Toward a Cubical Type Theory Univalent by Definition

Hugo Moeneclaey,
ENS Paris-Saclay
hmoenecl@ens-paris-saclay.fr

Under the supervision of:

Hugo Herbelin,
Université Paris Diderot
hugo.herbelin@inria.fr

April - July 2019

Abstract

Cubical Type Theory [11] provides a computational meaning to Voevodsky's univalence axiom. It uses an abstract interval to characterize equality types. Our goal here is to report on our progress trying to build a variant of this theory in which an equality between types is *by definition* an equivalence. Our approach is to use equality types computed by induction on types, inspired by parametricity.

Contents

1 Introduction

This report is concerned with equalities in constructive mathematics, in the context of type theory. We give a brief and partial overview of this subject. The reader is assumed familiar with at least one type theory and with sequent calculus.

1.1 Intensional Type Theory

Type theories in the sense of Martin-Löf [19, 20, 21] are foundational systems for constructive mathematics. They are based on the Curry-Howard isomorphism, an informal principle stating that we can interpret proofs as programs and propositions as types (in the sense of computer science). Most current proof assistants are based on some type theory (e.g. Coq, Agda, Lean), and illustrate in practice the strong similarity between mathematical proofs and certified programs.

In a type theory there are types and inhabitants of these types. If we try to interpret this in the usual set-theoretic framework, types correspond to sets or propositions, and their inhabitants correspond to elements of the sets or proofs of the propositions. We write $a : A$ to say that a is an inhabitant of a type A. It should be noted that, as expected from the Curry-Howard isomorphism, inhabitants of a type can be seen as programs, meaning there are rules to compute with them. We expect these rules to obey some properties, for example if $n : \mathbb{N}$ is a natural number (not depending on variables) we expect n to compute to a numeral (i.e. an actual natural number like 2 or 7). This property is called canonicity.

Recall that our main goal is to study equalities, in this context this means to study a family of types $a =_A a'$ for any type A and $a, a' : A$, such that inhabitants of $a =_A a'$ are witnesses that a and a' behave exactly the same. These types are called identity types or equality types.

Now we describe a minimal type theory which we call Intensional Type Theory. It has three kinds of types:

- Dependent function types:

  $$(x : A) \to B$$

  with A a type and B a type (possibly depending on x). Intuitively an inhabitant of $(x : A) \to B$ is a function taking $a : A$ and giving back an inhabitant of $B[x/a]$ (this is a notation for the type B where x is replaced by a). So for example if B does not depend on x, then we are dealing with the type of functions from A to B (denoted $A \to B$ to emphasize that B does not depend on x).

  Dependent function types are also called Π-types and denoted $\Pi(x : A).B$.

- Inductive types, i.e. the smallest types having a given list of constructors. For example the smallest type A having constructors $s : A \to A$ and $0 : A$ is the type of natural numbers, denoted $\mathbb{N}$. An extensive definition for inductive types can be found in [13].

  The so-called Σ-types are important examples of inductive types, denoted:

  $$\Sigma(x : A).B$$

  with A a type and B a type (possibly depending on x). An inhabitant of such a type is a pair (a, b) with $a : A$ and $b : B[x/a]$. We denote the first (resp. second) component of $c : \Sigma(x : A).B$ by $c.1 : A$ (resp. $c.2 : B[x/c.1]$).

- A hierarchy of universes $(\mathcal{U}^n)_{n\in\mathbb{N}}$. Universes are inhabited by types, and they are assumed closed under dependent function types and inductive types. They allow us to quantify over types, for example an inhabitant of:

  $$(X:\mathcal{U}^n) \to X \to X$$

  is a family of functions from A to A for any type $A:\mathcal{U}^n$. A term taking a type as input will be called polymorphic, a good example being the polymorphic identity function:

  $$\lambda(X:\mathcal{U}^n).\,\lambda(x:X).\,x$$

  We include a whole hierarchy of universes because we want any universe $\mathcal{U}$ to be itself the inhabitant of some universe, and $\mathcal{U}:\mathcal{U}$ is inconsistent. So the most natural solution is to use a denumerable family of universes, and to say that $\mathcal{U}^0:\mathcal{U}^1$, and more generally that $\mathcal{U}^n:\mathcal{U}^{n+1}$. We will often omit the superscript in $\mathcal{U}^n$ and simply write $\mathcal{U}$.

An important observation is that Intensional Type Theory is sufficient to formalize almost all of mathematics, although not always conveniently.

Now let's go back to equalities. In Intensional Type Theory identity types in $\mathcal{U}$ are defined as an inductive family of types, by stating that it is the smallest family:

$$\_ = \_ \;:\; (X:\mathcal{U}) \to X \to X \to \mathcal{U}$$

(we denote $(\_ = \_)(A,a,b)$ by $a =_A b$ in order to stay close with the usual notations) with a constructor:

$$\mathrm{refl}:(X:\mathcal{U}) \to (x:X) \to x =_X x$$

So it should be intuitively clear that if we have an inhabitant of $a =_A b$, then a and b behave the same, i.e. $P[x/a]$ implies $P[x/b]$ for any property P depending on x.

An obvious advantage of this definition is that our system is minimal: identity types are simply examples of inductive types. It has two main drawbacks:

- Identity types can be iterated, for example if we have $p,q:a =_A b$, then we can define the type $p =_{a=_A b} q$. These iterated identity types have a subtle structure, which is inconvenient when using Intensional Type Theory in practice.

- It is not clear how to interpret the identity types $A =_{\mathcal{U}} B$, and it is hard to build inhabitants in them.

A straightforward solution to the first problem is to add an axiom stating that all iterated identity types are inhabited. While such a type theory is easier to use in practice, it still has some drawbacks, for example quotient types are not definable. Next section present the univalent foundations of mathematics, which give an interpretation to the subtle structure encountered in the first problem, and a solution to the second problem.

1.2 Univalent foundations and Homotopy Type Theory

The motivating idea for univalent foundations is an analogy between type theory and topology:

Objects in type theory	Topological interpretation
Types $A : \mathcal{U}$	Spaces
Inhabitants $a : A$	Points in A
Identity types $a =_A b$	Spaces of paths from a to b in A

This gives a meaningful intuition about the subtle structure of iterated identity types. Note that this analogy can not be made formal directly, for at least two reasons:

1. Points linked by a path are considered equal, but this is clearly not true with the usual topological spaces.
2. Universes are themselves types, whereas it is unclear how universes of spaces can be seen as spaces.

In fact a type should be interpreted as an homotopy type, intuitively a space up to continuous deformation.

The notion of homotopy type has been studied for a long time in algebraic topology. It is well-known that a lot of different objects can be used to model homotopy types, although we will not present this menagerie here. An important result is that a universe of homotopy types can be seen as an homotopy type itself by using this menagerie. So univalent foundations can be summarized as follows:

Objects in type theory	Homotopical interpretation
Types $A : \mathcal{U}$	Homotopy types
Inhabitants $a : A$	Points in A
Identity types $a =_A b$	Homotopy types of paths from a to b in A
Universes $\mathcal{U}$	Universes of homotopy types

We give two key results which give some serious ground to univalent foundations:

- A type together with its iterated identity types has a structure of homotopy type [28].
- Type theory as a whole (including a hierarchy of universes) can be interpreted in a hierarchy of universes of homotopy types [15]. We will call this the simplicial interpretation.

For A and B types, we denote by $A \simeq B$ the type of equivalences between A and B, i.e. functions from A to B which have both a right and left inverses. Then we have a canonical map from $A =_{\mathcal{U}} B$ to $A \simeq B$. The univalence axiom states that this map is an equivalence, so it can be approximately written as:

$$(A =_{\mathcal{U}} B) \simeq (A \simeq B)$$

The simplicial interpretation validates this axiom.

Remark 1. *The reader familiar with higher category theory (see e.g. [18]) will be interested to know that type theory with a univalent universe is in some sense an internal langage for higher topoi [23].*

The simplicial interpretation also validates the existence of some higher inductive types. These are variant of inductive types which are generated by some inhabitants (as for usual inductive types) together with some inhabitants of their iterated identity types. It should be noted that a general definition of higher inductive types is still lacking, although a lot of examples are known and understood. Higher inductive types can be used to model for example quotient types, or some spaces like the spheres, the tori and more generally finite CW-complexes.

We call Homotopy Type Theory the theory consisting of Intensional Type Theory together with the univalence axiom and higher inductive types. A lengthy and accessible introduction to this theory can be found in [27]. Some subtle and important geometric facts can be stated and proved within Homotopy Type Theory, for example some homotopy groups of spheres can be computed [9, 16, 17].

So univalent foundations give a new interpretation to identity types and can be implemented straightforwardly using Homotopy Type Theory. Unfortunately this theory has an axiom, so it computes poorly. For example a natural number defined using univalence rarely computes to a numeral. It is desirable to define a type theory which obeys univalence and computes well.

1.3 Cubical type theory

Cubical sets are combinatorial objects representing homotopy types. A model for type theory in cubical sets is given in [7]. This model satisfies univalence [8], but it does not support higher inductive types. Nevertheless it paved the way for the first computational interpretation of univalence [11], using the so-called Cubical Type Theory.

This theory uses dimension names denoted i, j, and so on, which are intuitively elements of an abstract interval with two endpoints 0 and 1 (think of the topological interval $[0,1]$). Then if A is a type and a is an inhabitant (possibly depending on i) of A, we have by definition a term $\lambda i.a$ of type $a[i/0] =_A a[i/1]$. Similarly if $p{:}a =_A b$, then we have a term $p(j){:}A$, with the obvious computation rule:

$$(\lambda i.a)(j) \equiv a[i/j]$$

This allows to prove function extentionality:

$$\big(\Pi(x:A).\, f(x) =_B g(x)\big) \longrightarrow f =_{A\to B} g$$

straightforwardly with the term:

$$\lambda\big(H : \Pi(x:A).\, f(x) =_B g(x)\big).\lambda i.\lambda(x:A).H(x,i)$$

More generally this guarantees that identity types in Σ and Π-types behave as expected.

Cubical Type Theory also postulates Kan compositions, a natural higher dimensional generalization of the concatenation of paths. Kan compositions compute by induction on the type in which we compose. It also postulates a somewhat complicated gluing constructor, which guarantees univalence.

This theory has a presheaf model, and therefore is consistent. It also enjoys canonicity [14] (i.e. a natural number in a context with only dimension names is a numeral). It is believed that it enjoys normalization and decidability of type checking, although to my knowledge proofs of these facts are still lacking.

Note that identity types in Cubical Type Theory compute quite differently from the usual ones. They are often called path types and denoted $\mathrm{Path}_A(x, y)$ in order to emphasize this difference (especially in type theories where path types cohabit with more traditionally behaved identity types). We will nevertheless use identity types to refer to both path types and identity types as an inductive family.

Cubical Type Theory also supports some higher inductive types, as explained in [12].

1.4 Parametricity

Our goal in this report is to sketch a variant of Cubical Type Theory. This section introduces an idea called parametricity, which is essential in the design of our theory.

Parametricity was first introduced by Reynolds [22]. It comes from the intuition that polymorphic terms in type theory treat their type inputs uniformly. This is formalized by proving that all terms send related inputs to related outputs. Now we explain what it means technically in a simple case.

We assume given X_0 and X_1 two types together with a relation:

$$X_* : X_0 \to X_1 \to \mathcal{U}$$

Then for any simple type A depending only on a type variable X, we define A_0 as $A[X/X_0]$ and A_1 as $A[X/X_1]$. Now we define inductively $A_* : A_0 \to A_1 \to \mathcal{U}$ extending $X_* : X_0 \to X_1 \to \mathcal{U}$:

$$\begin{aligned} (A \times B)_*((a_1, b_1), (a_2, b_2)) &\equiv A_*(a_1, a_2) \times B_*(b_1, b_2) && (1) \\ (A \to B)_*(f, g) &\equiv (a_0 : A_0) \to (a_1 : A_1) \to (a_* : A_*(a_0, a_1)) \\ &\quad \to B_*(f(a_0), g(a_1)) && (2) \end{aligned}$$

For any term a in A, there are two obvious terms $a_0 : A_0$ and $a_1 : A_1$. What is less obvious is that it is possible to build $a_* : A_*(a_0, a_1)$ inductively on a. In [5] this translation of a to a_* is extended to Intensional Type Theory. An important technical idea for us is to define the relation A_* by induction on A.

We sketch an interesting application. Assume we have a term:

$$f : (X : \mathcal{U}) \to X \to X$$

Then using:

$$f_* : (X_0, X_1 : \mathcal{U}) \to (R : X_0 \to X_1 \to \mathcal{U})$$

$$\to (x_0 : X_0) \to (x_1 : X_1) \to R(x_0, x_1) \to R(f(X_0, x_0), f(X_1, x_1))$$

It is easy to build a term in:

$$(X : \mathcal{U}) \to (x : X) \to (P : X \to \mathcal{U}) \to P(x) \to P(f(X, x))$$

From this one can conclude that in a set-theoretic semantic type theory, all terms of type $(X : \mathcal{U}) \to X \to X$ will be interpreted as the polymorphic identity function.

Note that the existence of a_* is external to type theory, because it is proved by induction on terms. For example the formula:

$$(f : (X : \mathcal{U}) \to X \to X) \to (X : \mathcal{U}) \to (x : X)$$

$$\to (P : X \to \mathcal{U}) \to P(x) \to P(f(X, x))$$

is not provable. Parametricity can be internalized to type theory, as indicated for example in [4, 6]. It was already noted at this point that parametric and higher dimensional type theories are linked, and they have influenced each other since then. This is manifest in [10], where a Parametric Cubical Type Theory is presented. Parametric and cubical features are introduced in a strikingly parallel fashion:

Cubical	Parametric
Dimension names: i, j, ...	Color names: $\mathbf{i}$, $\mathbf{j}$, ...
Path types: $\mathrm{Path}_A(x, y)$	Bridge types: $\mathrm{Bridge}_A(x, y)$
Paths constructor: $\lambda i.t : \mathrm{Path}_A(t[i/0], t[i/0])$	Bridges constructor: $\lambda \mathbf{i}.t : \mathrm{Bridge}_A(t[\mathbf{i}/0], t[\mathbf{i}/1])$
Univalence axiom: $\mathrm{Path}_{\mathcal{U}}(A, B) \simeq (A \simeq B)$	Relativity axiom: $\mathrm{Bridge}_{\mathcal{U}}(A, B) \simeq (A \to B \to \mathcal{U})$

So this shows that cubical techniques can be used to internalize parametricity. On the other hand it is possible to use parametric techniques to internalize higher dimensional features to type theory. For example [1] presents some progress toward a cubical type theory without interval. In this theory identity types are defined by induction on types, in the same way that relations are defined by induction on types when using parametricity. Then we have an easy and natural way to guarantee univalence: $A =_{\mathcal{U}} B$ is *defined* as $A \simeq B$. This technique is also used in [26] to generate computationally effective transport of some libraries along equivalences in Coq.

Another line of thought worth mentioning here is the use of parametricity and cubical ideas to build extensional type theories, i.e. type theories where the iterated identity types are all inhabited. An early example is Observational Type Theory [2], which uses a closed universe. The main idea is to define equalities between types and heterogeneous equalities between inhabitants by induction on the universe. This is reminiscent of parametricity. On the other hand XTT [24] is built using cubical techniques, with the rule that two paths with the same endpoints are definitionally equal.

1.5 Overview of our theory

We want to build a type theory where identity types are characterized by an abstract interval, and are computed by induction on types. We need to make both points of view cohabit harmoniously.

We will use heterogeneous equalities, i.e. equalities over equalities between types. They are defined by the following rules:

$$\frac{\Gamma \vdash \epsilon : A =_{\lambda i.\mathcal{U}} B}{\Gamma \vdash \_ =_{\epsilon} \_ : A \to B \to \mathcal{U}}$$

$$\frac{\Gamma, i \vdash a : A}{\Gamma \vdash \lambda i.a : a[i/0] =_{\lambda i.A} a[i/1]}$$

$$\frac{\Gamma, i, \Gamma' \vdash p : a =_{\epsilon} b}{\Gamma, i, \Gamma' \vdash p(i) : \epsilon(i)}$$

We write $\widehat{a}$ for $\lambda i.a$ when i does not occur in a. We also write $\_ =_A \_$ rather than $\_ =_{\widehat{A}} \_$, to emphasize the link with the usual homogeneous identity types.

Now we define the type $A \simeq B$ of equivalences between A and B. An inhabitant $\epsilon : A \simeq B$ consists of:

- A relation $\_ =_{\epsilon} \_ : A \to B \to \mathcal{U}$.
- A function $\overrightarrow{\epsilon} : A \to B$ such that for all $a : A$ we have $a =_{\epsilon} \overrightarrow{\epsilon}(a)$.
- A function $\overleftarrow{\epsilon} : B \to A$ such that for all $b : B$ we have $\overleftarrow{\epsilon}(b) =_{\epsilon} b$.
- Some additional data guaranteeing that $\overrightarrow{\epsilon}$ and $\overleftarrow{\epsilon}$ are inverse to each other.

Recall that we want to compute identity types, as a first step we add the rule:

$$(A =_{\mathcal{U}} B) \;\equiv\; (A \simeq B)$$

Moreover if $\epsilon : A =_{\mathcal{U}} B$, we identify $\_ =_{\epsilon} \_$ with the underlying relation of ϵ seen as an equivalence, justifying the overloaded notations.

Remark 2. *We add that:*

$$\overrightarrow{A} \;\equiv\; \lambda(x:A).x$$

This rule is called regularity. More generally we add rules explaining how to compute with $\widehat{A}$ seen as an equivalence.

Note that there is no known model for both univalence and regularity. The semantic side of this problem is discussed in [25], which shows that regularity is false in certain presheaves models of Cubical Type Theory. We add this principle nevertheless, although we will be careful about its uses. If it turns out to imply a contradiction it will be restrained in a reasonable way.

Note that heterogeneous identity types allow to define the types of cubes. Indeed assume given $A:\mathcal{U}$ and $p:a_1 =_A a_2$ and $q:a_3 =_A a_4$, then:

$$\lambda i.p(i) =_A q(i) : (a_1 =_A a_3) =_{\mathcal{U}} (a_2 =_A a_4)$$

and for $r:a_1 =_A a_3$ and $s:a_2 =_A a_4$, we have that:

$$r =_{\lambda i.p(i) =_A q(i)} s$$

is the type of squares in A with border depicted as:

$$\begin{tikzcd}
a_1 \arrow[r, dash, "p"] \arrow[d, dash, "r"'] & a_2 \arrow[d, dash, "s"] \\
a_3 \arrow[r, dash, "q"'] & a_4
\end{tikzcd}$$

This can be iterated to define cubes in arbitrary dimensions.

This also allows to define Kan compositions, for example for $p:a =_A b$ and $q:b =_A c$ we define:

$$p \circ q \;\equiv\; \overrightarrow{\lambda i.a =_A q(i)}(p) : a =_A c$$

which corresponds intuitively to the concatenation of the paths p and q. Note that we have a square with border depicted as:

$$\begin{tikzcd}
a \arrow[r, dash, "\widehat{a}"] \arrow[d, dash, "p"'] & a \arrow[d, dash, "p\circ q"] \\
b \arrow[r, dash, "q"'] & c
\end{tikzcd}$$

Now we need to give rules for identity types, as we already did for $A =_{\mathcal{U}} B$. For example we add:

$$\begin{aligned}
(s_1, s_2) =_{\lambda i.A\times B} (t_1, t_2) \;&\equiv\; (s_1 =_{\lambda i.A} t_1) \times (s_2 =_{\lambda i.B} t_2) && (3)\\
f =_{\lambda i.A\to B} g \;&\equiv\; (a_0 : A[i/0]) \to (a_1 : A[i/1]) \\
&\qquad \to a_0 =_{\lambda i.A} a_1 \\
&\qquad \to f(a_0) =_{\lambda i.B} g(a_1) && (4)
\end{aligned}$$

Note the similarity with Equations 1 and 2. These rules can be extended to Σ and Π-types.

Then we need to explain what to do with terms $\lambda i.t$ when they are eliminated using their reduced type. For example we add:

$$(\lambda i.(a, b)).1 \;\equiv\; \lambda i.a$$

This is well typed because $\lambda i.(a, b)$ is a path in $A \times B$, hence a pair of a path in A and a path in B, so its first projection is indeed a path in A.

Similarly $\overrightarrow{\lambda i.A \to B}$ needs to be reduced to something depending on $\lambda i.A$ and $\lambda i.B$, indeed we add:

$$\overrightarrow{\lambda i.A \to B} \;\equiv\; \lambda f.\; \overrightarrow{\lambda i.B} \circ f \circ \overleftarrow{\lambda i.A}$$

More generally we need to reduce all the components of the equivalence:

$$\lambda i.\, T(x : A).B$$

(where T is Σ or Π) to expressions depending on $\lambda i.A$ and $\lambda i.B$. Note that this should respect the rules for constant paths, for example:

$$\lambda(f : A[i/0] \to B[i/0]).\, \overrightarrow{\lambda i.B} \circ f \circ \overleftarrow{\lambda i.A}$$

should be definitionally equal to the identity function when i does not occur in A and B. This is easy to check in this case using the η-rule for arrow types.

Once we have given all these rules we know how to compute with $\lambda i.t$ whenever t is canonical (i.e. begins by a constructor). So it is reasonable to hope that λi can be eliminated from closed terms in suitable types, although of course λi can not be eliminated when there are type variables.

The fact that identity types can be computed has some pleasant consequences, for example if we have $A, B : \mathcal{U}$ depending on i, we do not need to explain how $\lambda i.\, A =_{\mathcal{U}} B$ computes, because this term is equal to $\lambda i.\, A \simeq B$, so we know how it computes by using the rules for Σ and Π-types.

2 Definition

We give a precise definition for our theory, filling the details missing from Section 1.5.

2.1 Basic theory

First we present a type theory with Π, Σ (with η-rules) and a hierarchy of universes à la Russell, without identity types. This is standardl.

Recall that we write the syntactic substitution of x by a in A as $A[x/a]$.

2.1.1 A core type theory with universes

We use a hierarchy of universes $\mathcal{U}^n$ indexed by external natural numbers.

There are two kinds of judgments:

$$\Gamma \vdash$$

meaning that Γ is well formed and:

$$\Gamma \vdash a \equiv b : A$$

meaning that a and b are terms of type A, and that they are definitionally equal. We write $\Gamma \vdash a : A$ as a shorthand for $\Gamma \vdash a \equiv a : A$.

We add the rules for axioms and conversion. The first one says that the empty context is well-formed.

$$\frac{}{\vdash}$$

$$\frac{\Gamma \vdash A : \mathcal{U}^n}{\Gamma, x : A \vdash}$$

$$\frac{\Gamma, x : A, \Gamma' \vdash}{\Gamma, x : A, \Gamma' \vdash x : A}$$

$$\frac{\Gamma \vdash a \equiv a' : A \qquad \Gamma \vdash A \equiv B : \mathcal{U}^n}{\Gamma \vdash a \equiv a' : B}$$

We omit the rules stating that definitional equality is an equivalence and congruence. From now on we will omit the hypothesis of wellformedness for context, so that we will never write:

$$\Gamma \vdash$$

again.

We add the following rules for the universes, one for each natural number n:

$$\frac{}{\Gamma \vdash \mathcal{U}^n : \mathcal{U}^{n+1}}$$

2.1.2 Σ-types

We add the following rules:

$$\frac{\Gamma \vdash A : \mathcal{U}^m \qquad \Gamma, x : A \vdash B : \mathcal{U}^n}{\Gamma \vdash \Sigma(x : A).B : \mathcal{U}^{\max(m,n)}}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : B[x/a]}{\Gamma \vdash (a, b) : \Sigma(x : A).B}$$

$$\frac{\Gamma \vdash c : \Sigma(x : A).B}{\Gamma \vdash c.1 : A}$$

$$\frac{\Gamma \vdash c : \Sigma(x : A).B}{\Gamma \vdash c.2 : B[x/c.1]}$$

Then we add the usual computation rules:

$$(a, b).1 \;\equiv\; a \tag{5}$$
$$(a, b).2 \;\equiv\; b \tag{6}$$

Together with the η-rule:

$$(c.1, c.2) \;\equiv\; c \tag{7}$$

We write $A \times B$ for $\Sigma(x : A).B$ when x does not occur in B.

2.1.3 Π-types

We add the following rules:

$$\frac{\Gamma \vdash A : \mathcal{U}^m \qquad \Gamma, x : A \vdash B : \mathcal{U}^n}{\Gamma \vdash \Pi(x : A).B : \mathcal{U}^{\max(m,n)}}$$

$$\frac{\Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda(x : A).\, b : \Pi(x : A).B}$$

$$\frac{\Gamma \vdash f : \Pi(x : A).B \qquad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B[x/a]}$$

We add the usual computation rule:

$$(\lambda(x : A).\, b)(a) \;\equiv\; b[x/a] \tag{8}$$

Together with the η-rule, valid when x does not occur in f:

$$\lambda(x : A).\, f(x) \;\equiv\; f \tag{9}$$

We write $A \to B$ for $\Pi(x : A).B$ when x does not occur in B. We often write $(x : A) \to B$ rather than $\Pi(x : A).B$.

2.2 Univalent identity types

Now we define identity types similar to path types in Cubical Type Theory, and add univalence by definition.

2.2.1 The identity types

We assume given some dimension names i, j, and so on.

We define heterogeneous identity types, the constructor λi and the application of a path to i:

$$\frac{\Gamma \vdash \epsilon : A =_{\lambda i.\mathcal{U}^n} B}{\Gamma \vdash \_ =_{\epsilon} \_ : A \to B \to \mathcal{U}^n}$$

$$\frac{\Gamma, i \vdash a : A}{\Gamma \vdash \lambda i.a : a[i/0] =_{\lambda i.A} a[i/1]}$$

$$\frac{\Gamma, i, \Gamma' \vdash p : a =_{\epsilon} b}{\Gamma, i, \Gamma' \vdash p(i) : \epsilon(i)}$$

Remark 3. *In these rules $a[i/0]$ and $a[i/1]$ are defined by induction on a. The main case is $p(i)$ with $p : a =_{\epsilon} b$, for which we define $(p(i))[i/0]$ as $a[i/0]$ and $(p(i))[i/1]$ as $b[i/1]$. Otherwise $[i/0]$ and $[i/1]$ simply go through the terms like a substitution.*

We add the obvious computation rule:

$$(\lambda i.\, a)(j) \ \equiv\ a[i/j] \tag{10}$$

Together with an η-rule for paths, valid when i does not occur in p:

$$\lambda i.\, p(i) \ \equiv\ p \tag{11}$$

Remark 4. *The two first rules look circular. More precisely it is not obvious that types are well-formed in:*

$$Form_{=}\ \frac{\Gamma \vdash \epsilon : A =_{\lambda i.\mathcal{U}^n} B}{\Gamma \vdash \_ =_{\epsilon} \_ : A \to B \to \mathcal{U}^n}$$

$$Intro_{=}\ \frac{\Gamma, i \vdash a : A}{\Gamma \vdash \lambda i.a : a[i/0] =_{\lambda i.A} a[i/1]}$$

If we say that $Intro_{=}$ is of level n when its premise is in $\mathcal{U}^n$, and $Form_{=}$ is of level n when its conclusion is in $\mathcal{U}^n$. Then $Intro_{=}$ and $Form_{=}$ at level n are well-formed using $Intro_{=}$ and $Form_{=}$ at level $n+1$. This is the same situation than for $\mathcal{U}^n : \mathcal{U}^{n+1}$, rules of level n are well-formed because of rules of level $n+1$.

We denote by $\widehat{a}$ the term $\lambda i.a$ when i does not occur in a, and we denote $\_ =_{\widehat{A}} \_$ by $\_ =_A \_$.

2.2.2 Equivalences

We use the definition of equivalence from [1].

Definition 1. *We define an equivalence between A and B in $\mathcal{U}^n$ as the data of:*

- *A relation $R : A \to B \to \mathcal{U}^n$*
- *A function $f : A \to B$ such that for all $a : A$, we have $F(a) : R(a, f(a))$.*
- *A function $g : B \to A$ such that for all $b : B$, we have $G(b) : R(g(b), b)$.*
- *For any $a : A$, $b : B$ and $p : R(a, b)$:*
  - *An element $\overrightarrow{\mathrm{coh}}(p) : f(a) =_B b$ such that $F(a) =_{\lambda i.R(a,\overrightarrow{\mathrm{coh}}(p)(i))} p$.*
  - *An element $\overleftarrow{\mathrm{coh}}(p) : a =_A g(b)$ such that $p =_{\lambda i.R(\overleftarrow{\mathrm{coh}}(p)(i),b)} G(b)$.*

We denote by $A \simeq B$ the type of equivalences between A and B. Note that it is is clearly definable in our theory.

2.2.3 Univalence by definition

We add the rule:

$$(A =_{\mathcal{U}^n} B) \;\equiv\; (A \simeq B) \tag{12}$$

For $\epsilon : A =_{\mathcal{U}^n} B$, we add that the underlying relation of ϵ is definitionally equal to $\_ =_\epsilon \_$.

Remark 5. *We did not define $\_ =_\epsilon \_$ as the underlying relation of an equivalence because heterogeneous equalities are already needed in the definition of equivalence.*

Remark 6. *Note that for $A, B : \mathcal{U}^n$, both $A =_{\mathcal{U}^n} B$ and $A \simeq B$ are in $\mathcal{U}^{n+1}$. This is in contrast with the usual definitions of equivalence, where $A \simeq B$ is in $\mathcal{U}^n$.*

Now we will give notations for the elements extracted from an equivalence $\epsilon : A =_{\mathcal{U}^n} B$. We only give their types, using $\{x : A\} \to B$ to mean that x is an implicit argument:

$$\overrightarrow{\epsilon} : A \to B \tag{13}$$
$$\overset{\rightrightarrows}{\epsilon} : (x : A) \to x =_\epsilon \overrightarrow{\epsilon}(x) \tag{14}$$
$$\overrightarrow{\mathrm{coh}_\epsilon} : \{x : A\} \to \{y : B\} \to (p : x =_\epsilon y) \to \overrightarrow{\epsilon}(x) =_B y \tag{15}$$
$$\overset{\rightrightarrows}{\mathrm{coh}}_\epsilon : \{x : A\} \to \{y : B\} \to (p : x =_\epsilon y) \to \widehat{x} =_{\lambda i. \overset{\rightrightarrows}{\epsilon}(x)(i) =_{\epsilon(i)} p(i)} \overrightarrow{\mathrm{coh}_\epsilon}(p) \tag{16}$$
$$\overleftarrow{\epsilon} : B \to A \tag{17}$$
$$\overset{\leftleftarrows}{\epsilon} : (y : B) \to \overleftarrow{\epsilon}(y) =_\epsilon y \tag{18}$$
$$\overleftarrow{\mathrm{coh}_\epsilon} : \{x : A\} \to \{y : B\} \to (p : x =_\epsilon y) \to x =_A \overleftarrow{\epsilon}(y) \tag{19}$$
$$\overset{\leftleftarrows}{\mathrm{coh}}_\epsilon : \{x : A\} \to \{y : B\} \to (p : x =_\epsilon y) \to \overleftarrow{\mathrm{coh}_\epsilon}(p) =_{\lambda i. p(i) =_{\epsilon(i)} \overset{\leftleftarrows}{\epsilon}(y)(i)} \widehat{y} \tag{20}$$

Note that these are only notations, in the sense that $A \simeq B$ is in fact a Σ-type. For example if $\epsilon : A =_{\mathcal{U}^n} B$, then the term $\epsilon.1$ makes sense and is of type $A \to B \to \mathcal{U}^n$.

Note that $\overset{\rightrightarrows}{\mathrm{coh}}_\epsilon$ is not the square H appearing in the definition of equivalence, but rather its transpose $\lambda i.\lambda j.H(j,i)$.

We denote by $\langle \_, \cdots, \_ \rangle$ the constructor of equivalences taking nine arguments. We have computation rules, which can be deduced from the definition of equivalences. For example:

$$a =_{\langle R, \cdots \rangle} b \;\equiv\; R(a,b)$$

and

$$\overrightarrow{\langle R, f, \cdots \rangle} \;\equiv\; f$$

2.2.4 Rules for constant path types

For $A : \mathcal{U}^n$, the term $\widehat{A} : A =_{\mathcal{U}^n} A$ can be seen as an equivalence. We add some rules to compute its components:

$$\overrightarrow{\widehat{A}} \;\equiv\; \lambda(x : A).\, x \tag{21}$$
$$\overset{\rightrightarrows}{\widehat{A}} \;\equiv\; \lambda(x : A).\, \widehat{x} \tag{22}$$
$$\overrightarrow{\mathrm{coh}_{\widehat{A}}} \;\equiv\; \lambda(p : a =_{\widehat{A}} b).\, \overrightarrow{\lambda i. a =_A p(i)}(\widehat{a}) \tag{23}$$
$$\overset{\rightrightarrows}{\mathrm{coh}}_{\widehat{A}} \;\equiv\; \lambda(p : a =_{\widehat{A}} b).\, \overset{\rightrightarrows}{\lambda i. a =_A p(i)}(\widehat{a}) \tag{24}$$

Remark 7. *As explained in the introduction one should be careful about those rules, as there is no known univalent model for them.*

We omit the rules in the other direction, i.e. for $\overleftarrow{\widehat{A}}$, $\overset{\leftleftarrows}{\widehat{A}}$, $\overleftarrow{\mathrm{coh}_{\lambda i.A}}$ and $\overset{\leftleftarrows}{\mathrm{coh}}_{\lambda i.A}$. From now on we will do this systematically.

2.3 Toward full computation

At this point, paths in a universe can be built in two ways, as an equality using λi or as an equivalence using $\langle \_, \cdots, \_ \rangle$. In this section we indicate how to compute with equalities seen as equivalences. We will generalize these computations to all identity types.

2.3.1 The goal

For each type constructor T taking A and B as an argument (i.e. T is Σ or Π), we need to give:

- A way to reduce:
$$s =_{\lambda i.T(A,B)} t$$
to something depending on $\lambda i.A$ and $\lambda i.B$. We denote its constructors and eliminators by $\mathbf{cons}_=$ and $\mathbf{elim}_=$.

- We should also reduce $\overrightarrow{\lambda i.T(A,B)}$ (resp. $\overrightarrow{\overrightarrow{\lambda i.T(A,B)}}$) to something depending on $\lambda i.A$ and $\lambda i.B$. Moreover the given reduct must be the identity (resp. a constant path) when i does not occur in A or B.

- We do not need to give anything for $\overrightarrow{\mathrm{coh}_{\lambda i.T(A,B)}}$ and $\overrightarrow{\overrightarrow{\mathrm{coh}_{\lambda i.T(A,B)}}}$, as for $p : s =_{\lambda i.T(A,B)} t$ we can define them:
$$\overrightarrow{\mathrm{coh}_{\lambda i.T(A,B)}}(p) \overset{\triangle}{\equiv} \overrightarrow{\lambda j.\overrightarrow{\overrightarrow{\lambda i.T(A,B)}}(s)(j) =_{T(A,B)[i/j]} p(j)}(\widehat{s}) \qquad (25)$$
$$\overrightarrow{\overrightarrow{\mathrm{coh}_{\lambda i.T(A,B)}}}(p) \overset{\triangle}{\equiv} \overrightarrow{\overrightarrow{\lambda j.\overrightarrow{\overrightarrow{\lambda i.T(A,B)}}(s)(j) =_{T(A,B)[i/j]} p(j)}}(\widehat{s}) \qquad (26)$$

  These expressions can be simplified using the fact that:
$$\overrightarrow{\overrightarrow{\lambda i.T(A,B)}}(s)(j) =_{T(A,B)[i/j]} p(j)$$
can be simplified.

  This will guarantee that the rules for $\overrightarrow{\mathrm{coh}}$ and $\overrightarrow{\overrightarrow{\mathrm{coh}}}$ are respected when i does not occur in A or B.

- Now we face again the same situation: terms in $s =_{\lambda i.T(A,B)} t$ can be built either using λi or using $\mathbf{cons}_=$. Therefore for $c : T(A,B)$ depending on i, the term $\mathbf{elim}_=(\lambda i.c)$ should be expressed depending on c.

It seems reasonable to expect reductions for $\mathbf{cons}_=(c)(j) : T(A,B)[i/j]$. We include the natural such reduction for Σ-types, but we did not find any for Π-types, and neither for $\langle \_, \cdots, \_ \rangle(j)$. We discuss this further in Section 3.1.

2.3.2 Preliminary: connections

We give an auxiliary definition of certain squares of types built out of a path, which are usually called connections.

Lemma 1. *Assume given $\epsilon : A =_{\mathcal{U}} B$. Then we can build:*

$$\epsilon\rfloor : \widehat{A} =_{\lambda i.A =_{\mathcal{U}} \epsilon(i)} \epsilon \tag{27}$$

$$\lceil\epsilon : \epsilon =_{\lambda i.\epsilon(i) =_{\mathcal{U}} B} \widehat{B} \tag{28}$$

We denote $(\epsilon\rfloor)(i,j)$ by $\epsilon(i \cap j)$ and $(\lceil\epsilon)(i,j)$ by $\epsilon(i \cup j)$.

Proof. We define $\delta : A =_{\mathcal{U}} B$ as $\overrightarrow{\lambda i.A =_{\mathcal{U}} \epsilon(i)}(\widehat{A})$. Then we can fill the square:

$$\begin{tikzcd}
A \arrow[r, dash] \arrow[d, dash] \arrow[dr, phantom, "H"] & A \arrow[d, dash, "\delta"] \\
A \arrow[r, dash, "\epsilon"'] & B
\end{tikzcd}$$

where H is defined as $\overrightarrow{\overrightarrow{\lambda i.A =_{\mathcal{U}} \epsilon(i)}}(\widehat{A})$.
We use a Kan composition to fill the inner square in:

$$\begin{tikzcd}
 & A \arrow[r, dash] \arrow[d, dash] & A \arrow[d, dash] & \\
A \arrow[r, dash] \arrow[d, dash] & A \arrow[r, dash] \arrow[d, dash] & A \arrow[r, dash] \arrow[d, dash, "\epsilon"'] \arrow[dr, phantom, "H"] & A \arrow[d, dash] \\
A \arrow[r, dash] & A \arrow[r, dash, "\epsilon"] \arrow[d, dash] \arrow[dr, phantom, "H"] & B \arrow[r, dash, "\delta"] \arrow[d, dash, "\delta"] & A \\
 & A \arrow[r, dash] & A &
\end{tikzcd}$$

This defines $\epsilon\rfloor$. The term $\lceil\epsilon$ is defined similarly.

□

Remark 8. *We explain the intuition behind the notation $\epsilon(i \cup j)$.*

If we use the analogy between a dimension name i and an element in the topological interval $[0,1]$, then $\epsilon : A =_{\mathcal{U}} B$ is a function from $[0,1]$ to $\mathcal{U}$ and $\epsilon\rfloor$ can be seen as the function which takes i and j in $[0,1]$ and outputs $\epsilon(\max(i,j))$. This justify the notation $\epsilon(i \cup j)$.

Next lemma uses regularity.

Lemma 2. *We have that:*

$$\widehat{A}(i \cap j) \equiv A \tag{29}$$

$$\widehat{A}(i \cup j) \equiv A \tag{30}$$

when i and j do not occur in A.

Definition 2. *For $\epsilon : A =_{\mathcal{U}} B$ and $p : a =_\epsilon b$, we define:*

$$\overleftarrow{\mathrm{coh}}^2_\epsilon(p) \overset{\triangle}{\equiv} \overleftarrow{\lambda i.a =_{\lambda j.\epsilon(i \cap j)} \overset{\leftleftarrows}{\epsilon}(b)(i)}(p) : a =_{A[i/0]} \overleftarrow{\epsilon}(b) \tag{31}$$

$$\overleftarrow{\overleftarrow{\mathrm{coh}}}^2_\epsilon(p) \overset{\triangle}{\equiv} \overleftarrow{\overleftarrow{\lambda i.a =_{\lambda j.\epsilon(i \cap j)} \overset{\leftleftarrows}{\epsilon}(b)(i)}}(p) : \overleftarrow{\mathrm{coh}}^2_\epsilon(p) =_{\lambda i.a =_{\epsilon(i \cap j)} \overset{\leftleftarrows}{\epsilon}(b)(i)} p \tag{32}$$

We have the following computation rules:

$$\overleftarrow{\mathrm{coh}}^2_{\widehat{A}}(p) \equiv p \tag{33}$$

$$\overleftarrow{\overleftarrow{\mathrm{coh}}}^2_{\widehat{A}}(p) \equiv \widehat{p} \tag{34}$$

Note that $\overleftarrow{\mathrm{coh}}^2_\epsilon(p)$ has the same type as $\overleftarrow{\mathrm{coh}}_\epsilon(p)$ but computes differently, and $\overleftarrow{\overleftarrow{\mathrm{coh}}}^2_\epsilon(p)$ does not have the same type as $\overleftarrow{\overleftarrow{\mathrm{coh}}}_\epsilon(p)$.

2.3.3 Σ-types

Now we give the rules for equalities in Σ-types.

$$
\begin{aligned}
(a,b) =_{\lambda i.\Sigma(x:A).B} (a',b') &\equiv \Sigma(p : a =_{\lambda i.A} a').\, b =_{\lambda i.B[x/p(i)]} b' && (35)\\
\overrightarrow{\lambda i.\Sigma(x:A).B}(a,b) &\equiv \left(\overrightarrow{\lambda i.A}(a), \overrightarrow{\lambda i.B[x/\overset{\rightrightarrows}{\lambda i.A}(a)(i)]}(b)\right) && (36)\\
\overset{\rightrightarrows}{\lambda i.\Sigma(x:A).B}(a,b) &\equiv \left(\overset{\rightrightarrows}{\lambda i.A}(a), \overset{\rightrightarrows}{\lambda i.B[x/\overset{\rightrightarrows}{\lambda i.A}(a)(i)]}(b)\right) && (37)\\
(\lambda i.c).1 &\equiv \lambda i.(c.1) && (38)\\
(\lambda i.c).2 &\equiv \lambda i.(c.2) && (39)\\
(p,q)(i) &\equiv (p(i),q(i)) && (40)
\end{aligned}
$$

Remark 9. *As explained earlier, the conversion rule for* $\overrightarrow{\mathrm{coh}_{\lambda i.\,\Sigma(x:A).B}}(p_1,p_2)$ *with* $p_1 : a =_{\lambda i.A} a'$ *and* $p_2 : b =_{\lambda i.B[x/p_1(i)]} b'$ *can be deduced from the previous rules. We give it explicitly:*

$$
\overrightarrow{\mathrm{coh}_{\lambda i.\,\Sigma(x:A).B}}(p_1,p_2)
$$

$$
\equiv \left(\overrightarrow{\delta}\,(\widehat{a})\ ,\ \overrightarrow{\lambda j.\overset{\rightrightarrows}{\lambda i.B[x/\overset{\rightrightarrows}{\lambda i.A}(a)]}(b)(j) =_{\lambda i.B[i/j][x/\overset{\rightrightarrows}{\delta}\,(\widehat{a})(j,i)]} p_2(j)}(\widehat{b}) \right)
$$

where δ *is* $\lambda j.\overset{\rightrightarrows}{\lambda i.A}(a)(j) =_{A[i/j]} p_1(j)$.

2.3.4 Π-types

Now we explain how identity types in Π-types behave.

$$
\begin{aligned}
f =_{\lambda i.\Pi(x:A).B} f' &\equiv (a_0 : A[i/0]) \to (a_1 : A[i/1])\\
&\quad \to (a_* : a_0 =_{\lambda i.A} a_1)\\
&\quad \to f(a_0) =_{\lambda i.B[x/a_*(i)]} f'(a_1) && (41)\\
\overrightarrow{\lambda i.\Pi(x:A).B}(f) &\equiv \overrightarrow{\lambda i.B[x/\overset{\leftleftarrows}{\lambda i.A}(a)(i)]} \circ f \circ \overleftarrow{\lambda i.A} && (42)\\
\overset{\rightrightarrows}{\lambda i.\Pi(x:A).B}(f) &\equiv \text{See below} && (43)\\
(\lambda i.f)(a_0,a_1,a_*) &\equiv \lambda i.f(a_*(i)) && (44)
\end{aligned}
$$

Note that we did not find a satisfying rule for:

$$
(\lambda a_0, a_1, a_*.\, t)(j) : \Pi(x : A[i/j]).B[i/j]
$$

We discuss this problem in Section 3.1.

Lemma 3. *Assume* $\Gamma, i \vdash A : \mathcal{U}$ *and* $\Gamma, i, x : A \vdash B : \mathcal{U}$.

Assume given $f : (x : A[i/0]) \to B[i/0]$.

Then for any $a : A[i/0]$, $a' : A[i/1]$ *and* $p : a =_{\lambda i.A} a'$ *there exists:*

$$\psi_f(a, a', p) : f(a) =_{\lambda i.B[x/p(i)]} \overrightarrow{\lambda i.B[x/\overset{\Longleftarrow}{\lambda i.A}(a)(i)]}\left(f(\overleftarrow{\lambda i.A}(a'))\right)$$

Moreover if i *does not occur in* A *and* B*, then:*

$$\psi_f(a, a', p) \;\equiv\; \lambda k.f(p(k))$$

Proof. We denote:

$$f(a) =_{\lambda i.B[x/p(i)]} \overrightarrow{\lambda i.B[x/\overset{\Longleftarrow}{\lambda i.A}(a')(i)]}(f(\overleftarrow{\lambda i.A}(a')))$$

by $\Psi(a, a', p)$, so that our goal is to give a term in $\Psi(a, a', p)$. Recall that $\mathrm{coh}^2_{\lambda i.A}$ first occurred in Definition 2. Then we define:

$$r \;\overset{\triangle}{\equiv}\; \overleftarrow{\mathrm{coh}^2_{\lambda i.A}}(p) : a =_{\widehat{A_{\{0/i\}}}} \overleftarrow{\lambda i.A}(a')$$

$$H \;\overset{\triangle}{\equiv}\; \overset{\Longleftarrow}{\mathrm{coh}^2_{\lambda i.A}}(p) : r =_{\lambda i.a =_{\lambda j.A(i \cap j)} \overset{\Longleftarrow}{\lambda i.A}(a')(i)} p$$

$$G \;\overset{\triangle}{\equiv}\; \lambda i.f(a) =_{\lambda j.B[x/H(i,j)]} \overset{\Longrightarrow}{\lambda i.B[x/\overset{\Longleftarrow}{\lambda i.A}(a')(i)]}\left(f(\overleftarrow{\lambda i.A}(a'))\right)(i)$$

So that:

$$G : \left(f(a) =_{\lambda j.B[x/r(j)]} f(\overleftarrow{\lambda i.A}(a'))\right) =_{\mathcal{U}^m} \Psi(a, a'p)$$

And finally we define:

$$\psi_f(a, a', p) \;\overset{\triangle}{\equiv}\; \overrightarrow{G}(\lambda k.f(r(k)))$$

If i does not occur in A and B, then $r \equiv p$ and H is just the constant path $\widehat{p}$. Therefore G is a constant path and $\psi(a, a', p) \equiv \lambda k.f(p(k))$. □

We are now ready to add the rule:

$$\overset{\Rrightarrow}{\lambda i.\Pi(x : A).B}(f) \;\equiv\; \lambda a_0, a_1, a_*.\; \psi_f(a_0, a_1, a_*) \tag{45}$$

Remark 10. *If* i *does not occur in* A *and* B*, then:*

$$\overset{\Rrightarrow}{\lambda i.\Pi(x : A).B}(f) \;\equiv\; \lambda a_0, a_1, a_*.\lambda k.f(a_*(k))$$

But regularity requires this expression to be a constant path. This can be checked:

$$\widehat{f} \;\equiv\; \lambda a_0, a_1, a_*.\, (\lambda k.f)(a_0, a_1, a_*) \;\equiv\; \lambda a_0, a_1, a_*.\lambda k.f(a_*(k))$$

The first conversion is the η*-rule for* Π*-types, and the second conversion is a consequence of Equation 44.*

3 Discussion

In this section we include several informal discussions about our system and its possible extensions.

3.1 Computations in our system

In this section we discuss computations in our system, and we provide a tentative syntax for normal forms.

3.1.1 Definitional isomorphisms and Σ-types

The following notion will be useful in the next discussion.

Definition 3. *Two types A and B are called definitionally isomorphic if there exist $f : A \to B$ and $g : B \to A$ such that:*

$$g \circ f \;\equiv\; \lambda(a : A).\, a$$

and

$$f \circ g \;\equiv\; \lambda(b : B).\, b$$

A guiding principle is that it is innocuous to identify two types definitionally isomorphic. In this case we say that the identification is natural. The identity types in Σ-types provide a good example, indeed consider:

$$(a, b) =_{\lambda i.\, \Sigma(x:A).B} (a', b')$$

and

$$\Sigma(p : a =_{\lambda i.A} a').\, (b =_{\lambda i.B[x/p(i)]} b')$$

Denoting the first type by A and the second one by B, we can show that they are definitionally isomorphic using:

$$\lambda H.\big(\lambda i.(H(i).1), \lambda i.(H(i).2)\big) : A \to B \tag{46}$$

$$\lambda G.\lambda i.\big(G.1(i), G.2(i)\big) : B \to A \tag{47}$$

Therefore the conversion rule:

$$(a, b) =_{\lambda i.\, \Sigma(x:A).B} (a', b') \;\equiv\; \Sigma(p : a =_{\lambda i.A} a').\, (b =_{\lambda i.B[x/p(i)]} b')$$

is a natural identification. From the definitional isomorphism it is easy to deduce the rules to compute with $(p, q)(i)$, $(\lambda i.c).1$ and $(\lambda i.c).2$. Conversely one can find the definitional isomorphism from the computation rules, both problems are equivalent.

3.1.2 Heterogeneous function extentionality

Now we inspect the identification:

$$f =_{\lambda i.\Pi(x:A).B} f'$$

$$\equiv\ (a_0 : A_{\{0/i\}}) \to (a_1 : A_{\{1/i\}}) \to (a_* : a_0 =_{\lambda i.A} a_1) \to f(a_0) =_{\lambda i.B[x/a_*(i)]} f'(a_1)$$

We denote the second type by $\mathrm{Ext}(f,g)$. Then we have:

$$\lambda H.\lambda a_0, a_1, a_*.\lambda i.H(i, p(i)) : f =_{\lambda i.\Pi(x:A).B} f' \to \mathrm{Ext}(f,g) \tag{48}$$

But we were not able to find a suitable function the other way yet. This explain the incomplete computation rules for Π-types.

The first try for a term in:

$$\mathrm{Ext}(f,g) \to f =_{\lambda i.\Pi(x:A).B} f'$$

would be:

$$\lambda G.\lambda i.\lambda x.G(\lambda i.x)(i)$$

but this is not well-typed because of $\lambda i.x$. The general problem is that in a context $i, x : A$ we do not know how to see x as a path depending on i.

We believe this identification should be natural, so that we need to either find a definitional isomorphism or extend our system with some construction playing the role of $\lambda i.x$ in the first try. Note that we did find maps in:

$$\mathrm{Ext}(f,g) \to f =_{\lambda i.\Pi(x:A).B} f'$$

without the required computational rules, which is encouraging. On the other hand our attempts to build a reasonable system with $\lambda i.x$ for x a variable declared after i lead into unknown syntactic territories, so we stopped our investigations.

In the rest of the discussion we will admit a solution to this problem.

3.1.3 Normal forms

In order to analyze the normal forms in a type theory, it is common to define syntactically a set of values V (intuitively outputs of algorithm, which can compute if used in another program) and a set of neutral terms N (intuitively terms that cannot compute in any context, usually because they are built from variables). We will give a guess for V and N in our theory.

We introduce a new notation $\mathrm{Equiv}(\epsilon)$, which is the second projection of $\epsilon : A =_{\mathcal{U}} B$. So it consists of all the data making $\_ =_\epsilon \_ : A \to B \to \mathcal{U}$ the underlying relation of an equivalence.

Definition 4. *We define the set neutral terms N and values V by induction:*

$$\begin{aligned} N\ :=\ & x \mid N(i) \mid N.1 \mid N.2 \mid N(V) \mid \\ & \_ =_{\lambda i.N} \_ \mid \mathrm{Equiv}(\lambda i.N) \mid \langle V, \cdots, V\rangle(i) \end{aligned} \tag{49}$$

$$\begin{aligned} V \ := \ & N \mid \lambda i.V \mid (V,V) \mid \lambda x.V \mid \\ & \Sigma(x:V).V \mid \Pi(x:V).V \mid \mathcal{U}^n \end{aligned} \tag{50}$$

We described normal forms as if the identification for Π-types was natural, although we have already explained we were not yet able to provide a proof of this. On the other hand the rule:

$$(A =_{\mathcal{U}} B) \ \equiv \ (A \simeq B)$$

is unnatural and we believe this is unfixable. This explains the unusual neutral terms $\_ =_{\lambda i.N} \_$, $\mathrm{Equiv}(\lambda i.N)$ and $\langle V, \cdots, V\rangle(i)$.

We conjecture that once the problem with Π is fixed, a term in normal form in our system will belong to the the syntactic category V.

Remark 11. *For this result to hold reduction rules should be slightly changed, e.g. we want to reduce $\overrightarrow{\lambda i.\Sigma(x:A).B}$ directly instead of $\overrightarrow{\lambda i.\Sigma(x:A).B}(a,b)$.*

It should be noted that there is no hope for the reciprocal, for at least two reasons. Firstly V does not take the observational rules into account, so that for example with a variable $f : A \to B$, the term $\lambda x.\, f(x) : A \to B$ is in V but can be reduced to f. Secondly if $A : \mathcal{U}$ is a neutral term in which i does not occur, then $\mathrm{Equiv}(\lambda i.A)$ is in V, but can be reduced using the rules for constant paths between types. We guess these are the only obstructions to a reciprocal, but we did not investigate this matter further.

3.2 Toward interpretation

We plan to give an interpretation of our theory in some simpler type theory in order to justify some of its properties, including consistency.

3.2.1 From intensional type theory to our theory

We use an intensional type theory (abreviated ITT) with $\mathcal{U}^n$, Π, Σ and identity types. First we give an easy example of translation: we translate sequent from ITT to our theory. The cases of $\mathcal{U}^n$, Π and Σ are straightforward, indeed our theory has all the rules of ITT by definition. Now we need to treat the case of identity types, i.e. the inductive family:

$$\mathrm{Id} : (A : \mathcal{U}^n) \to A \to A \to \mathcal{U}^n$$

in ITT . We write down its rules.

$$\frac{\Gamma \vdash A : \mathcal{U}^n \qquad \Gamma \vdash a : A \qquad \Gamma \vdash a' : A}{\Gamma \vdash \mathrm{Id}_A(a, a') : \mathcal{U}^n}$$

$$\frac{\Gamma \vdash A : \mathcal{U}^n \qquad \Gamma \vdash a : A}{\Gamma \vdash \mathrm{refl}_a : \mathrm{Id}_A(a, a)}$$

$$\frac{\Gamma \vdash A : \mathcal{U}^n \qquad \Gamma \vdash C : \{a, b : A\} \to \mathrm{Id}_A(a, b) \to \mathcal{U}^n \qquad \Gamma \vdash d : (a : A) \to C(\mathrm{refl}_a)}{\Gamma \vdash J_A(P, d) : \{a, b : A\} \to (p : \mathrm{Id}_A(a, b)) \to C(p)}$$

Together with the computation rule:

$$J(P, d, \mathrm{refl}_a) \;\equiv\; d(a) \tag{51}$$

We interpret $\mathrm{Id}_A(a, b)$ as $a =_A b$, and refl_a as $\widehat{a}$. Now we need to give a suitable interpretation for J.

Lemma 4. *Assume given:*

$$A : \mathcal{U}^n$$

$$C : \{a, b : A\} \to a =_A b \to \mathcal{U}$$

and

$$d : (a : A) \to C(\widehat{a})$$

in our theory. It is possible to build a term:

$$J_A(C, d) : \{a, b : A\} \to (p : a =_A b) \to C(p)$$

such that:

$$J_A(C, d, \widehat{a}) \;\equiv\; d(a)$$

Proof. Assume given $p : a =_A b$. We define $q : a =_A b$ as $\overrightarrow{\lambda i.a =_A p(i)}(\widehat{a})$ and:

$$H : \widehat{a} =_{\lambda i.a =_A q(i)} p$$

as the transpose of $\overrightarrow{\overrightarrow{\lambda i.a =_A p(i)}}(\widehat{a})$.

Then we define $J_A(C, d, p)$ as:

$$\overrightarrow{\lambda i.C(H(i))}(d(a))$$

Indeed $\lambda i.C(H(i)) : C(\widehat{a}) =_{\mathcal{U}} C(p)$.

Now we check the computation rules. Assume p is $\widehat{a}$, then:

$$q \;\equiv\; \widehat{a}$$

$$H \;\equiv\; \widehat{\widehat{a}}$$

$$\lambda i.C(H(i)) \;\equiv\; \widehat{C(\widehat{a})}$$

and finally:

$$J_A(C, d, \widehat{a}) \;\equiv\; d(a)$$

□

Note that the proof of the computation rule makes an essential use of regularity.

So we have a translation from ITT to our theory. We know that this translation validates new principles, most notably univalence. It also validates a lot of definitional equalities. It would interesting to know precisely what are the new principles validated, i.e. have a list H of types in ITT such that:

The translation of a type is inhabited in our theory if and only it is inhabited in ITT + H.

3.2.2 The translation by iterated parametricity

Here we present our main idea for a translation justifying our theory, which acts as a guide for the design of our system. This is quite speculative as we are still in the process of convincing ourselves that such a translation exists.

The idea is to use a translation similar to parametricity, where a type is translated as a pair of types and an equivalence between them rather than a relation. We call such a translation a univalent translation, and we denote it by $[\_]$.

Then one could define a translation $[\![\_]\!]$ from our theory to some target theory T, with $[\_]$ going from T to T. It would look like:

$$\begin{aligned} [\![x : A, \Gamma \vdash A]\!] &= x : A, [\![\Gamma \vdash A]\!] \\ [\![i, \Gamma \vdash A]\!] &= [[\![\Gamma \vdash A]\!]] \end{aligned}$$

Of course a lot of details need to be filled, for example we were not careful at all about the dependencies between types. A conceptual problem is the interpretation of the identity types in the definition of equivalences. Our idea is that a type should come with iterated identity types and Kan compositions, presumably defined by induction on the type. We suspect that this structure on A is in fact equivalent to the family of the iterated univalent translations of A. It is not clear yet how to perform this precisely.

Now we would also need to perform $[\![\_]\!]$ on term. The most interesting cases should be $[\![\lambda i.a]\!]$ and $[\![p(i)]\!]$.

- We hope that identity types are interpreted using $[\_]$, so that if $a : A$ then $[\![\lambda i.a]\!]$ should be of type $[[\![A]\!]]$. But this is the type of a because it is in a context with one more dimension variable i, so $[\![\lambda i.a]\!]$ is very close $[\![a]\!]$. We suspect that $[\![a]\!]$ is a triplet of two endpoints and a path between them, and $[\![\lambda i.a]\!]$ is the path alone.

- The interpretation of $p(i)$ is less clear, because p and $p(i)$ are in the same context. In this case $[\![p]\!]$ is an iterated univalent translation of $[\![A]\!]$, with at least one translation inherited from i and another one from the fact that p is an equality proof. So we should find a diagonalisation of some sort, merging the two univalent translations.

This translation is still very sketchy, but the idea to iterate a univalent translation once for each dimension name has an important role in the design of our theory.

Remark 12. *Note that other potential translations can be contemplated, for example to a type theory with a closed universe, i.e. with an internal principle of induction on the universes.*

Technically we would define identity types by induction on the universe, presumably together with some extra things useful to define equivalences.

3.3 Data types

Our type theory only includes universes, Σ and Π-types, and therefore it is not suitable to formalize actual mathematics. In this section we give tentative definitions for some data types.

Since we will only consider types in the empty context and we admitted regularity, we do not need any rule to compute with equivalences built from these new types. If regularity turns out to be inconsistent, we should probably add it for data types only. We will omit the superscripts indicating the universe levels, so that we do not commit to large or small elimination principles.

This discussion is very preliminary and we do not take into account how the new types would compute, we only care that they obey the correct equations.

3.3.1 Unit type

We add a singleton type $\top$ with η-rule.

$$\frac{}{\Gamma \vdash \top : \mathcal{U}}$$

$$\frac{}{\Gamma \vdash * : \top}$$

For any $s : \top$ we add the computation rule:

$$s \equiv *$$

Next we need to add rules for the identity types in $\top$.

$$s =_\top t \equiv \top \tag{52}$$

This rule is enough, it implies that $*(i)$ and $\lambda i.s$ with $s : \top$ are equal to $*$.

Remark 13. *We could also add a singleton type without η-rule. In this case we would add:*

$$\frac{\Gamma \vdash P : \top \to \mathcal{U} \qquad \Gamma \vdash c : P(*)}{\Gamma \vdash \top - \mathbf{rec}(P, c) : (x : \top) \to P(x)}$$

and we would replace the η-rule by:

$$\top - \mathbf{rec}(P, c, *) \ \equiv \ c$$

Then the rules for identity types would be:

$$\begin{aligned} * =_{\top} * \ &\equiv \ \top \\ \widehat{*} \ &\equiv \ * \\ *(i) \ &\equiv \ * \end{aligned}$$

Note that here we need to compute with $\widehat{*}$ directly, without waiting for an eliminator $\top - \mathbf{rec}$ to be applied. This might be a sign that things goes a bit differently without η-rule, a question not investigated yet.

3.3.2 Empty type

We add $\bot$ in each universe.

$$\frac{}{\Gamma \vdash \bot : \mathcal{U}}$$

$$\frac{\Gamma \vdash A : \mathcal{U}}{\Gamma \vdash \bot - \mathbf{rec}(A) : \bot \to A}$$

For the identity types we add:

$$s =_{\bot} t \ \equiv \ \bot \tag{53}$$

There is no other reduction rules because there is no canonical inhabitant in $\bot$. We could also try with $(s =_{\bot} t) \equiv \top$, although it would lead to difficulties when computing with $*(i) : \bot$. It might even be possible to not add any rule for identity types.

Remark 14. *Also note that we do not add rules to deal with $\bot - \mathbf{rec}$ constructing path, because it is an eliminator. But it is possible to find a well-typed one:*

$$\lambda i.\bot{-}\mathbf{rec}(A, c) \ \equiv \ \bot{-}\mathbf{rec}(\bot{-}\mathbf{rec}(A[i/0], c[i/0]) =_{\lambda i.A} \bot{-}\mathbf{rec}(A[i/1], c[i/1]), \lambda i.c)$$

This rule would break confluence.

In fact it seems always possible to compute with λi applied to an eliminator, even if these rules break confluence. Indeed this is also the case for Π and Σ-types, and for the new types we are about to present. It would be interesting to understand this phenomena, which might have some applications.

3.3.3 Booleans

We add a type $\mathbb{B}$ of booleans.

$$\frac{}{\Gamma \vdash \mathbb{B} : \mathcal{U}}$$

$$\frac{}{\Gamma \vdash \mathbf{true} : \mathbb{B}}$$

$$\frac{}{\Gamma \vdash \mathbf{false} : \mathbb{B}}$$

$$\frac{\Gamma \vdash P : \mathbb{B} \to \mathcal{U} \qquad \Gamma \vdash c_0 : P(\mathbf{true}) \qquad \Gamma \vdash c_1 : P(\mathbf{false})}{\Gamma \vdash \mathbb{B} - \mathbf{rec}(P, c_0, c_1) : (b : \mathbb{B}) \to P(b)}$$

We add the usual conversion rules:

$$\mathbb{B} - \mathbf{rec}(P, c_0, c_1, \mathbf{true}) \equiv c_0 \tag{54}$$
$$\mathbb{B} - \mathbf{rec}(P, c_0, c_1, \mathbf{false}) \equiv c_1 \tag{55}$$

Now we need to treat the identity types. We add the rules:

$$\mathbf{true} =_{\mathbb{B}} \mathbf{true} \equiv \top \tag{56}$$
$$\mathbf{true} =_{\mathbb{B}} \mathbf{false} \equiv \bot \tag{57}$$
$$\mathbf{false} =_{\mathbb{B}} \mathbf{true} \equiv \bot \tag{58}$$
$$\mathbf{false} =_{\mathbb{B}} \mathbf{false} \equiv \top \tag{59}$$

Note that the rule for $\widehat{\mathbf{true}} \equiv *$ and $\widehat{\mathbf{false}} \equiv *$ are already consequences of the η-rule for $\top$. Similarly for $* : \mathbf{true} =_{\mathbb{B}} \mathbf{true}$ we have:

$$*(i) \equiv \widehat{\mathbf{true}}(i) \equiv \mathbf{true}$$

So we do not need to add any more rule to $\mathbb{B}$ because we already have all the equations we want. If we were concerned with normal forms, we would add reductions.

Remark 15. *It might be problematic that:*

$$(\mathbf{true} =_{\mathbb{B}} \mathbf{true}) \equiv (\mathbf{false} =_{\mathbb{B}} \mathbf{false})$$

For example one should be careful about $*(i) \equiv \mathbf{false}$ *and* $*(i) \equiv \mathbf{true}$ *(or even* $*(i) \equiv *$ *in the unit type) depending on the type of* $*$*. A possible alternative approach would be to generate inductively a family:*

$$\mathbf{Eq}_{\mathbb{B}} : \mathbb{B} \to \mathbb{B} \to \mathcal{U}$$

with

$$*_{\mathbf{true}} : \mathbf{Eq}_{\mathbb{B}}(\mathbf{true}, \mathbf{true})$$

and

$$*_{\mathbf{false}} : \mathbf{Eq}_{\mathbb{B}}(\mathbf{false}, \mathbf{false})$$

and then add:

$$s =_{\mathbb{B}} t \quad \equiv \quad \mathbf{Eq}_{\mathbb{B}}(s, t)$$

In this case we would have the reduction rules:

$$\begin{aligned}
\widehat{\mathbf{true}} &\equiv *_{\mathbf{true}} \\
\widehat{\mathbf{false}} &\equiv *_{\mathbf{false}} \\
*_{\mathbf{true}}(i) &\equiv \mathbf{true} \\
*_{\mathbf{false}}(i) &\equiv \mathbf{false}
\end{aligned}$$

But this would require inductive families in our theory, taking us beyond the scope of this discussion.

3.3.4 Natural numbers

Now we add our first recursive type, the type $\mathbb{N}$ of natural numbers.

$$\frac{}{\Gamma \vdash \mathbb{N} : \mathcal{U}}$$

$$\frac{}{\Gamma \vdash \mathbf{0} : \mathbb{N}}$$

$$\frac{\Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathbf{s}(n) : \mathbb{N}}$$

$$\frac{\Gamma \vdash P : \mathbb{N} \to \mathcal{U} \qquad \Gamma \vdash c_0 : P(\mathbf{0}) \qquad \Gamma \vdash c_s : \{n : \mathbb{N}\} \to P(n) \to P(\mathbf{s}(n))}{\Gamma \vdash \mathbb{N} - \mathbf{rec}(P, c_0, c_s) : (n : \mathbb{N}) \to P(n)}$$

We add the usual computation rules:

$$\begin{aligned}
\mathbb{N} - \mathbf{rec}(P, c_0, c_s, \mathbf{0}) &\equiv c_0 && (60) \\
\mathbb{N} - \mathbf{rec}(P, c_0, c_s, \mathbf{s}(n)) &\equiv c_s(\mathbb{N} - \mathbf{rec}(P, c_0, c_s, n)) && (61)
\end{aligned}$$

And finally the rules for identity types:

$$\begin{aligned}
\mathbf{0} =_{\mathbb{N}} \mathbf{0} &\equiv \top && (62) \\
\mathbf{0} =_{\mathbb{N}} \mathbf{s}(n) &\equiv \bot && (63) \\
\mathbf{s}(m) =_{\mathbb{N}} \mathbf{0} &\equiv \bot && (64) \\
\mathbf{s}(m) =_{\mathbb{N}} \mathbf{s}(n) &\equiv m =_{\mathbb{N}} n && (65)
\end{aligned}$$

Then we want to add rules to compute for $\mathbf{s}$ and paths (as before the rules for $\mathbf{0}$ are consequences of the η-rule for $\top$). The first try would be:

$$\lambda i.\mathbf{s}(n) \equiv \lambda i.n \tag{66}$$

$$p(i) \equiv \mathbf{s}(p(i)) \tag{67}$$

where the second equation should only apply when p is built in $m =_{\mathbb{N}} n$, and then seen as an element of $\mathbf{s}(m) =_{\mathbb{N}} \mathbf{s}(n)$. These equations look very unsafe, next remark gives a more reasonable approach.

Remark 16. *In order to write sensible computation rules, we really ought to define:*

$$\mathbf{Eq}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N} \to \mathcal{U}$$

inductively with

$$*_{\mathbf{0}} : \mathbf{Eq}_{\mathbb{N}}(\mathbf{0}, \mathbf{0})$$

and

$$*_{\mathbf{s}} : \mathbf{Eq}_{\mathbb{N}}(m, n) \to \mathbf{Eq}_{\mathbb{N}}(\mathbf{s}(m), \mathbf{s}(n))$$

Then we would be able to add:

$$m =_{\mathbb{N}} n \equiv \mathbf{Eq}_{\mathbb{N}}(m, n)$$

and the rules:

$$\begin{aligned} \widehat{\mathbf{0}} &\equiv *_{\mathbf{0}} \\ \lambda i.\mathbf{s}(n) &\equiv *_{\mathbf{s}}(\lambda i.n) \\ *_{\mathbf{0}}(i) &\equiv \mathbf{0} \\ *_{\mathbf{s}}(p)(i) &\equiv \mathbf{s}(p(i)) \end{aligned}$$

which look more reasonable.

To summarize this discussion about data types, it seems that a valid strategy would be to define their intended identity types as an inductive family, and then add some natural rules for λi and application to i. It seems that in general (i.e. without η-rule) we need to reduce $\lambda i.t$ directly, without waiting for it to be eliminated.

3.4 More general inductive types

Now we will consider some polymorphic inductive types. Using insights from our discussion on datatypes, we will proceed using an inductively defined family $\mathbf{Eq}$ to compute with identity types. Since we do not define inductive families in our system this is really an informal discussion, a first step toward inductive types in our theory.

In this whole section we omit arguments when they are endpoints of paths.

3.4.1 Disjoint Unions

We add the rule for disjoint union.

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash B : \mathcal{U}}{\Gamma \vdash A + B : \mathcal{U}}$$

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash \mathbf{inc}_1(a) : A + B}$$

$$\frac{\Gamma \vdash b : B}{\Gamma \vdash \mathbf{inc}_2(b) : A + B}$$

$$\frac{\Gamma \vdash P : A + B \to \mathcal{U} \qquad \Gamma \vdash f : (x : A) \to P(\mathbf{inc}_1(x)) \qquad \Gamma \vdash g : (y : B) \to P(\mathbf{inc}_2(y))}{\Gamma \vdash + - \mathbf{rec}(P, f, g) : (x : A + B) \to P(x)}$$

Note that we omitted type subscripts on $\mathbf{inc}_1$ and $\mathbf{inc}_2$, as they are easy to infer from the context. Now we add the computation rules:

$$+ - \mathbf{rec}(P, f, g, \mathbf{inc}_1(a)) \equiv f(a) \tag{68}$$
$$+ - \mathbf{rec}(P, f, g, \mathbf{inc}_2(b)) \equiv g(b) \tag{69}$$

Then for identity types, we define inductively a family:

$$\mathbf{Eq}_{\lambda i.A+B} : (A + B)[i/0] \to (A + B)[i/1] \to \mathcal{U}$$

with

$$*_1 : a =_{\lambda i.A} a' \to \mathbf{Eq}_{\lambda i.A+B}(\mathbf{inc}_1(a), \mathbf{inc}_1(a'))$$

and

$$*_2 : b =_{\lambda i.B} b' \to \mathbf{Eq}_{\lambda i.A+B}(\mathbf{inc}_2(b), \mathbf{inc}_2(b'))$$

Then we add:

$$s =_{\lambda i.A+B} t \equiv \mathbf{Eq}_{\lambda i.A+B}(s, t) \tag{70}$$
$$\lambda i.\mathbf{inc}_1(a) \equiv *_1(\lambda i.a) \tag{71}$$
$$\lambda i.\mathbf{inc}_2(b) \equiv *_2(\lambda i.b) \tag{72}$$
$$*_1(p)(i) \equiv \mathbf{inc}_1(p(i)) \tag{73}$$
$$*_2(q)(i) \equiv \mathbf{inc}_2(q(i)) \tag{74}$$

Finally we need to explain how to compute with $\lambda i.\, A + B$ seen as an equivalence:

$$\overrightarrow{\lambda i.A + B}(\mathbf{inc}_1(a)) \equiv \mathbf{inc}_1(\overrightarrow{\lambda i.A}(a)) \tag{75}$$
$$\overrightarrow{\lambda i.A + B}(\mathbf{inc}_2(b)) \equiv \mathbf{inc}_2(\overrightarrow{\lambda i.B}(b)) \tag{76}$$
$$\overset{\Longrightarrow}{\lambda i.A + B}(\mathbf{inc}_1(a)) \equiv *_1(\overset{\Longrightarrow}{\lambda i.A}(a)) \tag{77}$$
$$\overset{\Longrightarrow}{\lambda i.A + B}(\mathbf{inc}_2(b)) \equiv *_2(\overset{\Longrightarrow}{\lambda i.B}(b)) \tag{78}$$

3.4.2 Lists

We add the rules for lists.

$$\frac{\Gamma \vdash A : \mathcal{U}}{\Gamma \vdash \mathbf{List}_A : \mathcal{U}}$$

$$\frac{}{\Gamma \vdash \mathbf{nil}_A : \mathbf{List}_A}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash l : \mathbf{List}_A}{\Gamma \vdash a \frown l : \mathbf{List}_A}$$

$$\frac{\Gamma \vdash P : \mathbf{List}_A \to \mathcal{U} \qquad \Gamma \vdash c : P(\mathbf{nil}_A) \qquad \Gamma \vdash f : (a : A) \to (l : \mathbf{List}_A) \to P(l) \to P(a \frown l)}{\Gamma \vdash \mathbf{List}_A - \mathbf{rec}(P, c, f) : (x : \mathbf{List}_A) \to P(x)}$$

With:

$$\begin{aligned} \mathbf{List}_A - \mathbf{rec}(P, c, f, \mathbf{nil}_A) &\equiv c & (79)\\ \mathbf{List}_A - \mathbf{rec}(P, c, f, a \frown l) &\equiv f(a, l, \mathbf{List}_A - \mathbf{rec}(P, f, l)) & (80) \end{aligned}$$

Now we define inductively:

$$\mathbf{Eq}_{\lambda i.\mathbf{List}_A} : \mathbf{List}_{A[i/0]} \to \mathbf{List}_{A[i/1]} \to \mathcal{U}$$

with

$$*_{\mathbf{nil}} : \mathbf{Eq}_{\lambda i.\mathbf{List}_A}(\mathbf{nil}_{A[i/0]}, \mathbf{nil}_{A[i/1]})$$

and

$$*_{\frown} : a =_{\lambda i.A} a' \to \mathbf{Eq}_{\lambda i.\mathbf{List}_A}(l, l') \to \mathbf{Eq}_{\lambda i.\mathbf{List}_1}(a \frown l, a' \frown l')$$

Then we add conversion rules for identity types:

$$\begin{aligned} l =_{\lambda i.\mathbf{List}_A} l' &\equiv \mathbf{Eq}_{\lambda i.\mathbf{List}_A}(l, l') & (81)\\ \lambda i.\mathbf{nil}_A &\equiv *_{\mathbf{nil}} & (82)\\ \lambda i.\, a \frown l &\equiv *_{\frown}(\lambda i.a, \lambda i.l) & (83)\\ *_{\mathbf{nil}}(j) &\equiv \mathbf{nil}_{A[i/j]} & (84)\\ *_{\frown}(p, q)(j) &\equiv p(j) \frown q(j) & (85) \end{aligned}$$

And now we explain how to compute with $\lambda i.\mathbf{List}_A$ seen as an equivalence:

$$\begin{aligned} \overrightarrow{\lambda i.\mathbf{List}_A}(\mathbf{nil}_{A[i/0]}) &\equiv \mathbf{nil}_{A[i/1]} & (86)\\ \overrightarrow{\lambda i.\mathbf{List}_A}(a \frown l) &\equiv \overrightarrow{\lambda i.A}(a) \frown \overrightarrow{\lambda i.\mathbf{List}_A}(l) & (87)\\ \overset{\Longrightarrow}{\lambda i.\mathbf{List}_A}(\mathbf{nil}_{A[i/0]}) &\equiv *_{\mathbf{nil}} & (88)\\ \overset{\Longrightarrow}{\lambda i.\mathbf{List}_A}(a \frown l) &\equiv *_{\frown}(\overset{\Longrightarrow}{\lambda i.A}(a), \overset{\Longrightarrow}{\lambda i.\mathbf{List}_A}(l)) & (89) \end{aligned}$$

3.4.3 W-types

We add the rules for W-types, which represent well-founded trees.

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash B : A \to \mathcal{U}}{\Gamma \vdash \mathbf{W}(A, B) : \mathcal{U}}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash f : B(a) \to \mathbf{W}(A, B)}{\Gamma \vdash \mathbf{cons}(a, f) : \mathbf{W}(A, B)}$$

We omit the elimination principle.

Now we define inductively:

$$\mathbf{Eq}_{\lambda i.\mathbf{W}(A,B)} : \mathbf{W}(A, B)[i/0] \to \mathbf{W}(A, B)[i/1] \to \mathcal{U}$$

with

$$\begin{gathered} *_{\mathbf{cons}} : \{a : A[i/0]\} \to \{a' : A[i/1]\} \to (p : a =_{\lambda i.A} a') \\ \to \big(\{b : B[i/0]\} \to \{b' : B[i/1]\} \to b =_{\lambda i.B(p(i))} b' \to \mathbf{Eq}_{\lambda i.\mathbf{W}(A,B)}(f(b), f'(b'))\big) \\ \to \mathbf{Eq}_{\lambda i.\mathbf{W}(A,B)}(\mathbf{cons}(a, f), \mathbf{cons}(a', f')) \end{gathered}$$

Then we add:

$$s =_{\lambda i.\mathbf{W}(A,B)} t \equiv \mathbf{Eq}_{\lambda i.\mathbf{W}(A,B)}(s, t) \tag{90}$$

$$\lambda i.\mathbf{cons}(a, f) \equiv *_{\mathbf{cons}}(\lambda i.a, \lambda i.f) \tag{91}$$

$$*_{\mathbf{cons}}(p, q)(i) \equiv \mathbf{cons}(p(i), q(i)) \tag{92}$$

We also need to explain how to compute with $\lambda i.\mathbf{W}(A, B)$ seen as an equivalence:

$$\overrightarrow{\lambda i.\mathbf{W}(A, B)}(\mathbf{cons}(a, f)) \equiv \mathbf{cons}\big(\overrightarrow{\lambda i.A}(a), \overrightarrow{\lambda i.B(\overset{\Longrightarrow}{\lambda i.A}(a)(i)) \to \mathbf{W}(A, B)}(f)\big)$$

$$\overset{\Longrightarrow}{\lambda i.\mathbf{W}(A, B)}(\mathbf{cons}(a, f)) \equiv \mathbf{cons}\big(\overset{\Longrightarrow}{\lambda i.A}(a), \overset{\Longrightarrow}{\lambda i.B(\overset{\Longrightarrow}{\lambda i.A}(a)(i)) \to \mathbf{W}(A, B)}(f)\big)$$

3.4.4 The circle is not so easy

The circle $\mathbb{S}^1$ is the only higher inductive type we discuss. We will explain why the straightforward approach to compute with it fails.

The type $\mathbb{S}^1$ is inductively generated by an element $\mathbf{base} : \mathbb{S}^1$, and a path $\mathbf{loop} : \mathbf{base} =_{\mathbb{S}^1} \mathbf{base}$. This means that the type $\mathbf{base} =_{\mathbb{S}^1} \mathbf{base}$ is freely generated by $\mathbf{loop}$, so that it is equivalent to $\mathbb{Z}$. We admit that $\mathbb{Z}$ has been defined in our theory, this is easy using $\mathbb{N}$ and disjoint unions.

We give the precise rules for the circle:

$$\frac{}{\Gamma \vdash \mathbb{S}^1 : \mathcal{U}}$$

$$\frac{}{\Gamma \vdash \mathbf{base} : \mathbb{S}^1}$$

$$\frac{}{\Gamma \vdash \mathbf{loop} : \mathbf{base} =_{\mathbb{S}^1} \mathbf{base}}$$

$$\frac{\Gamma \vdash P : \mathbb{S}^1 \to \mathcal{U} \qquad \Gamma \vdash b : P(\mathbf{base}) \qquad \Gamma \vdash l : b =_{\lambda i. P(\mathbf{loop}(i))} b}{\Gamma \vdash \mathbb{S}^1 - \mathbf{rec}(P, b, l) : (x : \mathbb{S}^1) \to P(x)}$$

Together with:

$$\mathbb{S}^1 - \mathbf{rec}(P, b, l, \mathbf{base}) \equiv b \tag{93}$$

$$\mathbb{S}^1 - \mathbf{rec}(P, b, l, \mathbf{loop}(i)) \equiv l(i) \tag{94}$$

So we just defined the circle by its universal principle. At this point we want to make its identity types compute, as we did for other types. It is tempting to add a rule like:

$$\mathbf{base} =_{\mathbb{S}^1} \mathbf{base} \equiv \mathbb{Z}$$

but it is unclear how to do so. We explain why the naive approach fails.

We denote by **Succ** the equivalence in $\mathbb{Z} \simeq \mathbb{Z}$ with underlying function

$$\lambda(x : \mathbb{Z}).\, x + 1$$

and **Pred** its inverse. We use a square $\Psi : \mathbf{Succ} =_{\lambda i. \mathbf{Pred}(i) =_{\mathcal{U}} \mathbf{Pred}(i)} \mathbf{Succ}$, certainly definable using the definition of $\mathbb{Z}$. Then we define:

$$\mathbf{Eq}_{\mathbb{S}^1} : \mathbb{S}^1 \to \mathbb{S}^1 \to \mathcal{U}$$

by double induction on the circle:

$$\mathbf{Eq}_{\mathbb{S}^1}(\mathbf{base}, \mathbf{base}) \equiv \mathbb{Z} \tag{95}$$

$$\mathbf{Eq}_{\mathbb{S}^1}(\mathbf{base}, \mathbf{loop}(j)) \equiv \mathbf{Succ}(j) \tag{96}$$

$$\mathbf{Eq}_{\mathbb{S}^1}(\mathbf{loop}(i), \mathbf{base}) \equiv \mathbf{Pred}(i) \tag{97}$$

$$\mathbf{Eq}_{\mathbb{S}^1}(\mathbf{loop}(i), \mathbf{loop}(j)) \equiv \Psi(i, j) \tag{98}$$

Then one could add:

$$s =_{\mathbb{S}^1} t \equiv \mathbf{Eq}_{\mathbb{S}^1}(s, t)$$

Together with the computations:

$$\widehat{\mathbf{base}} \equiv 0 \tag{99}$$

$$\mathbf{loop} \equiv 1 \tag{100}$$

And then I guess $m(i)$ for m a numeral in $\mathbb{Z}$ should be considered a normal form.

But this has some serious drawbacks:

- The application of $\mathbb{S}^1 - \mathbf{rec}$ to say $2(i)$ does not compute.
- Composition of paths does not compute, for example $\overrightarrow{\lambda i.5(i) =_{\mathbb{S}^1} 7(i)}(2)$ seems to be a normal form, so it is a non-canonical element in $\mathbb{Z}$.

In the end these computation rules for $\mathbb{S}^1$ are not satisfying at all. Note that the option to generate $\mathbf{Eq}_{\mathbb{S}^1}$ inductively is not so easily usable because if we just define:

$$\mathbf{Eq}_{\mathbb{S}^1} : \mathbb{S}^1 \to \mathbb{S}^1 \to \mathcal{U}$$

using

$$*_{\mathbf{base}} : \mathbb{Z} \to \mathbf{Eq}_{\mathbb{S}^1}(\mathbf{base}, \mathbf{base})$$

then $\mathbf{Eq}_{\mathbb{S}^1}(\mathbf{base}, \mathbf{base})$ contains a lot more than $\mathbb{Z}$, indeed all transport of elements in $\mathbb{Z}$ along **loop** will be added as well. The natural solution seems to add a constructor $*_{\mathbf{loop}}$, then $\mathbf{Eq}_{\mathbb{S}^1}$ is a higher inductive family. We did not investigate this further.

Overall this discussion shows that the situation for higher inductive types is less clear that for usual inductive types.

4 Conclusion

We give a list of interesting questions left unsolved.

4.1 Current goals

This work is unfinished, and the following points should be worked on immediately.

- The problem with computations for Π-types needs to be solved.
- We need to provide an interpretation, at least to justify consistency. This would also be useful when adding new constructors to our theory. Note that regularity might break consistency, in which case it will be restrained.
- There are a lot of possible choices of definition for equivalence. We should think about them, and choose one which allows natural computations. For example our computation rules for $\lambda i.\, \Pi(x : A).B$ are not very straightforward, can this can be improved?

4.2 Further work

Now we give a list of more ambitious questions:

- Does our theory enjoy canonicity? How to prove it? More generally does it compute well? Decidability of type checking is far from obvious because a lot of types are equal in our theory.

- How to add more general inductive types in our theory? Our discussion on usual inductive types should be developed further. What about inductive families? They seem to be necessary for a smooth development of data types. What about higher inductive types?

- Assume that instead of:

$$(A =_{\mathcal{U}} B) \;\equiv\; (A \simeq B)$$

  we use:

$$(A =_{\mathcal{U}} B) \;\equiv\; (A \to B \to \mathcal{U})$$

  The notation " $=$ " is misleading here, we should probably write e.g. " $\sim$ " instead. Nevertheless it seems reasonable to expect a theory internalizing parametricity. Is this true? This can probably be generalized to other types than $A \simeq B$ or $A \to B \to \mathcal{U}$. Which one? What is the meaning of a theory obtained this way?

References

[1] Thorsten Altenkirch and Ambrus Kaposi. Towards a cubical type theory without an interval. *Leibniz International Proceedings in Informatics*, 2017.

[2] Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. Observational equality, now! *PLPV*, 7:57–68, 2007.

[3] Carlo Angiuli, Kuen-Bang Favonia Hou, and Robert Harper. Cartesian cubical computational type theory: Constructive reasoning with paths and equalities. *Computer Science Logic 2018*, 2018.

[4] Jean-Philippe Bernardy, Thierry Coquand, and Guilhem Moulin. A presheaf model of parametric type theory. *Electronic Notes in Theoretical Computer Science*, 319:67–82, 2015.

[5] Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. Parametricity and dependent types. In *ACM Sigplan Notices*, volume 45, pages 345–356. ACM, 2010.

[6] Jean-Philippe Bernardy and Guilhem Moulin. Type-theory in color. In *ICFP*, pages 61–72. Citeseer, 2013.

[7] Marc Bezem, Thierry Coquand, and Simon Huber. A model of type theory in cubical sets. In *19th International Conference on Types for Proofs and Programs (TYPES 2013)*, volume 26, pages 107–128, 2014.

[8] Marc Bezem, Thierry Coquand, and Simon Huber. The univalence axiom in cubical sets. *Journal of Automated Reasoning*, pages 1–13, 2018.

[9] Guillaume Brunerie. On the homotopy groups of spheres in homotopy type theory. *arXiv preprint arXiv:1606.05916*, 2016.

[10] Evan Cavallo and Robert Harper. Parametric cubical type theory. *arXiv preprint arXiv:1901.00489*, 2019.

[11] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical type theory: a constructive interpretation of the univalence axiom. *arXiv preprint arXiv:1611.02108*, 2016.

[12] Thierry Coquand, Simon Huber, and Anders Mörtberg. On higher inductive types in cubical type theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 255–264. ACM, 2018.

[13] Peter Dybjer. Inductive families. *Formal aspects of computing*, 6(4):440–465, 1994.

[14] Simon Huber. Canonicity for cubical type theory. *Journal of Automated Reasoning*, pages 1–38, 2018.

[15] Chris Kapulkin and Peter LeFanu Lumsdaine. The simplicial model of univalent foundations (after Voevodsky). *arXiv preprint arXiv:1211.2851*, 2012.

[16] Daniel Licata and Guillaume Brunerie. $\pi_n(S^n)$ in homotopy type theory. In *International Conference on Certified Programs and Proofs*, pages 1–16. Springer, 2013.

[17] Daniel Licata and Michael Shulman. Calculating the fundamental group of the circle in homotopy type theory. In *2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 223–232. IEEE, 2013.

[18] Jacob Lurie. *Higher Topos Theory (AM-170)*, volume 189. Princeton University Press, 2009.

[19] Per Martin-Löf. An intuitionistic theory of types: Predicative part. In *Studies in Logic and the Foundations of Mathematics*, volume 80, pages 73–118. Elsevier, 1975.

[20] Per Martin-Löf. Constructive mathematics and computer programming. In *Studies in Logic and the Foundations of Mathematics*, volume 104, pages 153–175. Elsevier, 1982.

[21] Per Martin-Löf and Giovanni Sambin. *Intuitionistic type theory.* Bibliopolis Naples, 1984.

[22] John Reynolds. Types, abstraction and parametric polymorphism. In *Information Processing 83, Proceedings of the IFIP 9th World Computer Congres*, pages 513–523, 1983.

[23] Michael Shulman. All $(\infty, 1)$-toposes have strict univalent universes. *arXiv preprint arXiv:1904.07004*, 2019.

[24] Jonathan Sterling, Carlo Angiuli, and Daniel Gratzer. Cubical syntax for reflection-free extensional equality. *arXiv preprint arXiv:1904.08562*, 2019.

[25] Andrew Swan. Separating path and identity types in presheaf models of univalent type theory. *arXiv preprint arXiv:1808.00920*, 2018.

[26] Nicolas Tabareau, Éric Tanter, and Matthieu Sozeau. Equivalences for free: Univalent parametricity for effective transport. *Proceedings of the ACM on Programming Languages*, 2(ICFP):92, 2018.

[27] The univalent foundations program. *Homotopy type theory: univalent foundations of mathematics.* Institute for Advanced Study, 2013. `https://homotopytypetheory.org/book`.

[28] Benno Van Den Berg and Richard Garner. Types are weak ω-groupoids. *Proceedings of the London Mathematical Society*, 102(2):370–394, 2011.

[29] Philip Wadler. Theorems for free! In *FPCA*, volume 89, pages 347–359. Citeseer, 1989.