JFP **30**, e17, 14 pages, 2020.
doi:10.1017/S0956796820000155

THEORETICAL PEARL

$\doteq \simeq \equiv$ *Leibniz equality is isomorphic to Martin-Löf identity, parametrically*

ANDREAS ABEL
Gothenburg University
(*e-mail:* andreas.abel@gu.se)

JESPER COCKX
TU Delft, Netherlands
(*e-mail:* j.g.h.cockx@tudelft.nl)

DOMINIQUE DEVRIESE
Vrije Universiteit Brussel, Belgium
(*e-mail:* dominique.devriese@vub.be)

AMIN TIMANY
Aarhus University, Denmark
(*e-mail:* timany@cs.au.dk)

PHILIP WADLER
University of Edinburgh
(*e-mail:* 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 *vice versa*, 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 functional extensionality and Reynolds' notion of parametricity. 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 a 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. (Indeed, we also provide Coq versions of our results as an online supplement.) Such systems typically define equality in the style introduced by 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. As we discuss below, some assume that refl is the unique inhabitant of $a \equiv a$, while others do not; our results will turn out to hold either way.

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). Our results were new at the time of writing; similar results appear in Cavallo and Harper (2019), discussed below.

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 (1983). Applications of parametricity to functional programming are given by 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 (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; Nuyts & Devriese, 2018).

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. Since Martin-Löf identity and Leibniz equality are known to imply each other, it follows that extensionality with regard to Martin-Löf identity and extensionality with regard to Leibniz equality also imply each other.

We postulate extensionality in Agda. A postulate cannot be executed computationally, but that doesn't detract from our purpose.

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 we prove our results inside type theory rather than meta-theoretically (contrary to the other papers cited in this paragraph).

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 property 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.

Technically, the properties of Martin-Löf identity are expressed in terms of two axioms, called J and K. Axiom J is always assumed and corresponds to $\equiv$-elimination as originally defined by Martin-Löf (1975). Axiom K is more controversial; it corresponds to asserting *uniqueness of identity proofs* (UIP), that is, all proofs of the same equality are equal. However, UIP is incompatible with the approach taken in homotopy type theory, where one assumes Voevodsky's *univalence axiom*, that is, any two isomorphic types are equal (the Univalent Foundations Program, 2013). Consider possible proofs that the boolean type is equal to itself, $\mathsf{Bool} \equiv \mathsf{Bool}$. Under the former, there is exactly one such proof, refl, while under the latter there are two, corresponding to the two isomorphisms of Bool to itself—one which takes each of true and false to itself, and the other which swaps them. Fortunately, we do not require K for our development, as indicated by the pragma with which we begin our script; hence our results are compatible with both UIP and univalence. Axiom K was introduced by Streicher (1993); details of J and K and how to pattern match in the absence of the latter are discussed by Cockx *et al.* (2014).

Similar results to ours appear in Cavallo and Harper (2019). Their paper was written at the same time as ours and is currently under submission. They present a type theory that

combines univalence and parametricity, and prove the equivalence of Leibniz and Martin-Löf equality as one example result. Their results depend on a technical notion of *bridge-discrete* types, which we do not try to explain here. Our results are initially restricted to small types (types of the first in a hierarchy of universes), which can be taken to be bridge-discrete.[1] In Section 6 we discuss how to generalise to arbitrary universes by using parametric quantifiers, as proposed by Nuyts *et al.* (2017). Cavallo and Harper (2019) present many more results than we do, but is dense and technical. We focus solely on the equivalence between Leibniz identity and Martin-Löf equality, and aim to be accessible to a broader audience.

This note is a literate Agda script. Its source is available as an artifact and can be directly executed in Agda. In fact, the source consists of two parts, one of which can be executed in Agda, while, for reasons we will explain, the second can only be executed in Agda-parametric, the proof-of-concept implementation of Nuyts *et al.*'s parametric type theory (2017). 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 organized 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 generalizes our result to equalities between values of large types, explaining how parametricity requires additional nuance in the presence of universe types, and demonstrating one way to do this. 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, 2016; Cockx & Devriese, 2017).

```
{-# 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, at least in a version that supports the parametricity axioms we use.

We make use of modules. As required by Agda, our development is encapsulated in a module with the same name as the file containing Sections 2–5 of 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 (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, $\mathsf{refl} : a \equiv a$. Whenever a and b are *definitionally equal*, meaning that they have the same normal form, then we have $\mathsf{refl} : a \equiv b$. 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

[1] By this we mean that a type theory where all small types are bridge-discrete is compatible with the model.

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 as a parameter for all definitions contained in this module.

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

We define equality at types A in any universe Set ℓ, where ℓ is a universe level, 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 characterizes Martin-Löf identity as the least type containing refl : $a \equiv a$ for any $a : A$.

Martin-Löf identity satisfies the standard properties of an equivalence relation: reflexivity, symmetry, and transitivity.

```
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
```

Here we show these properties using *dependent pattern matching* (Coquand, 1992).

We end the module and make all its definitions available.

```
open Martin-Löf public
```

This allows us to invoke $\equiv$ at any type, not just A.

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.

3 Leibniz equality

Following Leibniz, two objects are equal if they satisfy the same predicates. Let a and b be the objects of type A. For now, we restrict our discussion to sets A that are small, i.e., of type Set, not in higher universes Set ℓ.

```
module Leibniz {A : Set} where
```

In Section 6 we will come back to this.

We say that $a \doteq b$ holds if for every predicate P over type A we have that $P\ a$ implies $P\ b$.

```
_≐_ : (a b : A) → Set₁
_≐_ a b = (P : A → Set) → 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.

```
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)
```

Here the first is shown by a variant of the identity function and the second by a variant of function composition.

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.

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

Let us unpack this. 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 what we wish to show that $P\ b$ implies $P\ a$.

Again, we end the module and make all its definitions available.

```
open Leibniz
```

4 A related result

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

```
T : Set → Set₁
T A = ∀ (X : Set) → (A → X) → X
```

The type $\mathsf{T}\ A$ is related to continuation-passing style: 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 $\mathsf{Box}\ A$ with a single constructor $\mathsf{box} : A \to \mathsf{Box}\ A$.

We encapsulate the remainder of this section as an Agda module.

```
module WarmUp (A : Set) where
```

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

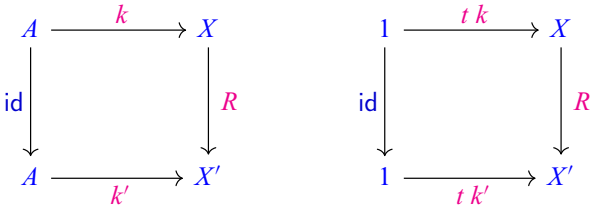
```
  postulate
    paramT : (t : T A) → (X X′ : Set) (R : X → X′ → Set) →
             (k : A → X) (k′ : A → X′) (kR : (a : A) → R (k a) (k′ a)) →
             R (t X k) (t X′ k′)
```

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 t 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 $t : \mathsf{T}\ A$, with functions $k : A \to X$ and $k' : A \to X'$. Parametricity for $\mathsf{T}\ A$ 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, and the commuting square on the left implies the one on the right.

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 A
i a X k = k a
```

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

```
id : A → A
id a = a

j : T A → 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 A) (X : Set) (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
    R a x = k a ≡ x
```

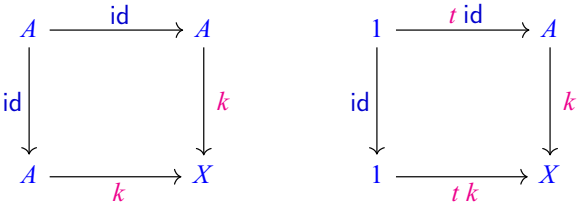
To unpack this, given $t : T\ A$ and X and $k : A \to X$, we have i (j t) X $k = k$ (t A id) by definition of i and j, so we need to show

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

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 A) → (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.

```
module MainResult (A : Set) where
```

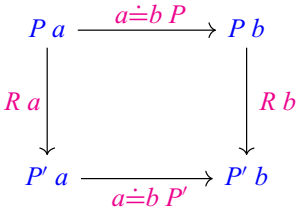
One direction of the isomorphism will require the parametricity of Leibniz equality, which we postulate. Without providing many details, the property is known to be sound, because it is validated by the model of Atkey, Ghani, and Johann (2014), although, as we shall see later, this is only true because we have restricted ourselves to small sets $A : \mathsf{Set}$.

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

To unpack this, for a given $c : A$ and predicates 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 what we wish to show that $a \equiv b$.

```
j : {a b : A} → a ≐ b → a ≡ b
j {a} {b} a≐b = Qb
  where
    Q : A → Set
    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) (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
    Q c = a ≡ c
    R : (c : A) (Qc : Q c) (Pc : P c) → Set
    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 i (j $a \doteq b$) $P\ Pa$ = i ($a \doteq b$ Q refl) $P\ Pa$ by definition of j, so we need to show that

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

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 $(Qc, Pc) \in \mathsf{R}\ c$ iff i $Qc\ P\ Pa \equiv Pc$. Note that Q $a = a \equiv a$, so we can prove it by refl. Then parametricity instantiates to

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

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 restricted our discussion of Leibniz equality $a \doteq b$ to values a and b of a small set A : Set. However, 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. In this section, we will extend our discussion to sets A in an arbitrary universe. For reasons we will explain, we will not do this in pure Agda, but in Agda-parametric, the implementation of the parametric dependent type theory of Nuyts *et al.* (2017). The previous code executes in regular Agda. To allow the code of this section to be processed

with Agda-parametric, it is contained in a separate source file. We repeat the previous pragma, start a new module, and import some of the previous definitions.

```
{-# OPTIONS –without-K #-}
module LeibnizEqualityParam where
open import LeibnizEquality
```

As we will see when we extend the definition to big A, parametricity in dependently typed languages needs a bit more nuance in the presence of higher universes. We will explain the problem and one way to resolve it. We will also consider versions of Leibniz equality that quantify over more general predicates and consider how they are related.

We will make use of universe polymorphism: 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.

First, let us try to generalize our definition of Leibniz equality to bigger A. We again begin a new module in order to reuse names from the previous section.

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

A natural generalization of Leibniz equality for bigger A can be defined as follows.

```
 module FirstAttempt where
  _≐_ : ∀ (a : A) (b : A) → Set (lsuc ℓa)
  _≐_ a b = (P : A → Set ℓa) → P a → P b
```

The corresponding generalization of the parametricity axiom param$\doteq$ looks as follows.

```
 postulate
  param≐ : {a b : A} (a≐b : a ≐ b) →
   (P P′ : A → Set ℓa) → (R : (c : A) → P c → P′ c → Set) →
    (Pa : P a) (P′a : P′ a) → R a Pa P′a →
     R b (a≐b P Pa) (a≐b P′ P′a)
```

Unfortunately, to the best of our knowledge, none of the models for parametric dependent type theory supports this axiom for general A.

Contrary to what one might think, this is not just a technical problem. Instead, it illustrates a general fact about parametricity in dependent type theories, namely that it becomes more nuanced in the presence of higher universes.

To understand why this is the case, consider the simpler example of a function f of type $(B : \mathsf{Set}) \to B \to A$. Intuitively, parametricity for f expresses that it has to work for an arbitrary set B, and consequently, that it cannot actually use the argument of type B in any way. In other words, parametricity should imply that the function f is constant.

And indeed, this is the case for small $A : \mathsf{Set}$. However, consider what happens when $A = \mathsf{Set}$ of type Set_1. In this case, a legal definition of f is $\lambda\ B\ b \to B$. That is, a lambda function that effectively does not make use of its argument b of type B, but returns the type B. Clearly, the result is a non-constant function, contradicting our intuitive parametricity result.

For similar reasons, it is far from clear that our generalized parametricity result is actually sound. Luckily, a solution is offered by the parametric quantifiers of Nuyts *et al.* (2017). The idea is to not simply assume that a function like $(B : \mathsf{Set}) \to B \to A$ cannot make non-parametric use of the argument B, but to enforce this using a parametric quantification over B. In other words, we will work with a different function space $(B \ : \{\#\}\ \mathsf{Set}) \to B \to A$ for which the language enforces parametric treatment of the argument B.

The parametric quantifiers of Nuyts *et al*. are not available in regular Agda, but an implementation called Agda-parametric is available. This is the reason why this section is defined in a separate literate Agda file that should be typechecked with Agda-parametric.

Using Nuyts *et al*.'s parametric quantifier #, we can define Leibniz equality as follows.

$$\begin{aligned}
&_\doteq_ : \forall\ (a : A)\ (b : A) \to \mathsf{Set}\ (\mathsf{lsuc}\ \ell_{\mathrm{a}}) \\
&_\doteq_\ a\ b = (P : \{\#\}\ A \to \mathsf{Set}\ \ell_{\mathrm{a}}) \to P\ a \to P\ b
\end{aligned}$$

For this alternative definition, our parametricity axiom is supported by Nuyts *et al*.'s model. In fact, instead of keeping it as an axiom, we can even prove the statement inside the Agda-parametric type theory, using special internal parametricity primitives, that we do not go into details on here.

$$\begin{aligned}
&\mathsf{param}\doteq : \{a\ b : A\}\ (a \doteq b : a \doteq b) \to \\
&(P\ P' : A \to \mathsf{Set}\ \ell_{\mathrm{a}}) \to (R : (c : A) \to P\ c \to P'\ c \to \mathsf{Set}\ \ell_{\mathrm{a}}) \to \\
&\ (Pa : P\ a)\ (P'a : P'\ a) \to R\ a\ Pa\ P'a \to \\
&\ R\ b\ (a \doteq b\ P\ Pa)\ (a \doteq b\ P'\ P'a)
\end{aligned}$$

With these modifications, our proof from the previous section goes through without additional changes, so we do not repeat it here. The full proof is in the accompanying artifact.

Although we can now show that Leibniz equality and Martin-Löf identity are isomorphic for arbitrary A, 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 $\mathsf{Set}\ \ell$ must be defined at least at level $\mathsf{lsuc}\ \ell$. 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).

7 Conclusion

We have shown that Leibniz equality and Martin-Löf identity are isomorphic, under assumptions of parametricity and functional extensionality. Along the way, we noticed how parametricity requires additional nuance in the presence of higher universes, and looked at how this can be avoided by using the explicit parametric quantifiers of Nuyts *et al.* (2017). Our proof is straightforward, and it was previously unknown. We wonder whether, with this proof in hand, it might not be too difficult to show, internally in a

type theory like Agda-parametric, the general result, conjectured by Pfenning and Paulin-Mohring(1989), that inductive data types are isomorphic to their Church encodings in the presence of parametricity.

Acknowledgments

We are indebted to the reviewers of our manuscript, whose feedback helped us to great improvements. Andreas Abel acknowledges support by the Swedish Research Council through grant 621-2014-4864 *Termination Certificates for Dependently-Typed Programs and Proofs via Refinement Types* and by the EU Cost Action CA15123 *Types for programming and verification*. Philip Wadler acknowledges support from UK EPSRC programme grant EP/K034413/1 *ABCD: A Basis for Concurrency and Distribution*. Dominique Devriese held a Postdoctoral Fellowship of the Research Fund - Flanders (FWO) during part of the development of this work.

Conflicts of Interest

None.

References

Altenkirch, T., McBride, C. & Swierstra, W. (2007) Observational equality, Now! In Workshop on Programming Languages Meets Program Verification. ACM, pp. 57–68.

Atkey, R., Ghani, N. & Johann, P. (2014) A relationally parametric model of dependent type theory. In Principles of Programming Languages (POPL). ACM, pp. 503–515.

Bernardy, J.-P., Coquand, T. & Moulin, G. (2015) A Presheaf model of parametric type theory. *Electr.Notes Theor. Comput. Sci.* **319**(Supplement C), 67–82.

Bernardy, J.-P., Jansson, P. & Paterson, R. (2010) Parametricity and dependent types. In International Conference on Functional Programming (ICFP). ACM, pp. 345–356.

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

Cavallo, E. & Harper, R. 2019. Parametric Cubical Type Theory. arXiV:1901.00489.

Cockx, J. & Devriese, D. (2017) Lifting proof-relevant unification to higher dimensions. In Certified Programs and Proofs (CPP). ACM, pp. 173–181.

Cockx, J., Devriese, D. & Piessens, F. (2014) Pattern matching without K. In International Conference on Functional Programming (ICFP). ACM.

Cockx, J., Devriese, D. & Piessens, F. (2016) Unifiers as equivalences: Proof-relevant unification of dependently typed data. In International Conference on Functional Programming (ICFP). ACM, pp. 270–283.

Cohen, C., Coquand, T., Huber, S. & Mörtberg, A. (2016) Cubical Type Theory: A constructive interpretation of the univalence axiom. Arxiv e-prints.

Coq Developers. (2017) *Coq documentation*.

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

Hasegawa, R. (1994) Categorical data types in parametric polymorphism. *Math. Struct. Comput. Sci.* **4**(1), 71–109.

Hofmann, M. & Streicher, T. (1996) The groupoid interpretation of type theory. In *Venice Festschrift*. Oxford University Press, pp. 83–111.

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

Martin-Löf, P. (1975) An intuitionistic theory of types: Predicative part. In *Logic Colloquium*. Studies in Logic and the Foundations of Mathematics, vol. 80. Elsevier, pp. 73–118.

Nuyts, A. & Devriese, D. (2018) Degrees of relatedness: A unified framework for parametricity, irrelevance, Ad Hoc polymorphism, intersections, unions and algebra in dependent type theory. In Symposium on Logic in Computer Science (LICS). ACM.

Nuyts, A., Vezzosi, A. & Devriese, D. (2017) Parametric quantifiers for dependent type theory. *Proc. ACM Program. Lang. (PACMPL)*, **1**(ICFP).

Pfenning, F. & Paulin-Mohring, C. (1989) Inductively defined types in the calculus of constructions. Lecture Notes in Computer Science. Springer, New York, NY.

Reynolds, J. C. (1983) Types, abstraction, and parametric polymorphism. In *Information Processing*, Mason, R. E. A. (ed), North Holland, pp. 513–523.

Streicher, T. (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, V. (2011) *Resizing Rules – Their Use and Semantic Justification*. Talk at the Institute for Advanced Study in Princeton, NJ.

Wadler, P. (1989) Theorems for free. In *Functional Programming Languages and Computer Architecture (FPCA)*. ACM.

Wadler, P. (2007) The Girard-Reynolds isomorphism (second edition). *Theor. Comput. Sci.* **375**(1–3), 201–226.