Under consideration for publication in J. Functional Programming

THEORETICAL PEARL

$\doteq \simeq \equiv$

Leibniz Equality is Isomorphic to Martin-Löf Identity, Parametrically

ANDREAS ABEL
Gothenburg University
andreas.abel@gu.se

JESPER COCKX
Gothenburg University
cockx@chalmers.se

DOMINIQUE DEVRIESE
KU Leuven, Belgium
dominique.devriese@cs.kuleuven.be

AMIN TIMANY
KU Leuven, Belgium
amin.timany@cs.kuleuven.be

PHILIP WADLER
University of Edinburgh
wadler@inf.ed.ac.uk

Abstract

Consider two widely used definitions of equality. That of Leibniz: one value equals another if any predicate that holds of the first holds of the second. And that of Martin-Löf: the type identifying one value with another is occupied if the two values are identical. The former dates back several centuries, while the latter is widely used in proof systems such as Agda and Coq.

Here we show that the two definitions are isomorphic: we can convert any proof of Leibniz equality to one of Martin-Löf identity, and each conversion followed by the other is the identity. One direction of the isomorphism depends crucially on values of the type corresponding to Leibniz equality satisfying Reynolds' notion of parametricity and functional extensionality. The existence of the conversions is widely known (meaning that if one can prove one equality then one can prove the other), but that the two conversions form an isomorphism (internally) in the presence of parametricity and functional extensionality is, we believe, new.

Our result is a special case of a more general relation that holds between inductive families and their Church encodings. Our proofs are given inside type theory, rather than meta-theoretically. Our paper is a literate Agda script.

1 Introduction

If anything is sufficiently important, there will be more than one way to do it. For instance, the concept of two objects being the same can be expressed in at least two ways, Leibniz equality and Martin-Löf identity.

Leibniz asserted the *identity of indiscernibles*: two objects are equal if and only if they satisfy the same properties (Leibniz, 1686). This principle sometimes goes by the name Leibniz' Law, and is closely related to Spock's Law, "A difference that makes no difference is no difference". Leibniz equality is usually formalized to state that $a \doteq b$ holds if every property P that holds of a also holds of b. Perhaps surprisingly, this definition is sufficient to also ensure the converse, that every property P that holds of b also holds of a.

The setting for our paper is dependent type theory. We present our development as a literate script for the proof assistant Agda, although our results should be easy to carry over to other settings. In fact, we also provide Coq versions of our results. Such systems typically define equality in the style introduced by Martin-Löf (Martin-Löf, 1975). We will express Martin-Löf identity by stating that the type $a \equiv b$ is inhabited when a is the same as b. We say that refl inhabits the type $a \equiv a$, while the type $a \equiv b$ is empty when a and b are distinct.

Martin-Löf identity has been the subject of considerable discussion (Streicher, 1993; Hofmann & Streicher, 1996; Altenkirch *et al.*, 2007; the Univalent Foundations Program, 2013; Cohen *et al.*, 2016). In contrast, Leibniz equality has seen surprisingly little discussion in our community.

It is not hard to show that both concepts are equivalence relations: each is reflexive, symmetric, and transitive. Further, it is well known that Leibniz equality ($a \doteq b$) implies Martin-Löf identity ($a \equiv b$), and vice versa. Indeed, the former is sometimes taken as the name of the latter (Pfenning & Paulin-Mohring, 1989; Coq Developers, 2017). Here, to avoid confusion, we stick to calling each by its own name as defined above.

Simply stating that Leibniz equality implies Martin-Löf identity, and vice versa, is a weak connection: it means that there is a function that takes every proof of the former into a proof of the latter, and a function that takes every proof of the latter into a proof of the former. Here we prove something stronger, namely that Leibniz equality is *isomorphic* to Martin-Löf identity: the two functions of the previous sentence are inverses, giving a one-to-one correspondence. As such, our results imply more than just the existence of equality proofs, but also relate their computational content. Isomorphism plays a distinguished role in category theory and homotopy type theory, where it is referred to as an *equivalence*, and the computational content of equality proofs is increasingly important in recent type theories such as Cubical (Cohen *et al.*, 2016) and HoTT (the Univalent Foundations Program, 2013). So far as we are aware, our claim that Leibniz equality is isomorphic to Martin-Löf identity is new.

Our proof of the isomorphism depends on the assumption that values of the type corresponding to Leibniz equality satisfy a suitable notion of *parametricity*. Parametricity was originally defined for the polymorphic lambda calculus (also known as System F) by Reynolds (Reynolds, 1983). Applications of parametricity to functional programming are given by Wadler (Wadler, 1989). While most descriptions of parametricity are in terms of semantics, a simple syntactic description of parametricity as a mapping from System F to

second-order logic appears in Wadler (Wadler, 2007). Parametricity has been adapted to dependent type theory (Bernardy *et al.*, 2012; Atkey *et al.*, 2014; Bernardy *et al.*, 2015; Nuyts *et al.*, 2017).

Our proof of the isomorphism also depends on the assumption that the Martin-Löf identity type satisfies functional extensionality: two functions are equal if and only if they map equal inputs to equal results. A posteriori, our result implies Leibniz equality then also satisfies functional extensionality.

Leibniz equality can be viewed as the Church encoding of Martin-Löf identity. Pfenning and Paulin-Mohring conjectured that inductively defined data types are isomorphic to their Church encodings in the presence of parametricity (Pfenning & Paulin-Mohring, 1989). Several special cases of this are known in the literature. Hasegawa demonstrates that categorical product and sum are isomorphic to their Church encodings in the presence of parametricity (Hasegawa, 1994), and Wadler demonstrates that the inductive definition of natural numbers is isomorphic to the Church numerals assuming that Church numerals satisfy parametricity (Wadler, 2007). Atkey, Ghani and Johann proved a general version of the conjecture using their formulation of parametricity in dependent type theory (Atkey *et al.*, 2014). This paper can be viewed as zooming in on a special case of the property, but note that we prove the equivalence inside type theory, rather than meta-theoretically.

Leibniz equality is normally formulated in an *impredicative* setting: when we say that $a \doteq b$ holds if every property P that holds of a also holds of b, the property P may itself refer to Leibniz equality. Here, perhaps surprisingly, we formulate our results in the *predicative* setting of Agda, where the propery P may not refer to Leibniz equality. However, there is no difficulty in extending our results to an impredicate setting, and we have done so in Coq.

Our results depend only on parametricity and functional extensionality. Practically, this means that our results are valid in a traditional setting where the principle of Uniqueness of Identity Proofs (UIP) holds (all proofs of the same equality are equal), or equivalently, Streicher's Axiom K (Streicher, 1993). However, they also apply in the setting of homotopy type theory (the Univalent Foundations Program, 2013), where other principles like Univalence (roughly: any two isomorphic types are equal) can be used, but UIP and Axiom K are inadmissible.

This note is a literate Agda script. Its source is available as an artifact, and can be directly executed in Agda. We also submit as an artifact the same proof in Coq. Our Agda proof uses a predicative setting, and the Coq proof uses both predicative and impredicative settings.

This paper is organised as follows. Section 2 introduces Martin-Löf identity, and Section 3 introduces Leibniz equality. Section 4 reprises a related result from the literature, where two types are shown to be isomorphic in the presence of parametricity. Section 5 presents our proof that Leibniz equality is isomorphic to Martin-Löf identity. Section 6 applies our main result to show that all universe-polymorphic variants of Leibniz equality are isomorphic. Section 7 concludes.

Our script begins with a little housekeeping. We use an Agda pragma to avoid pattern matches that depend on Axiom K (Cockx *et al.*, 2014; Cockx *et al.*, 2016).

```
{-# OPTIONS --without-K #-}
```

This ensures that we do not use Axiom K by accident, and thus, our development can be carried out the same in homotopy type theory.

We make use of modules. As required by Agda, our entire development is encapsulated in a module with the same name as the file containing this paper.

```
module LeibnizEquality where
```

We only import one module from the Agda standard library, which defines universe levels.

```
open import Agda.Primitive public
```

2 Martin-Löf identity

As mentioned in the introduction, we follow Martin-Löf (Martin-Löf, 1975) by taking the *identity type* $a \equiv b$ to be the type expressing the equality of two objects a and b. This type has a single constructor, $\mathsf{refl} : a \equiv a$. If an element $a{\equiv}b : a \equiv b$ can be constructed, then a and b are considered equal. On the other hand, if a and b are obviously distinct, for example $a = 0$ and $b = 1$, then $a \equiv b$ is an empty type. (Here we have followed the usual Agda convention, where we let the identifier $a{\equiv}b$ range over values of the type $a \equiv b$. In Agda almost any sequence of characters not containing a space may be used as an identifier.)

We encapsulate this section as an Agda module, assuming a set A for all definitions at once.

```
module Martin-Löf {ℓ} {A : Set ℓ} where
```

We define equality at types A in any universe $\mathsf{Set}\ \ell$, as usual in Agda. We will return to universe polymorphism in Section 6.

We define $\equiv$ inductively as an indexed data type with the single constructor refl:

```
  data _≡_ (a : A) : A → Set ℓ where
    refl : a ≡ a
```

This characterises Martin-Löf identity as the least type containing $\mathsf{refl} : a \equiv a$ for any $a : A$.

Martin-Löf identity satisfies the standard properties of an equivalence relation: reflexivity, symmetry, and transitivity. In Agda, we can show these properties using *dependent pattern matching* (Coquand, 1992):

```
  reflexive≡ : {a : A} → a ≡ a
  reflexive≡ = refl

  symmetric≡ : {a b : A} → a ≡ b → b ≡ a
  symmetric≡ refl = refl

  transitive≡ : {a b c : A} → a ≡ b → b ≡ c → a ≡ c
  transitive≡ refl refl = refl
```

We end the module, and make all its definitions available. This allows us to invoke $\equiv$ at any type, not just A.

```
open Martin-Löf public
```

We require one additional assumption, namely *functional extensionality*, which says that two functions are equal if they map equal values to equal results:

```
postulate
  ext : ∀ {ℓ ℓ′} {A : Set ℓ} {B : A → Set ℓ′} {f g : (a : A) → B a} →
          (∀ (a : A) → f a ≡ g a) → f ≡ g
```

Although functional extensionality follows from univalence, it is strictly weaker and it can also be used in type theories that assume UIP. (Throughout the paper, we use Agda's ∀ syntax which allows us to omit the type annotation for some quantified variables, in this case ℓ and ℓ').

3 Leibniz Equality

Following Leibniz, two objects are equal if they satisfy the same predicates. Let a and b be objects of type A. We say that $a \doteq b$ holds if for every predicate P over type A we have that $P\ a$ implies $P\ b$.

```
module Leibniz {ℓ} {A : Set ℓ} where
  _≐_ : (a b : A) → Set (lsuc ℓ)
  _≐_ a b = (P : A → Set ℓ) → P a → P b
```

We will see shortly that this definition implies its converse: for every predicate P over type A we also have that $P\ b$ implies $P\ a$.

Leibniz equality is reflexive and transitive, where the first is shown by a variant of the identity function and the second by a variant of function composition.

```
  reflexive≐ : {a : A} → a ≐ a
  reflexive≐ P Pa = Pa

  transitive≐ : {a b c : A} → a ≐ b → b ≐ c → a ≐ c
  transitive≐ a≐b b≐c P Pa = b≐c P (a≐b P Pa)
```

Symmetry is less obvious. We have to show that if $P\ a$ implies $P\ b$ for all predicates P, then the implication holds the other way round as well. Given a specific P and a proof Pb of $P\ b$, we have to construct a proof of $P\ a$ given $a \doteq b$. To do so, we instantiate the equality with a predicate Q such that $Q\ c$ holds if $P\ c$ implies $P\ a$. The property $Q\ a$ is trivial by reflexivity, and hence $Q\ b$ follows from $a \doteq b$. But $Q\ b$ is exactly a proof of what we require, that $P\ b$ implies $P\ a$.

```
  symmetric≐ : {a b : A} → a ≐ b → b ≐ a
  symmetric≐ {a} {b} a≐b P = Qb
    where
      Q : A → Set ℓ
      Q c = P c → P a
      Qa : Q a
      Qa = reflexive≐ P
      Qb : Q b
```

```
  Qb = a≐b Q Qa

open Leibniz public
```

4 A related result

As warm up to the equivalence of Martin-Löf identity and Leibniz equality, we show a well known related result where proving an isomorphism requires parametricity (Wadler, 1989).

We encapsulate this section as an Agda module, so that we may reuse some names later.

```
module WarmUp {ℓ} (A : Set ℓ) where
```

We show that the types A and $T\ A$ are isomorphic, where $T\ A$ is defined as follows.

```
  T : Set (lsuc ℓ)
  T = ∀ (X : Set ℓ) → (A → X) → X
```

The type $T\ A$ is related to continuation passing style (CPS): a value of type A accepts a continuation of type $A \to X$ and returns a value of type X. It can also be seen as the Church encoding of the datatype Box A with a single constructor box : $A \to$ Box A.

One direction of the isomorphism will require the parametricity of type $T\ A$, which we postulate.

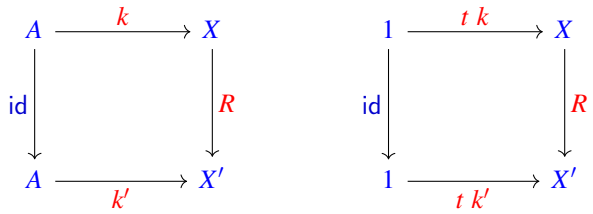
```
  postulate
    paramT : (t : T) → (X X′ : Set ℓ) (R : X → X′ → Set ℓ) →
             (k : A → X) (k′ : A → X′) (kR : (a : A) → R (k a) (k′ a)) →
             R (t X k) (t X′ k′)
```

This postulate (and the one we will encounter in the next Section) can be proven in type theories that offer an internal notion of parametricity (Nuyts *et al.*, 2017). In other type theories, a proof can be derived mechanically for any closed value of type t (Bernardy *et al.*, 2010).

To unpack the property, let R be a relation between types X and X'. Assume $a : A$ and $t : T\ A$, with functions $k : A \to X$ and $k' : A \to X'$. Parametricty for $T\ A$ tells us

$$\text{if } (k\ a, k'\ a) \in R \text{ then } (t\ k, t\ k') \in R.$$

We can diagram parametricity by



where horizontal lines represent functions and vertical lines relations.

The isomorphism between A and $T\ A$ is given by a pair of functions, i and j. The forward direction takes $a : A$ to a function that accepts $k : A \to X$ and returns $k\ a$.

```
i : A → T
i a X k = k a
```

The reverse direction takes $t : T\ A$, instantiates it at type A and applies it to the identity function at type A.

```
id : A → A
id a = a

j : T → A
j t = t A id
```

It is trivial to show that j is a (left) inverse of i.

```
ji : (a : A) → (j (i a) ≡ a)
ji a = refl
```

Showing that i is inverse to j requires an application of parametricity. First, we prove the result up to functional extensionality.

```
ij_ext : (t : T) (X : Set ℓ) (k : A → X) → (i (j t) X k ≡ t X k)
ij_ext t X k = paramT t A X R id k (λ a → refl)
  where
    R : A → X → Set ℓ
    R a x = k a ≡ x
```

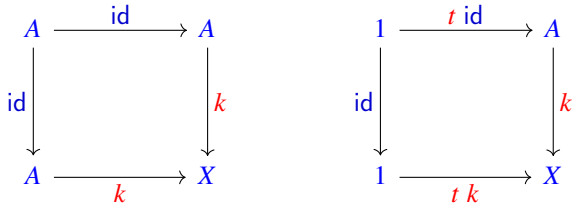
To unpack this, given $t : T\ A$ and X and $k : A \to X$, we have $\mathsf{i}\ (\mathsf{j}\ t)\ X\ k = k\ (t\ A\ \mathsf{id})$ by definition of i and j, so we need to show

$$k\ (t\ A\ \mathsf{id}) \equiv t\ X\ k$$

Recall that parametricity holds for an arbitrary relation R from X to X', and arbitrary $k : A \to X$ and $k' : A \to X'$. Instantiate X to A and k to id, and X' to X and k' to k, and choose R so that $(a,x) \in R$ iff $k\ a \equiv x$. Then parametricity instantiates to

$$\text{if}\ \ k\ a \equiv k\ a\ \ \text{then}\ \ k\ (t\ A\ \mathsf{id}) \equiv t\ X\ k.$$

The hypothesis holds trivially, and the desired result follows immediately from the conclusion. We can diagram the instance of parametricity by



where now both horizontal and vertical lines represent functions.

Applying extensionality yields the final result.

```
ij : (t : T) → (i (j t) ≡ t)
ij t = ext (λ X → ext (λ k → ij_ext t X k))
```

This completes the section and closes the module (permitting us to redefine i, j, ij, and ji in the next section).

5 $\doteq \simeq \equiv$

Following the outline of the previous section, we now prove an isomorphism between Leibniz equality and Martin-Löf identity.

One direction of the isomorphism will require the parametricity of Leibniz equality, which we postulate.

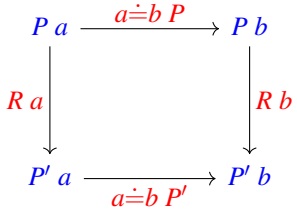
```
module MainResult {ℓ} (A : Set ℓ) where

postulate
  param�doteq : {a b : A} (a�doteqb : a ≐ b) →
               (P P′ : A → Set ℓ) → (R : (c : A) → P c → P′ c → Set ℓ) →
                 (Pa : P a) (P′a : P′ a) → R a Pa P′a →
                   R b (a�doteqb P Pa) (a�doteqb P′ P′a)
```

To unpack this, for a given $c : A$ and predicate P, P' over A, let $R\ c$ be a relation between $P\ c$ and $P'\ c$. If Pa is a proof that $P\ a$ holds, then $a \doteq b\ P\ Pa$ is a proof that $P\ b$ holds, and similarly if $P'a$ is a proof that $P'\ a$ holds. Parametricity for $\doteq$ tells us

$$\text{if } (Pa, P'a) \in R\ a \text{ then } (a \doteq b\ P\ Pa, a \doteq b\ P'\ P'a) \in R\ b.$$

We can diagram parametricity by



where horizontal lines represent functions and vertical lines relations.

The isomorphism between $a \equiv b$ and $a \doteq b$ is given by a pair of functions, i and j. In the forward direction, if we know $a \equiv b$ we need for any P to take a proof of $P\ a$ to a proof of $P\ b$, which is easy since identity of a and b implies that any proof of $P\ a$ is also a proof of $P\ b$.

```
i : {a b : A} (a≡b : a ≡ b) → a ≐ b
i refl P Pa = Pa
```

This function i is often called subst because it allows us to substitute b for a if we have a proof $a \equiv b$. It is also called transport in the context of homotopy type theory.

In the reverse direction, given that for any P we can take a proof of $P\ a$ to a proof of $P\ b$ we need to show $a \equiv b$. The proof is similar to that for symmetry of Leibniz equality. We take Q to be the predicate that holds of c if $a \equiv c$. Then $Q\ a$ is trivial by reflexivity of Martin-Löf identity, and hence $Q\ b$ follows from $a \doteq b$. But $Q\ b$ is exactly a proof of what we require, that $a \equiv b$.

```
j : {a b : A} → a ≐ b → a ≡ b
j {a} {b} a≐b = Qb
  where
    Q : A → Set ℓ
    Q c = a ≡ c
    Qa : Q a
    Qa = reflexive≡
    Qb : Q b
    Qb = a≐b Q Qa
```

It is trivial to show that j is the inverse of i.

```
ji : {a b : A} (a≡b : a ≡ b) → j (i a≡b) ≡ a≡b
ji refl = refl
```

Showing that i is inverse to j requires an application of parametricity. First, we prove the result up to extensionality.

```
ij_ext : {a b : A} (a≐b : a ≐ b) →
  (P : A → Set ℓ) (Pa : P a) → i (j a≐b) P Pa ≡ a≐b P Pa
ij_ext {a} a≐b P Pa = param≐ a≐b Q P R refl Pa refl
  where
    Q : A → Set ℓ
    Q c = a ≡ c
    R : (c : A) (Qc : Q c) (Pc : P c) → Set ℓ
    R c Qc Pc = i Qc P Pa ≡ Pc
```

To unpack this, given a proof $a \doteq b$ of $a \doteq b$, a predicate P over type A, and a proof $Pa : P\ a$, we have $\mathsf{i}\ (\mathsf{j}\ a \doteq b)\ P\ Pa = \mathsf{i}\ (a \doteq b\ Q\ \mathsf{refl})\ P\ Pa$ by definition of j, so we need to show that

$$\mathsf{i}\ (a \doteq b\ Q\ \mathsf{refl})\ P\ Pa \equiv a \doteq b\ P\ Pa$$

where $Q\ c = a \equiv c$, just as in the definition of j. Recall that parametricity holds for an arbitrary relation $R\ c$ from $P\ c$ to $P'\ c$. Instantiate P to Q and P' to P, and choose $R\ c$ so that $(Qc, Pc) \in R$ iff $\mathsf{i}\ Qc\ P\ Pa \equiv Pc$. Note that $Qa = a \equiv a$, so we can prove it by refl. Then parametricity instantiates to

$$\text{if}\ \ \mathsf{i}\ \mathsf{refl}\ P\ Pa \equiv Pa\ \ \text{then}\ \ \mathsf{i}\ (a \doteq b\ Q\ \mathsf{refl})\ P\ Pa \equiv (a \doteq b\ P\ Pa).$$

The hypothesis holds by definition of i, and the desired result follows immediately from the conclusion.

Applying extensionality yields the final result.

```
ij : {a b : A} (a≐b : a ≐ b) → i (j a≐b) ≡ a≐b
ij a≐b = ext (λ P → ext (λ pa → ij_ext a≐b P pa))
```

6 Universes and predicativity

So far, we have not paid much attention to the role of universe levels in the two definitions of equality. In this section we rectify this by studying variants of the two equalities with

more general universe levels and considering how they are related. If you don't care about universe levels, you can safely skip this section.

To avoid inconsistencies with the rule Set : Set, most type theories use a hierarchy of universes $\mathsf{Set}_0, \mathsf{Set}_1, \mathsf{Set}_2, \ldots$ where each universe is a member of the next. Universe polymorphism is a feature of proof assistants such as Agda and Coq that allows one to define types at all universe levels at once. For example, in Section 2 we defined Martin-Löf identity for types in any universe Set ℓ. Here ℓ can be instantiated to any concrete universe level $0, 1, 2, \ldots$ Agda provides two basic operations on levels: lsuc ℓ increases the level by 1 and $\ell_1 \sqcup \ell_2$ takes the maximum of two levels.

Although we have shown that Leibniz equality and Martin-Löf identity are isomorphic, there is still a small but significant difference if we look at their universe levels. Specifically, if $A : \mathsf{Set}\ \ell$ and $x, y : A$ then we have $x \equiv y : \mathsf{Set}\ \ell$ but $x \doteq y : \mathsf{Set}\ (\mathsf{lsuc}\ \ell)$. The reason for this is that Agda is a predicative theory: any type that makes use of Set ℓ must be defined at least at level lsuc ℓ. What our proof shows is that in the particular case of Leibniz equality, this was not really necessary: since it is equivalent to a type defined at level ℓ, it could be defined at universe level ℓ itself without breaking soundness of the system. This could be accomplished in a type theory with resizing rules (Voevodsky, 2011). Here we show a similar result relating variants at different levels directly in Agda.

We again begin a new module in order to reuse names from the previous section.

```
module UniversePolymorphic {ℓa} (A : Set ℓa) where
```

In the definition of Leibniz equality, both the type A and the property P have the same universe level ℓ. Here, instead, we fix A to be at level ℓ_a and permit P to be at a different level ℓ. This leads to an infinite family of variants of Leibniz equality $\doteq\langle\ \ell\ \rangle$, one for each universe level ℓ.

```
_≐⟨_⟩_ : ∀ (a : A) ℓ (b : A) → Set (ℓa ⊔ lsuc ℓ)
_≐⟨_⟩_ a ℓ b = (P : A → Set ℓ) → P a → P b
```

This definition enjoys the same basic properties of Leibniz equality $\doteq$ as defined above, e.g., reflexivity, symmetry and transitivity.

```
reflexiveℓ : ∀ {ℓ} {a : A} → a ≐⟨ ℓ ⟩ a
reflexiveℓ P pa = pa

transitiveℓ : ∀ {ℓ} {a b c : A} → a ≐⟨ ℓ ⟩ b → b ≐⟨ ℓ ⟩ c → a ≐⟨ ℓ ⟩ c
transitiveℓ a≐b b≐c P pa = b≐c P (a≐b P pa)

symmetricℓ : ∀ {ℓ} {a b : A} → a ≐⟨ ℓ ⟩ b → b ≐⟨ ℓ ⟩ a
symmetricℓ {ℓ} {a} {b} a≐b P pb = a≐b Q qa pb
  where
    Q : (c : A) → Set ℓ
    Q c = P c → P a
    qa : Q a
    qa pa = pa
```

One might think that the relation $\doteq\langle\,\ell\,\rangle$ gets smaller as ℓ increases, since we quantify over more predicates. However, an interesting consequence of our main result is that all different instances of this polymorphic Leibniz equality are actually equivalent, provided the level ℓ is at least as big as ℓ_a (the universe level of A). In other words, $a \doteq\langle\,\ell\,\rangle\, b$ and $a \doteq\langle\,\ell'\,\rangle\, b$ behave identically for all $\ell, \ell' \geq \ell_a$. Intuitively, this is because they are all equivalent to a single type, namely Martin-Löf equality $x \equiv y$. We show this formally by implementing a function converting from one instance to another, and showing it is its own inverse.

If ℓ and ℓ' are distinct, then we cannot convert directly from $a \doteq\langle\,\ell\,\rangle\, b$ to $a \doteq\langle\,\ell'\,\rangle\, b$. Instead, we will convert first from $a \doteq\langle\,\ell\,\rangle\, b$ to a universe-polymorphic variant of Martin-Löf identity. Once this is done, we only have to convert between the different versions of this Martin-Löf identity type, which will turn out to be straightforward.

First, we define the polymorphic variant of Martin-Löf identity. This type $\equiv\langle\,\ell\,\rangle$ is defined at universe level $\ell_a \sqcup \ell$, in particular its level is always at least as big as ℓ_a. (Agda also allows defining $\equiv\langle\,\ell\,\rangle$ at level ℓ or indeed even at level 0, but the theoretical foundations of this feature seems rather shaky so we refrain from using it here.) If $\ell = \ell_a$ we regain the standard definition of Martin-Löf identity.

```
data _≡⟨_⟩_ (x : A) ℓ : A → Set (ℓₐ ⊔ ℓ) where
  refl : x ≡⟨ ℓ ⟩ x
```

Since $\equiv\langle\,\ell\,\rangle$ is an inductive datatype, we can eliminate it into types of any universe. In contrast, we can eliminate $\doteq\langle\,\ell\,\rangle$ only into types of universe level exactly ℓ by its very definition.

As in the previous section, we can define a function i converting from $a \equiv\langle\,\ell\,\rangle\, b$ to $a \doteq\langle\,\ell\,\rangle\, b$:

```
i : ∀ {ℓ} {a b : A} (a≡b : a ≡⟨ ℓ ⟩ b) → a ≐⟨ ℓ ⟩ b
i refl P Pa = Pa
```

To go in the opposite direction, we want to apply the proof of Leibniz equality to the polymorphic Martin-Löf identity type, which is defined at universe level $\ell_a \sqcup \ell$. So we need to require that the level argument to $\doteq\langle\,\_\,\rangle$ is of the form $\ell_a \sqcup \ell$. This means we can convert from $a \doteq\langle\,\ell_a \sqcup \ell\,\rangle\, b$ to $a \equiv\langle\,\ell_a \sqcup \ell\,\rangle\, b$:

```
j : ∀ ℓ {a b : A} (a≐b : a ≐⟨ ℓₐ ⊔ ℓ ⟩ b) → a ≡⟨ ℓₐ ⊔ ℓ ⟩ b
j ℓ {a} {b} a≐b = a≐b P₀ refl
  where
    P₀ : A → Set (ℓₐ ⊔ ℓ)
    P₀ c = a ≡⟨ ℓₐ ⊔ ℓ ⟩ c
```

We make the argument ℓ to j explicit to help Agda with the inference of universe levels in the proofs below.

We can also implement a function k≡ converting between variants of $a \equiv\langle\,\ell\,\rangle\, b$ at different universe levels:

```
k≡ : ∀ {ℓ} ℓ' {a b : A} (p : a ≡⟨ ℓ ⟩ b) → a ≡⟨ ℓ' ⟩ b
k≡ ℓ' refl = refl
```

Using this conversion together with i and j allows us to convert between variants of $a \doteq\langle\, \ell \,\rangle\, b$ too:

```
k≐ : ∀ ℓ ℓ′ {a b : A}
  (a≐b : a ≐⟨ ℓa ⊔ ℓ ⟩ b) → a ≐⟨ ℓa ⊔ ℓ′ ⟩ b
k≐ ℓ ℓ′ a≐b = i (k≡ (ℓa ⊔ ℓ′) (j (ℓa ⊔ ℓ) a≐b))
```

To prove that all types of the form $a \doteq\langle\, \ell_a \sqcup \ell \,\rangle\, b$ are equivalent, we need to show that k≐ is its own inverse. More precisely, we will show that k≐ $\ell\ \ell'$ and k≐ $\ell'\ \ell$ are inverse to each other. Since k≐ is composed of three functions i, k≡ and j, it is sufficient to show that:

1. i and j are mutual inverses.
2. k≡ is its own inverse.

The proof of the first statement is a straightforward generalization of the proof in the previous section. For the interesting direction it relies again on parametricity of Leibniz equality:

```
postulate
  param≐ : ∀ {ℓ} {a b : A} → (a≐b : a ≐⟨ ℓ ⟩ b) →
    (P P′ : A → Set ℓ) → (R : (x : A) → P x → P′ x → Set ℓ) →
      (Pa : P a) (P′a : P′ a) → R a Pa P′a →
        R b (a≐b P Pa) (a≐b P′ P′a)

ji : ∀ ℓ {a b : A} (a≡b : a ≡⟨ ℓa ⊔ ℓ ⟩ b) → j (ℓa ⊔ ℓ) (i a≡b) ≡ a≡b
ji ℓ refl = refl

ij_ext : ∀ ℓ {a b : A} (a≐b : a ≐⟨ ℓa ⊔ ℓ ⟩ b) (P : A → Set (ℓa ⊔ ℓ)) (Pa : P a) →
      i (j (ℓa ⊔ ℓ) a≐b) P Pa ≡ a≐b P Pa
ij_ext ℓ {a} a≐b P Pa = param≐ a≐b P0 P R refl Pa refl
  where
    P0 : A → Set (ℓa ⊔ ℓ)
    P0 c = a ≡⟨ ℓa ⊔ ℓ ⟩ c
    R : (c : A) (P0c : P0 c) (Pc : P c) → Set (ℓa ⊔ ℓ)
    R c P0c Pc = i P0c P Pa ≡ Pc

ij : ∀ ℓ {a b : A} (a≐b : a ≐⟨ ℓa ⊔ ℓ ⟩ b) → i (j (ℓa ⊔ ℓ) a≐b) ≡ a≐b
ij ℓ a≐b = ext (λ P → ext (λ pa → ij_ext ℓ a≐b P pa))
```

As promised, the second statement is easy to prove by pattern matching:

```
kk≡ : ∀ {ℓ ℓ′} {a b : A} (a≡b : a ≡⟨ ℓ ⟩ b) → k≡ ℓ (k≡ ℓ′ a≡b) ≡ a≡b
kk≡ refl = refl
```

Finally, we put these components together (using cong and transitive≡) to prove that the types $a \doteq\langle\, \ell \,\rangle\, b$ are equivalent for all ℓ.

```
cong : ∀ {ℓ ℓ′} {X : Set ℓ} {Y : Set ℓ′} {a b : X} → (f : X → Y) → a ≡ b → f a ≡ f b
cong P refl = refl
```

```
kk≐ : ∀ {ℓ ℓ′} {a b : A} (a≐b : a ≐⟨ ℓa ⊔ ℓ ⟩ b) → k≐ ℓ′ ℓ (k≐ ℓ ℓ′ a≐b) ≡ a≐b
kk≐ {ℓ} {ℓ′} a≐b =
  transitive≡
    (cong i (cong (k≡ (ℓa ⊔ ℓ)) (ji ℓ′ (k≡ (ℓa ⊔ ℓ′) (j (ℓa ⊔ ℓ) a≐b)))))
    (transitive≡ (cong i (kk≡ (j (ℓa ⊔ ℓ) a≐b))) (ij ℓ a≐b))
```

7 Conclusion

We have shown that Leibniz equality and Martin-Löf identity are isomorphic, under uncontroversial assumptions of parametricity and functional extensionality. The proof is straightforward, and it was previously unknown. We wonder whether, with this proof in hand, it might not be too difficult to show the general result, conjectured by Pfenning and Paulin-Mohring (Pfenning & Paulin-Mohring, 1989), that inductive data types are isomorphic to their Church encodings in the presence of parametricity.

References

Altenkirch, Thorsten, McBride, Conor, & Swierstra, Wouter. (2007). Observational Equality, Now! *Pages 57–68 of: Workshop on Programming Languages Meets Program Verification*. ACM.

Atkey, Robert, Ghani, Neil, & Johann, Patricia. (2014). A Relationally Parametric Model of Dependent Type Theory. *Pages 503–515 of: Principles of Programming Languages*. ACM.

Bernardy, J.-P., Jansson, P., & Paterson, R. (2012). Proofs for free—parametricity for dependent types. *Journal of functional programming*, **22**(2), 107–152.

Bernardy, Jean-Philippe, Jansson, Patrik, & Paterson, Ross. (2010). Parametricity and dependent types. *Pages 345–356 of: ICFP*. ACM.

Bernardy, Jean-Philippe, Coquand, Thierry, & Moulin, Guilhem. (2015). A Presheaf Model of Parametric Type Theory. *Electronic notes in theoretical computer science*, **319**(Supplement C), 67–82.

Cockx, Jesper, Devriese, Dominique, & Piessens, Frank. (2014). Pattern matching without K. *International Conference on Functional Programming*. ACM.

Cockx, Jesper, Devriese, Dominique, & Piessens, Frank. (2016). Unifiers As Equivalences: Proof-relevant Unification of Dependently Typed Data. *Pages 270–283 of: ICFP*. ACM.

Cohen, Cyril, Coquand, Thierry, Huber, Simon, & Mörtberg, Anders. (2016). Cubical Type Theory: a constructive interpretation of the univalence axiom. *Arxiv e-prints*, Nov.

Coq Developers. (2017). *Coq documentation*.

Coquand, Thierry. (1992). Pattern matching with dependent types. *Types for Proofs and Programs*. TYPES.

Hasegawa, R. (1994). Categorical data types in parametric polymorphism. *Mathematical structures in computer science*, **4**(1), 71–109.

Hofmann, Martin, & Streicher, Thomas. (1996). The Groupoid Interpretation of Type Theory. *Pages 83–111 of: Venice Festschrift*. Oxford University Press.

Leibniz, Gottfried. (1686). *Discourse on metaphysics*.

Martin-Löf, Per. (1975). An Intuitionistic Theory of Types: Predicative Part. *Pages 73 – 118 of: Logic Colloquium*. Studies in Logic and the Foundations of Mathematics, vol. 80. Elsevier.

Nuyts, Andreas, Vezzosi, Andrea, & Devriese, Dominique. 2017 (Sept.). Parametric quantifiers for dependent type theory. *Proceedings of the ACM on functional programming*, vol. 1.

Pfenning, Frank, & Paulin-Mohring, Christine. (1989). Inductively defined types in the Calculus of Constructions. *Pages 209–228 of: Mathematical Foundations of Programming Semantics*. Lecture Notes in Computer Science. Springer, New York, NY.

Reynolds, John C. (1983). Types, abstraction, and parametric polymorphism. *Pages 513–523 of:* Mason, R. E. A. (ed), *Information Processing*. North Holland.

Streicher, Thomas. (1993). *Investigations into Intensional Type Theory*. Habilitation thesis, Ludwig-Maximilians-University Munich.

the Univalent Foundations Program. (2013). *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: https://homotopytypetheory.org/book.

Voevodsky, Vladimir. (2011). *Resizing rules – their use and semantic justification*. Talk at the Institute for Advanced Study in Princeton, NJ.

Wadler, Philip. (1989). Theorems for free. *International Conference on Functional Programming and Computer Architecture*. ACM.

Wadler, Philip. (2007). The Girard-Reynolds isomorphism (second edition). *Theoretical computer science*, **375**((1–3)), 201–226.