the equivalence principle" describes different notions of sameness: Equality, isomorphism, equivalence and how they relate to each. Each of them state that two entities are the same when both of them satisfy the same specific set of properties. What these properties are, depends on the domain in which these two entities reside, the difference between these notions of sameness is thus about the domain in which you want to specify sameness, that is what determines the properties that need to hold. In the case of isomorphisms, the notion of sameness we are especially interested in for this paper, they are usually used when we are talking about objects that resort in a category.

In the case we are considering in this paper we do not only want properties of these objects to be invariant under isomorphism, but we also want structures on these objects, as Ahrens and North call it, to be invariant under isomorphism. In our paper we will see that this corresponds to the following: Two instances of codes are isomorphic if the type of a-structures on the carrier type can be transported to another carrier type, if there is an equivalence between the carrier types (Ahrens & North, 2019).

2 Goal of research and preliminary definitions

The goal of this research is to prove that isomorphism is equivalent to equality, in order to do this, we first have to define what isomorphism is. We would also like to construct a concrete universe for which we give a definition of isomorphism. For both these goals we first need to introduce preliminary definitions and theorems in the field of HoTT. We use notation that is similar to Coq and adhere to the style as used in the UniMath library.

2.1 The transport function

We have previously introduced the identity type for propositional equality. The type of equality is substitive, meaning if we have elements x, y : X and we have a property $\prod (x : X), P(x)$, if we have an equality x = y, then we also have P(x) = P(y). Thus if a property holds for x, it must also hold for y. Another way to say this is if there is a path from x to y, then we can transport P(x) along this path to P(y):
$transportf : \{X : UU\} \to (P : X \to UU) \to \{x\ x' : X\} \to (e : x = x') \to P\ x \to P\ x'$

2.2 Contractibility

A type T is contractible if all it's elements can be contracted to a single point:
$iscontr\ (T : UU) : \sum (cntr : T),\ \prod (t : T),\ t = cntr$

2.3 isaprop

A type X is a proposition, if the equality between two elements of that type, is contractible:
$isaprop\ (X : UU) : \prod (x\ x' : X),\ iscontr\ (x = x')$

2.4 Equality of sigma types

For two elements x y of (sigma type) $\sum z : A,\ B(z)$, the equality x = y is equivalent to the pair of equalities, of the first and second projections of x and y, where for the equality between the second projections, we transport the left-hand-side of the equality along the equality (path) between the first projections:
$sigma\_equality : x = y \simeq \sum p : pr1\ x = pr1\ y,\ transportf\ A\ p\ (pr2\ x)\ = pr2\ y$

For two elements x y of (sigma type) $\sum z : A,\ B(z)$, where second components are propositionally typed, the equality between x and y is merely the equality between their first components: $remove\_prop : x = y \simeq pr1\ x = pr1\ y$

2.5 Univalence

As equality is a stricter definition of sameness than equivalence, it is easy to see that a function from equality to equivalence needs to exist. We will call it eqweqmap:
$eqweqmap\ \{T1\ T2 : UU\} : T1 = T2 \to T1 \simeq T2$

What univalence states is that this function is, in fact an equivalence. Thus we also we get the inverse of eqweqmap, we will call it weqtopaths:
$weqtopaths\ \{T1\ T2 : UU\} : T1 \simeq T2 \to T1 = T2$

2.6 The transport theorem

Say we have a predicate $P : UU \to UU$ and a function R that says that for any two types X and X', P respects equivalence between X and X' and a function r that says that R maps the identity equivance of X to the identity function, then for all X X', if we have an equivalence between X and X', we can use univalence to get to the following:
$\prod (p : PX),\ R\ X\ X'\ w\ p = transportf\ P\ (weqtopaths\ w)\ p$

2.7 Associativity of $\sum type$

Nested $\sum types$ have an associative property. If we have a nested pair (x,, y,, z), by associativity we can write this as ((x,, y),, z). In other words, for a type X and a type family P on X, and a type family Q on $(\sum\ y,\ P\ y)$ we get:
$(\sum\ y,\ Q\ y) \simeq (\sum (x : X)\ (p : P\ x),\ Q\ (x,,\ p))$

3 Coq formalisation of isomorphism is equality

The following coq formalisations of the proofs "equality-pair-lemma" and "isomorphism-is-equality" in "isomorphism is equality" (Coquand & Danielsson, 2013), are written in Coq version 8.15.2 (released Jun 2022) with the first package pick from Apr 2022, also know as 8.15 2022.04, and added to the UniMath library. The configuration of coq in the UniMath library can be found in the repo (Voevodsky et al., n.d.). One notable setting is the disabling of a type hierarchy, which adds inconsistency (one can prove false). This was necessary in order implement Voevodsky's resizing rules. Further research is being done in order to solve this problem . The isomorphism paper does utilize a type hierarchy for the agda formalisations, thus we deviate from the isomorphism paper in that aspect, we use UU everywhere to denote a universe.

As we specified previously we will define a notion of isomorphism between structures on objects that reside in categories, going forwards these objects will be called codes, and the structures on these objects will be called a-structures on codes. We will define what it means to be a code and what it means for two codes to be isomorphic, then we will prove that this notion of isomorphism is equivalent to the notion of equality. Lastly, we will define a concrete universe, to give meaning to the vague term 'a-structures', and define an alternative, more intuitive, notion of isomorphism for this concrete universe, and we prove that this new definition is equivalent to our original definition of isomorphism.

3.1 Parametrizations

Our work is parametrized by the following:

- A type of codes U.
- A decoding function $El : U \to UU \to UU$

- The requirement that El a, when seen as a property for a type B : UU, respects equivalences, which we express by the following function:
$resp : \prod(a : U),\ \prod(B\ C : UU),\ B \simeq C \to El\ a\ B \to El\ a\ C$
- The resp function should map the identity equivalence to the identity function:
$resp\_id : \prod a\ B,\ \prod(x : El\ a\ B),\ resp\ a\ B\ B\ (idweq\ B)\ x = x$

3.2 Constructions

Codes consist of two parts, a code, that is an element of the universe, and a family of propositions:

$Code := \sum(a : U),\ \prod(C : UU),\ El\ a\ C \to \sum P,\ isaprop\ P$

Instances of codes consist of a carrier type, an element, and a number of propositional laws that define what it means to be an instance of a code. For example a mathematical group has a propositional law that state that every element has an inverse:

$instance\ (code : Code) := \sum(C : UU),\ \sum(x : (El\ (pr1\ code)\ C)),\ pr1\ ((pr2\ code)\ C\ x)$

In order to prove isomorphism is equivalent to equality we need to define isomorphism. We define a predicate which states when a given equivalence is an isomorphism from one element to another:

$isIsomorphism\ (a : U)\ \{B\ C\}\ (eq : B \simeq C)\ (x : El\ a\ B)\ (y : El\ a\ C) := (resp\ a\ B\ C\ eq\ x) = y$

Isomorphism between two instance of code x and y can be defined by stating that there is in fact such an equivalence:

$Isomorphic := \sum eq : (pr1\ x) \simeq (pr1\ y),$
isIsomorphism (pr1 code) eq (pr1 (pr2 x)) (pr1 (pr2 y))

The carrier is the type on which we define a family of types: $Carrier\ (code : Code)\ (inst : instance\ code) := (pr1\ inst)$

The element is then a type of a-structure on the carrier type: $element\ (code : Code)\ (inst : instance\ code) := pr1\ (pr2\ inst)$

3.3 Equality pair lemma

In order to prove that isomorphism is equality, we need a lemma which states that the equality of two instances of code is the same as the pair of equalities between their carrier types and their elements, with the first element transported along the equality from the carrier type of X, to the the carrier type of Y. We state this "equality-pair-lemm as follows:
$\prod(c : Code)\ (X\ Y : instance\ c),\ X = Y \simeq (\sum eq : Carrier\ c\ X = Carrier\ c\ Y,\ transportf\ (El\ (pr1\ c))\ eq\ (element\ c\ X) = element\ c\ Y)$

Because Codes and instances of codes are elements of sigma types, we can split them up in pairs, namely: c = (a,, p), X = (C, x, p) and Y = (D, y, q). Where a is an element of the universe U and P is the family of propositions, and where C is the carrier type of X, x is the element of X and p is the propositional laws for X, Y follows a similar pattern. We unfold these definitions as well as the definitions of Carrier and element, we get: $(C,,x,,p) = (D,,y,,q) \simeq \sum eq : C = D,\ transportf\ (El\ a)\ eq\ x = y$

We apply the associative quality of sigma types:
$(C,,\ x),,\ p = (D,,\ y),,\ q \simeq \sum eq : C = D,\ transportf(El\ a)\ eq\ x = y$

Now we know that p and q are propositionally typed. This means we can drop them:
$(C,,\ x) = (D,,\ y) \simeq \sum eq : C = D,\ transportf(El\ a)\ eq\ x = y$

As we can see we are left with the definition of equality of sigma types. The statement is thus proven.

3.4 Isomorphism is equality

Now that we have proven the equality-pair-lemma we move on to the main theorem. Again we assume c = (a,, P), X = (C,, x,, p) and Y = (D,, y,, q).

$\prod(c : Code)\ (X\ Y : instance\ c),\ Isomorphic\ c\ X\ Y \simeq X = Y$

First we unfold the definitions of Code and instance and isomorphic, giving us the following equivalence:

$\sum eq : C \simeq D,\ resp\ a\ C\ D\ eq\ x = y \simeq (C,,x,,p) = (D,,y,,q)$

We apply the transport theorem, instantiated with resp and resp-id as R and r:
$\sum eq : C \simeq D,\ transportf\ (El\ a)\ (weqtopaths\ eq)\ x = y \simeq (C,,x,,p) = (D,,y,,q)$

Now we apply univalence:
$\sum eq : C = D,\ transportf\ (El\ a)\ eq\ x = y \simeq (C,,x,,p) = (D,,y,,q)$

Lastly we apply our previously defined equality pair lemma and see that we indeed have an equivalence between isomorphism and equality.

3.5 Concrete universes

In order to define a concrete universe, we aim to give a concrete type U, decoding function El, as well as concrete definitions for the functions resp and resp_id.
The concrete type U is defined as follows:

```
Inductive U : UU :=
  | id : U
  | type : U
  | k : UU -> U
  | arrow : U -> U -> U
  | cartesian : U -> U -> U
  | binary : U -> U -> U.
```

The decoding function is defined as follows:

```
Fixpoint El (u : U) (C : UU) : UU :=
  match u with
  | id => C
  | type => Type
  | k A => A
  | (arrow a b) => (El a C) -> (El b C)
  | (cartesian a b) => (El a C) × (El b C)
  | (binary a b) => (El a C)  (El b C)
  end.
```

U is defined as an inductive type with the six constructors id, type, k etc. and the decoding function El a is defined by recursion on the strucutre of a.
In order to define resp, we first define a cast operator which shows that El preserves equivalence.
We define the cast operator by recursion on the structure of a (where func = arrow, cart = cartesian and bin = binary):

```
Fixpoint cast (a : U) {B C : UU} (eq : B  C)
    : (El a B)  (El a C)
 :=
 match a with
 | id => (eq : (El id B)  (El id C))
 | type => (idweq Type)
 | (k A) => (idweq A)
 | (func l r) => func_eq (cast l eq) (cast r eq)
 | (cart l r) => cart_eq (cast l eq) (cast r eq)
 | (bin l r) => bin_eq (cast l eq) (cast r eq)
 end.
```

The logical combinators, arrow_eq, cartesian_eq and binary_eq, have the following type signatures (we define them in our coq formalisation):
$arrow\_eq : \{A\ B\ C\ D : UU\}\ (x : A \simeq B)\ (y : C \simeq D) : (A \to C) \simeq (B \to D)$

$cartesian\_eq : \{A\ B\ C\ D : UU\}\ (x : A \simeq B)\ (y : C \simeq D) : (A \times C) \simeq (B \times D)$

$binary\_eq : \{A\ B\ C\ D : UU\}\ (x : A \simeq B)\ (y : C \simeq D) : (A \coprod C) \simeq (B \coprod D)$

Using cast we can define resp. We have not been able to prove that cast maps identity equivalence to identity equivalence. Assuming we have proven this, we can define resp_id. Thus we have given all parameters.

Next, now that we have a concrete universe, we define an alternative notion on the struture of a. We do this as we feel it is a more intuitive definition (where, a = arrow, c = cartesian, b = binary, isIso = isIsomorphism:

```
Fixpoint isIsomorphism' (a : U) {B C : UU}
    (eq : B  C) :  (x : El a B) (y : El a C), UU
  :=
  match a with
  | id =>  x y, ((pr1weq eq) (x : El id B))
                   = (y : El id C)
  | type =>  X Y, X  Y
  | (k A) =>  x y, (x = y)
  | (a l r) => a_rel (isIso' l eq) (isIso' r eq)
  | (c l r) => c_rel (isIso' l eq) (isIso' r eq)
  | (b l r) => b_rel (isIso' l eq) (isIso' r eq)
  end.
```

With the relational combinators above defined as follows:

$arrow\_rel\ P\ Q\ l\ r := \prod x\ y,\ P\ x\ y \to Q\ (f\ x)\ (g\ y)$

$cartesian\_rel\ P\ Q\ l\ r := (P\ (pr1\ l)\ (pr1\ r)) \times (Q\ (pr2\ l)\ (pr2\ r))$

$binary\_rel\ P\ Q\ (ii1\ x)\ (ii1\ y) := P\ x\ y$
$binary\_rel\ P\ Q\ (ii1\ x)\ (ii2\ y) := \emptyset$
$binary\_rel\ P\ Q\ (ii2\ x)\ (ii1\ y) := \emptyset$
$binary\_rel\ P\ Q\ (ii2\ x)\ (ii2\ y) := Q\ x\ y$

What is left is to prove that this notion of isomorphism is equivalent to our original notion. As when we do, we can define what it means for concrete instances of codes, like monoids and posets, to be isomorphic, using our definitions.

4 Challenges and responsible research

During the course of this research there were some challenges we encountered, on which we will reflect in this section, we will also reflect upon how we conducted our research and if we were responsible in our research.

4.1 Challenges during formalisation in UniMath

Some UniMath theorems that we used in the proofs, needed to be adapted to better suit the proofs, as the agda formalisations (Coquand & Danielsson, 2013) sometimes used different conventions or used similar theorems to the ones in UniMath but with a different type signature.

In particular the proof weqtransportf in our formalisation, is an adaption of the proof weqtransportbUAH from UniMath/Foundations/UnivalenceAxiom.v, where the function R (which corresponds to resp) is slightly different and the theorem uses transporb instead of transportf, thus we rewrote the theorem to use transportf, also using the univalence axiomas instead of the hypotheses because they were not usuable in the current context (outside of the UnivalenceAxiom.v file), the implications of this are not clear but my assumption is that if these are significant, there is a way to prove weqtransportf using weqtransportbUAH and possibly other already proven theorems.

Another issue that arose was that the steps used in the proofs at first seemed not to translate to theorems that were present in UniMath, requiring me to find other theorems that did seem to allow me to take the steps neccesary to prove certain statements, however upon later reflection on my formalisations, combining certain theorems already present in UniMath did allow me to follow the same order of steps as present in the isomorphism paper.

4.2 Responsible Research

As this research can be considered the study of pure mathematics, no ethical implications of this research have been considered. Instead laying responsibility for any ethical implications of those who apply mathematics. While realizing there are other views(Ernest, 2021), we find this justified as the possible applications of pure mathematics are usually not in any way obvious, thus not allowing for rigorous evaluation of ethical implications.

The reproducibility of this research project mainly depends on the understanding of the mathematics, but also, to a lesser degree, on the setup of the code, we think this research paper provides enough information on both, to be considered reproducible. The intuition behind most mathematical reasoning is given, as well as extended resources, in the form of references, to read more about the topic of HoTT such that it is possible to understand. The versions for both the Coq language and the unimath library, on which the formalisations depend, have been described in the third chapter of this paper.

5 Conclusions and Future Work

In this paper, we set out to prove that the notion of isomorphism, as characterized in the isomorphism paper and again in this paper, is equivalent to the notion of equality. Furthermore we set out to provide a concrete universe and provide a new, more intuitive notion of isomorphism between elements created from this universe, and prove that this notion of isomorphism is equivalent to our original notion.

In order to prove the equivalence of isomorphism and equality, we first created a definition for codes and a notion of isomorphism between instances of these codes, then we proved the equality pair lemma lemma, allowing us to drop the propositional laws of the instances of code, and used this lemma to prove isomorphism, in fact, equivalent to equality. As such, the main research question "Can we formalise the results of the paper "isomorphism is equality" into code to extend the UniMath library?" has been answered succesfully.

We have defined a concrete universe U, an inductive data type, and a decoding function El for the concrete universe. We have also defined a cast function, to show that (El a) preserves equivalence between carrier types. We then use the cast function to define resp and using the fact that cast maps the identity to the identity, we define resp_id. We then define a notion if isomorphism for this concrete universe.

5.1 Future work

The proof that this notion of isomorphism is equivalent to our original notion, is a subject for future work, as we were not able to fully prove this. More precisely, we were not able to prove the cases for the function space and binary sum. Additionally, we have assumed the fact that cast maps the identity to the identity. This fact can however be proven by defining a function cast_id. Which we have used to define our resp_id function.

To prove the usefulness of our construction, one could define examples of codes (e.g. monoids, posets etc.) using our concrete universe, and then show what isomorphism between instances of these codes would mean using our definitions of isomorphism.

References

Ahrens, B., & North, P. R. (2019). *Univalent foundations and the equivalence principle*. https://doi.org/10.48550/arXiv.2202.01892

Coquand, T., & Danielsson, N. A. (2013). Isomorphism is equality [In memory of N.G. (Dick) de Bruijn (1918–2012)]. *Indagationes Mathematicae*, *24*(4), 1105–1120. https://doi.org/https://doi.org/10.1016/j.indag.2013.09.002

Ernest, P. (2021). Mathematics, ethics and purism: An application of macintyre's virtue theory. *Synthese*, *199*, 3137–3167. https://doi.org/10.1007/s11229-020-02928-1

Escardó, M. H. (2018). A self-contained, brief and complete formulation of voevodsky's univalence axiom. https://doi.org/10.48550/arXiv.1803.02294

The Univalent Foundations Program. (2013). *Homotopy type theory: Univalent foundations of mathematics* (tech. rep.). Institute for Advanced Study.

Voevodsky, V., Ahrens, B., Grayson, D., et al. (n.d.). Unimath — a computer-checked library of univalent mathematics. available at http://unimath.org. https://doi.org/10.5281/zenodo.7848572