Higher Inductive types

Egbert Rijke Bas Spitters

Radboud University Nijmegen

May 7th, 2013

Introduction – Violating UIP

Recall that in Martin-Löf type theory, every type A has an associated identity type $=_A : A \to A \to \mathcal{U}$.
We do not assume UIP, but can there exist types for which UIP fails?
The answer is yes:

- UIP fails in Hoffmann and Streicher's groupoid model
- UIP fails in Voevodsky's Kan simplicial sets model

When we want types violating UIP *inside the type theory*, we have to use higher inductive types. (and UA).

Introduction – The first HITs

Higher inductive types were conceived by Bauer, Lumsdaine, Shulman and Warren at the Oberwolfach meeting in 2011. The first examples of higher inductive types include:

- The interval
- The circle
- Squash types

It was shown that:

- Having the interval implies function extensionality.
- The fundamental group of the circle is $\mathbb{Z}$.

Introduction – Some current uses of HITs

At the year of univalent foundations at the IAS, higher inductive types have been used to implement:

- higher truncations
- set quotients
- homotopy colimits (e.g. pushouts)
- various other constructions familiar from homotopy theory (e.g. suspensions, joins, smash product, wedge product, etc...)
- the cumulative hierarchy for sets
- the Cauchy real numbers.

Recalling some basic notions

The homotopical interpretation of type theory by Steve Awodey is that we think of:

- types as spaces
- dependent types as fibrations (continuous families of types)
- identity types as path spaces

For example, using identity types one can attempt to describe the identity relation on $A \to B$ by:

$$f \sim g :\equiv \prod_{x:A} f(x) =_B g(x)$$

The function extensionality axiom asserts that the canonical function $(f =_{A \to B} g) \to (f \sim g)$ is an equivalence.

Equivalences

Recall that an isomorphism between sets is a function $f : A \to B$ such that

$$\mathsf{isIso}(f) :\equiv \sum_{g:B \to A} (g \circ f \sim \mathsf{id}) \times (f \circ g \sim \mathsf{id})$$

When doing proof-relevant mathematics, this notion is not so well-behaved

Instead, we want a stronger notion of $\mathsf{isequiv}(f)$ which satisfies:

- $\mathsf{isequiv}(f) \leftrightarrow \mathsf{isIso}(f)$
- For any $e, e' : \mathsf{isequiv}(f)$ we have $e = e'$.

If A and B are equivalent, we write $A \simeq B$.

For instance, this would allow us to show that equivalences are identical precisely when they are identical as functions.

Some possible implementations of the notion of equivalence

To implement such a notion of equivalence, one needs function extensionality.
Among the possible definitions are:

$$\left(\sum_{(g:B\to A)}(g\circ f\sim \mathrm{id})\right)\times\left(\sum_{(h:B\to A)}(f\circ h\sim \mathrm{id})\right)$$

or

$$\prod_{b:B}\mathrm{isContr}\left(\sum_{(a:A)}(f(a)=_B b)\right)$$

The hierarchy of complexity

Definition

We say that a type A is contractible if there is an element of type

$$\mathsf{isContr}(A) :\equiv \sum_{(x:A)} \prod_{(y:A)} x =_A y$$

Contractible types are said to be of level -2.

Definition

We say that a type A is a mere proposition if there is an element of type

$$\mathsf{isProp}(A) :\equiv \prod_{x,y:A} \mathsf{isContr}(x =_A y)$$

Mere propositions are said to be of level -1.

The hierarchy of complexity

Definition

We say that a type A is a set if there is an element of type

$$\mathrm{isSet}(A) :\equiv \prod_{x,y:A} \mathrm{isProp}(x =_A y)$$

Sets are said to be of level 0.

In general we define:

Definition

Let A be a type. We define

$$\begin{aligned} \text{is-}(-2)\text{-type}(A) &:\equiv \mathrm{isContr}(A) \\ \text{is-}(n+1)\text{-type}(A) &:\equiv \prod_{x,y:A} \text{is-}n\text{-type}(x =_A y) \end{aligned}$$

The classes of n-types are closed under

- dependent products
- dependent sums
- idenity types
- W-types, when $n \geq -1$
- equivalences

Thus, besides 'propositions as types' we also get propositions as n-types for every $n \geq -2$. Often, we will stick to 'propositions as types', but some mathematical concepts (e.g. the axiom of choice) are better interpreted using 'propositions as (-1)-types'.

With propositions as (-1)-types, we have the following logical connectives:

$$\begin{aligned}
\top &:\equiv \mathbf{1} \\
\bot &:\equiv \mathbf{0} \\
P \wedge Q &:\equiv P \times Q \\
P \Rightarrow Q &:\equiv P \to Q \\
P \Leftrightarrow Q &:\equiv P = Q \\
\neg P &:\equiv P \to \mathbf{0} \\
P \vee Q &:\equiv \|P + Q\| \\
\forall (x : A).\, P(x) &:\equiv \prod_{x:A} P(x) \\
\exists (x : A).\, P(x) &:\equiv \Big\|\sum_{x:A} P(x)\Big\|
\end{aligned}$$

The identity type of the universe

The univalence axiom describes the identity type of the universe $\mathcal{U}$. Recall that there is a canonical function

$$(A =_U B) \to (A \simeq B)$$

The univalence axiom is the assertion that this function is an equivalence.

- The univalence axiom formalizes the informal practice of substituting a structure for an isomorphic one.
- It implies function extensionality
- It is used to reason about higher inductive types

Martín Escardo has shown that univalence is incompatible with the following naive form of the law of excluded middle

$$\prod_{A:\mathcal{U}} A + \neg A.$$

This is an instance where we would prefer a formulation using propositions as (-1)-types. The proper formulation of LEM is:

$$\prod_{A:\mathrm{Prop}} A + \neg A.$$

This holds in the Kan simplicial set model, and hence is compatible with univalence.

Recall that ordinary inductive types are introduced with

1. basic constructors
2. from which we derive an induction principle.

The induction principle is formulated dependently:

1. it tells us under what condition there exists a term of type $\prod_{(x:W)} P(X)$ given a dependent type P over the inductively defined type W.
2. the dependency of the induction principle ensures the uniqueness part of the universal.

With higher inductive types, we allow paths among the basic constructors. For example:

- The interval I has basic constructors

$$0_I, 1_I : I \qquad \text{and} \qquad \mathsf{seg} : 0_I =_I 1_I.$$

- The circle $\mathbb{S}^1$ has basic constructors

$$\mathsf{base} : \mathbb{S}^1 \qquad \text{and} \qquad \mathsf{loop} : \mathsf{base} =_{\mathbb{S}^1} \mathsf{base}.$$

With paths among the basic constructors, the induction principle becomes more complicated.
The induction principle describes a condition under which we can prove a property $P(x)$ for all x in the inductively defined type.

Recalling some basic properties:

Lemma

Suppose $P : A \to \mathcal{U}$ is a family of types, let $p : x =_A y$ and let $u : P(x)$. Then there is a term $p_*(u) : P(y)$, called the transportation of u along p.

Lemma

Suppose $f : \prod_{(x:A)} P(x)$ is a dependent function, and let $p : x =_A y$. Then there is a path $f(p) : p_*(f(x)) =_{P(y)} f(y)$.

In the case of the interval, we see that in order for a function $f : \prod_{(x:I)} P(x)$ to exist, we must have

$$\begin{aligned} f(0_I) &: P(0_I) \\ f(1_I) &: P(1_I) \\ f(\mathsf{seg}) &: \mathsf{seg}_*(f(0_I)) =_{P(1_I)} f(1_I) \end{aligned}$$

Induction with the interval

The induction principle for the interval is that for every $P : I \to \mathcal{U}$, if there are

- $u : P(0_I)$ and $v : P(1_I)$
- $p : \mathrm{seg}_*(u) =_{P(1_I)} v$

then there is a function $f : \prod_{(x:I)} P(x)$ with

- $f(0_I) :\equiv u$ and $f(1_I) :\equiv v$
- $f(\mathrm{seg}) = p$.

Induction with the circle

The induction principle for the circle is that for every $P : \mathbb{S}^1 \to \mathcal{U}$, if there are

- $u : P(\mathsf{base})$
- $p : \mathsf{loop}_*(u) =_{P(\mathsf{base})} u$

then there is a function $f : \prod_{(x:\mathbb{S}^1)} P(x)$ with

- $f(\mathsf{base}) :\equiv u$
- $f(\mathsf{loop}) = p$.

Using univalence to reason about HITs

How do we use univalence to reason about HITs?

- Suppose we have a HIT W.
- and we want to describe a property $P : W \to \mathcal{U}$.
- for the point constructors of W we have to give types.
- for the path constructors of W we have to give paths between those types
- by univalence, it suffices to give equivalences between those types.

Suppose, in our inductive type W we have $p : x =_W y$ and $P(x) :\equiv A$ and $P(y) :\equiv B$ and to p we have assigned the equivalence $e : A \simeq B$.Then transporting along p computes as applying the equivalence e.

The universal cover, computing base $=_{\mathbb{S}^1}$ base

With this idea, we can construct the universal cover of the circle: $C : \mathbb{S}^1 \to \mathcal{U}$. Our goal is to use C to show that

$$(\mathsf{base} =_{\mathbb{S}^1} \mathsf{base}) \simeq \mathbb{Z}.$$

We define $C : \mathbb{S}^1 \to \mathcal{U}$ by:

- $C(\mathsf{base}) :\equiv \mathbb{Z}$
- To transport along loop we apply the equivalence $\mathsf{succ} : \mathbb{Z} \to \mathbb{Z}$.

Theorem

The cover C has the property that

$$\mathsf{isContr}\Big(\sum_{(x:\mathbb{S}^1)} C(x)\Big)$$

Before we prove the theorem let us indicate why it is useful.

- Suppose A, $a : A$ is a type and $P : A \to \mathcal{U}$.
- there is a term of $P(a)$.
- and $\sum_{(x:A)} P(x)$ is contractible.

Note that

- $\sum_{(x:A)} x =_A a$ is contractible as well
- by the assumption $P(a)$, there exists a function

$$f(x) : (x =_A a) \to P(x)$$

for every $x : A$.

Recall that

Theorem

If $f : \prod_{(x:A)} P(x) \to Q(x)$ *induces an equivalence*

$$(\sum_{(x:A)} P(x)) \to (\sum_{(x:A)} Q(x)),$$

then each $f(x) : P(x) \to Q(x)$ *is an equivalence.*

Hence under the above assumptions we obtain that

$$P(x) \simeq (x =_A a)$$

In particular, the theorem about the universal cover has the corollary that

$$C(x) \simeq (x =_{\mathbb{S}^1} \mathrm{base})$$

Hence, as a corollary of the theorem

Theorem

The cover C has the property that

$$\mathsf{isContr}\Big(\sum_{(x:\mathbb{S}^1)} C(x)\Big)$$

we get that

Theorem

$(\mathsf{base} =_{\mathbb{S}^1} \mathsf{base}) \simeq \mathbb{Z}$.

Proof:

- We first have to give the center of contraction: $(\mathsf{base}, 0)$
- Now we have to show

$$\prod_{x:\sum_{(s:\mathbb{S}^1)} C(s)} (x = (\mathsf{base}, 0))$$

which is equivalent to

$$\prod_{(t:\mathbb{S}^1)} \prod_{(u:C(x))} ((t, u) = (\mathsf{base}, 0))$$

We do this with induction on $\mathbb{S}^1$.

The base case

We have to show

$$\prod_{u:C(\mathsf{base})} ((\mathsf{base}, u) = (\mathsf{base}, 0)),$$

which is equivalent to

$$\prod_{(k:\mathbb{Z})} \sum_{(p:\mathsf{base} =_{\mathbb{S}^1} \mathsf{base})} (p_*(k) =_{C(\mathsf{base})} 0)$$

For $k : \mathbb{Z}$ we may simply take:

$$p :\equiv \mathsf{loop}^{-k}$$

and we have $p_*(k) = \mathsf{succ}^{-k}(k) = 0$.

We have obtained a term

$$\alpha : \prod_{(k:\mathbb{Z})} \sum_{(p:\mathsf{base}=_{\mathbb{S}^1}\mathsf{base})} p_*(k) =_{\mathbb{Z}} 0.$$

The next step is to show that

$$\mathsf{loop}_*(\alpha) = \alpha.$$

Notice that since $p_*(u) =_{\mathbb{Z}} 0$ is a proposition (and hence has proof irrelevance). It suffices to show that

$$\mathsf{loop}_*(\beta) = \beta,$$

where

$$\beta :\equiv \lambda k.\mathsf{loop}^{-k} : \prod_{k:\mathbb{Z}} \mathsf{base} =_{\mathbb{S}^1} \mathsf{base}$$

To show this, we have to understand how transporting along paths with respect to the family

$$t \mapsto \prod_{k:C(t)} t =_{\mathbb{S}^1} \mathsf{base}$$

works. I.e. we need to evaluate what it means to transport with respect to a dependent type $P : A \to \mathcal{U}$ when...

- ...the family P is a family of dependent products
- ...the family P is a family of identity types.

Transporting with respect to products

Lemma

Suppose the dependent type $B : A \to \mathcal{U}$ *and* $Q : \prod_{(x:A)}(B(x) \to \mathcal{U})$ *are families and define* $P : A \to \mathcal{U}$ *by*

$$P(x) :\equiv \prod_{b:B(x)} Q(x, b).$$

Let $p : x =_A y$ *be a path in* A *and let* $f : P(x)$*. Then*

$$\prod_{b:B} p_*(f)(b) = \tilde{p}_*\big(f(p^{-1}{}_*(b))\big)$$

where $\tilde{p} : (x, p^{-1}{}_*(b)) = (y, b)$ *is defined by path induction.*

Transporting with respect to identity types

Lemma

Let $f : A \to B$ be a function and let $b : B$. Define $P : A \to \mathcal{U}$ by

$$P(x) :\equiv (f(x) =_A b)$$

and let $p : x =_A y$ and $q : P(x)$. Then

$$p_*(q) = f(p)^{-1} \cdot q$$

where $\cdot$ stands for path concatenation.

We now apply these insights to the function

$$\beta :\equiv \lambda k.\mathsf{loop}^{-k} : \prod_{k:\mathbb{Z}} \mathsf{base} =_{\mathbb{S}^1} \mathsf{base}.$$

We have:

- $(\mathsf{loop}_*(\beta))(k) = \tilde{\mathsf{loop}}_*\big(\beta(\mathsf{succ}^{-1}(k))\big)$
- Notice that on the right, we have a transportation with respect to the fibration

$$P(x) :\equiv \mathsf{pr}_1(x) =_{\mathbb{S}^1} \mathsf{base}$$

  where $\mathsf{pr}_1 : \sum_{(t:\mathbb{S}^1)} C(t) \to \mathbb{S}^1$.
- Note that $\mathsf{pr}_1(\tilde{\mathsf{loop}}) = \mathsf{loop}$

Therefore, we have

$$\prod_{k:\mathbb{Z}} \tilde{\mathsf{loop}}_*\big(\beta(\mathsf{succ}^{-1}(k))\big) = \mathsf{loop} \cdot \beta(\mathsf{succ}^{-1}(k))$$

Now we finish the computation by:

$$\begin{aligned} {\mathsf{loop}^{-1}}_*\big(\beta(\mathsf{succ}^{-1}(k))\big) &:\equiv \mathsf{loop} \cdot \mathsf{loop}^{\mathsf{succ}^{-1}(k)} \\ &= \mathsf{loop}^k \\ &\equiv \beta(k). \end{aligned}$$

Hereby the proof is finished, and we conclude that
$\mathbb{S}^1$ is a 1-type, not a set.