Lawvere-Tierney Sheafification in Homotopy Type Theory

By

Kevin Quirin

Major Subject: Computer Science

Approved by the
Examining Committee:

_____
Pierre Cointe, Thesis Advisor

_____
Nicolas Tabareau, Thesis Advisor

_____
Carl F. Gauß, Göttingen

_____
Euclid, Athens

Mines de Nantes

2016

Contents

3 Higher modalities

As said in the introduction, the main purpose of our work is to build, from a model $\mathfrak{M}$ of homotopy type theory, another model $\mathfrak{M}'$ satisfying new principles. Of course, $\mathfrak{M}'$ should be describable *inside* $\mathfrak{M}$. In set theory, it corresponds to building *inner models* ([Kun]).In type theory, it can be rephrased in terms of left-exact modalities: it consists of an operator $\bigcirc$ on types such that for any type A, $\bigcirc A$ satisfies a desired property. If the operator has a "good" behaviour, then it is a modality, and, with a few more properties, the universe of all types satisfying the chosen property forms a new model of homotopy type theory.

Modalities are actually a generalization of *modal logics* and $\bigcirc$type theories [Mog91]

3.1 Modalities

Definition 3.1
A left exact modality is the data of

(i) *A predicate* $P : \text{Type} \to \text{HProp}$

(ii) *For every type* A*, a type* $\bigcirc A$ *such that* $P(\bigcirc A)$

(iii) *For every type* A*, a map* $\eta_A : A \to \bigcirc A$

such that

(iv) *For every types* A *and* B*, if* $P(B)$ *then*

$$\begin{cases} (\bigcirc A \to B) & \to & (A \to B) \\ f & \mapsto & f \circ \eta_A \end{cases}$$

is an equivalence.

(v) *for any* $A : \text{Type}$ *and* $B : A \to \text{Type}$ *such that* $P(A)$ *and* $\prod_{x:A} P(Bx)$*, then* $P\left(\sum_{x:A} B(x)\right)$

(vi) *for any* $A : \text{Type}$ *and* $x, y : A$*, if* $\bigcirc A$ *is contractible, then* $\bigcirc(x = y)$ *is contractible.*

Conditions (i) to (iv) define a reflective subuniverse*, (i) to (v) a* modality.

Notation – The inverse of $- \circ \eta_A$ from point *(iv)* will be denoted $\bigcirc_{\mathrm{rec}} : (A \to B) \to (\bigcirc A \to B)$, and its computation rule $\bigcirc^{\beta}_{\mathrm{rec}} : \prod_{f:A\to B} \prod_{x:A} \bigcirc_{\mathrm{rec}}(f)(\eta_A x) = f x$.

If $\bigcirc$ is a modality, the type of modal types will be denoted $\mathrm{Type}^{\bigcirc}$. Let us fix a left-exact modality $\bigcirc$ for the rest of this section. A modality acts functorialy on Type, in the sense that

Lemma 3.2 : Functoriality of modalities
Let A, B : Type and $f : A \to B$. Then there is a map $\bigcirc f : \bigcirc A \to \bigcirc B$ such that

- $\bigcirc f \circ \eta_A = \eta_B \circ f$
- *if* $g : B \to C$, $\bigcirc(g \circ f) = \bigcirc g \circ \bigcirc f$
- *if* IsEquiv f, *then* IsEquiv $\bigcirc f$.

Proposition 3.3
Any left-exact modality $\bigcirc$ satisfies the following properties[1].

- *(R) A is modal if and only if η_A is an equivalence.*
- *(R) $\mathbf{1}$ is modal.*
- *(R)* $\mathrm{Type}^{\bigcirc}$ *is closed under dependent products,* i.e. $\prod_{x:A} B x$ *is modal as soon as all $B x$ are modal.*
- *(R) For any types A and B, the map*

$$\bigcirc(A \times B) \to \bigcirc A \times \bigcirc B$$

  is an equivalence.
- *(R) If A is modal, then for all $x, y : A$, $(x = y)$ is modal.*
- *(M) For every type A and $B : \bigcirc(A) \to \mathrm{Type}^{\bigcirc}$, then*

$$- \circ \eta_A : \begin{array}{ccc} \prod_{z:\bigcirc A} B x & \longrightarrow & \prod_{a:A} B(\eta_A a) \\ f & \longmapsto & f \circ \eta_A \end{array}$$

  is an equivalence.
- *(M) If A, B :* Type *are modal, then so are* Is-n-type A, $A \simeq B$ *and* IsEquiv f *for all $f : A \to B$.*
- *(L) If $A : \mathrm{Type}_n$, then $\bigcirc A : \mathrm{Type}_n$.*

[1] Properties needing only a reflective subuniverse are annoted by (R), a modality by (M), a left-exact modality by (L)

•*(L) If* $X, Y : \mathrm{Type}$ *and* $f : X \to Y$*, then the map*

$$\bigcirc\big(\mathrm{fib}_f(y)\big) \to \mathrm{fib}_{\bigcirc f}(\eta_B y)$$

is an equivalence, and the following diagram commutes

$$\begin{array}{ccc} \mathrm{fib}_f(y) & \xrightarrow{\eta} & \bigcirc\big(\mathrm{fib}_f(y)\big) \\ {\scriptstyle\gamma}\big\downarrow & \swarrow & \\ \mathrm{fib}_{\bigcirc f}(\eta_B y) & & \end{array}$$

Notation – Again, the inverse of $- \circ \eta_A$ will be denoted $\bigcirc_{\mathrm{ind}} : \prod_{a:A} B(\eta_A a) \to \prod_{z:\bigcirc A} B\,x$, and its computation rule $\bigcirc^{\beta}_{\mathrm{ind}} : \prod_{f:\prod_{a:A} B(\eta_A a)} \prod_{x:A} \bigcirc_{\mathrm{ind}}(f)(\eta_A x) = f\,x$

Proof. We only prove the last point. It is straighforward to define a map

$$\phi : \sum_{x:X} f\,x = y \to \sum_{x:\bigcirc X} \bigcirc f\,x = \eta_Y y,$$

using η functions. We will use the following lemma to prove that the function induced by ϕ defines an equivalence:

Lemma 3.4
Let $X : \mathrm{Type}$*,* $Y : \mathrm{Type}^{\bigcirc}$ *and* $f : X \to Y$*. If for all* $y : Y$*,* $\bigcirc(\mathrm{fib}_f(y))$ *is contractible, then* $\bigcirc X \simeq Y$*.*

Hence we just need to check that every $\bigcirc$-fiber $\bigcirc(\mathrm{fib}_\phi(x;p))$ is contractible. Technical transformations allow one to prove

$$\mathrm{fib}_\phi(x;p) \simeq \mathrm{fib}_s(y;p^{-1})$$

for

$$s : \begin{array}{ccc} \mathrm{fib}_{\eta_X}(x) & \longrightarrow & \mathrm{fib}_{\eta_Y}(\bigcirc f\,x) \\ (a,q) & \longmapsto & (f\,a,-) \end{array}$$

But left-exctness allows to characterize the contractibility of fibers:

Lemma 3.5
Let $A, B : \mathrm{Type}$*. Let* $f : A \to B$*. If* $\bigcirc A$ *and* $\bigcirc B$ *are contractible, then so is* $\mathrm{fib}_f(b)$ *for any* $b : B$*.*

Thus, we just need to prove that $\bigcirc(\mathrm{fib}_{\eta_X}(a))$ and $\bigcirc(\mathrm{fib}_{\eta_Y}(b))$ are contractible. But one can check that η maps always satisfy this property. Finally, $\bigcirc(\mathrm{fib}_s(y;p^{-1}))$ is contractible, so $\bigcirc(\mathrm{fib}_\phi(x;p))$ also, and the result is proved. □

Let us finish these properties by the following proposition, giving an equivalent characterization of left-exactness.

Proposition 3.6
Let $\bigcirc$ be a modality. Then $\bigcirc$ is left-exact if and only if $\bigcirc$ preserves path spaces, i.e.

$$\prod_{A:\mathrm{Type}} \prod_{x,y:A} \mathrm{IsEquiv}(\bigcirc(\mathrm{ap}_{\eta_A}))$$

where $\bigcirc(\mathrm{ap}_{\eta_A}) : \bigcirc(x=y) \to \eta_A x = \eta_A y$.

Proof. We will rather prove something slighlty more general, using an encode-decode proof [HoTT, Section 8.9] ; we will characterize, for a type A and a fixed inhabitant $x : A$ the type

$$\eta_A x = y$$

for any $y : \bigcirc A$.

Let $\mathrm{Cover} : \bigcirc A \to \mathrm{Type}^{\bigcirc}$ be defined by induction by

$$\mathrm{Cover}(y) \overset{def}{=} \bigcirc_{\mathrm{rec}}(\lambda y, \bigcirc(x=y)).$$

Note that for any $y : \bigcirc A$, $\mathrm{Cover}(y)$ is always modal. We will show that $\eta_A x = y \simeq Cover(y)$. Now, let $\mathrm{Encode} : \prod_{y:\bigcirc A} \eta_A x = y \to \mathrm{Cover}(y)$ be defined by

$$\mathrm{Encode}(y,p) \overset{def}{=} \mathrm{transport}^{p}_{\mathrm{Cover}}\left(\mathrm{transport}^{\bigcirc^{\beta}_{\mathrm{rec}}((\lambda z, \bigcirc(x=z)),x)}_{\mathrm{idmap}}(\eta_{x=x}1)\right)$$

and $\mathrm{Decode} : \prod_{y:\bigcirc A} \mathrm{Cover}(y) \to \eta_A x = y$ by

$$\mathrm{Decode} \overset{def}{=} \bigcirc_{\mathrm{ind}}\left(\lambda y\, p, \bigcirc(\mathrm{ap}_{\eta_A})\left(\mathrm{transport}^{\bigcirc^{\beta}_{\mathrm{rec}}((\lambda z, \bigcirc(x=y)),y)}_{\mathrm{idmap}} p\right)\right)$$

Then one can show, using $\bigcirc$-induction and path-induction, that for any $y : \bigcirc A$, $\mathrm{Encode}(y,-)$ and $\mathrm{Decode}(y,-)$ are each other inverses. Then, taking $y' = \eta_A y$, we have just shown that $\eta_A x = \eta_A y \simeq Cover(\eta_A y)$, which is itself equivalent, by $\bigcirc^{\beta}_{\mathrm{rec}}$, to $\bigcirc(x=y)$. It is straightforward to check that the composition $\bigcirc(x=y) \to \mathrm{Cover}(\eta_A y) \to \eta_A x = \eta_A y$ is exactly $\bigcirc(\mathrm{ap}_{\eta_A})$.

Now, let us prove the backward implication. Let A be a type such that $\bigcirc A$ is contractible, and $x, y : A$. As $\eta_A x, \eta_A y : \bigcirc A$, we know that $\eta_A x = \eta_A y$ is contractible. But as $\eta_A x = \eta_A y \simeq \bigcirc(x=y)$ by assumption, $\bigcirc(x=y)$ is also contractible. □

As this whole thesis deals with truncation levels, it should be interesting to see how they are changed under a modality. We already know that if a type T is (-2)-truncated, *i.e.* contractible, then it is unchanged by the reflector:

$$\bigcirc T \simeq \bigcirc \mathbf{1} \simeq \mathbf{1} \simeq T.$$

Thus, Type_{-2} is closed by any reflective subuniverse. Now, let $T : \text{HProp}$. To check that $\bigcirc T$ is an h-proposition, it suffices to check that

$$\prod_{x,y:\bigcirc T} x = y$$

For any $x : \bigcirc T$, the type $\prod_{y:\bigcirc T} x = y$ is modal, as all $x = y$ are ; by the same argument, $\prod_{x:\bigcirc T} x = y$ is modal too for any $y : \bigcirc T$. Using twice the dependent eliminator of $\bigcirc$, it now suffices to check that

$$\prod_{x,y:T} \eta_T x = \eta_T y.$$

As T is supposed to be an h-proposition, this is true. It suffices to state

Lemma 3.7
For any modality, Type_{-1} *is closed under the reflector* $\bigcirc$*,* i.e.

$$\prod_{P:\text{HProp}} \text{IsHProp}(\bigcirc P).$$

A simple induction on the truncation level, together with the left-exactness property allows to state

Lemma 3.8
For any left-exact modality, all Type_p *are closed under the reflector* $\bigcirc$*,* i.e.

$$\prod_{P:\text{Type}_p} \text{Is-}p\text{-type}(\bigcirc P).$$

We note that this is not a equivalent characterization of left-exactness, as it is satisfied by truncations, and we will see they are not left-exact.

3.2 Examples of modalities

3.2.1 The identity modality

Let us begin with the most simple modality one can imagine: the one doing nothing. We can define it by letting $\bigcirc A \overset{def}{=} A$ for any type A, and $\eta_A \overset{def}{=} \text{idmap}$. Obviously, the desired computation rules are satisfied, so that the identity modality is indeed a left-exact modality.

It might sound useless to consider such a modality, but it can be precious when looking for properties of modalities: if it does not hold for the identity modality, it cannot hold for an abstract one.

3.2.2 Truncations

The first class of non-trivial examples might be the *truncations* modalities, seen in 2.4.1.

3.2.3 Double negation modality

Proposition 3.9

The double negation modality $\bigcirc A \overset{def}{=} \neg\neg A$ *is a modality.*

Proof. We define the modality with

(i) We will define the predicate P later.

(ii) $\bigcirc$ is defined by $\bigcirc A = \neg\neg A$

(iii) We want a term η_A of type $A \to \neg\neg A$. Taking

$$\eta_A \overset{def}{=} \lambda x : A, \lambda y : \neg A, y\, a$$

do the job.

Now, we can define P to be exactly $\prod_{A:\mathrm{Type}} \mathrm{IsEquiv}\, \eta_A$.

(iv) Let $A, B : \mathrm{Type}$, and $\varphi : A \to \neg\neg B$. We want to extend it into $\psi : \neg\neg A \to \neg\neg B$. Let $a : \neg\neg A$ and $b : \neg B$. Then $a(\lambda x : A, \varphi\, x\, b) : \mathbf{0}$, as wanted. One can check that it forms an equivalence.

(v) Let $A : \mathrm{Type}$ and $B : A \to \mathrm{Type}$ such that $P(A)$ and $\prod_{a:A} P(B\,a)$. There is a map

$$\sum_{x:A} B\,x \to A$$

thus by the preivous point, we can extend it into

$$\kappa : \neg\neg \sum_{x:A} B\,x \to A.$$

It remains to check that for any $x : \neg\neg \sum_{x:A} B\,x$, $B(\kappa\, x)$.

But the previous map can be easily extended to the dependent case, and thus it suffices to show that for all $x : \sum_{x:A} B\,x$, $B(\kappa(\eta\, x))$. As $\kappa \circ \eta =$ idmap, the goal is solved by $\pi_2 x$.

□

Unfortunately, it appears that the only type which can be modal are h-propositions, as they are equivalent to their double negation which is always an h-proposition. Thus, the type of modal types is consisted only of h-proposition, which is not satisfactory. The main purpose of this thesis, in particular chapter 5 is to extend this modality into a better one.

3.3 New type theories

New type theory: Enhance me. Actually, this might be better in the section "Translation".

We suppose here that $\bigcirc$ is a left-exact modality such that $\mathrm{Type}^{\bigcirc}$ is modal. This is for example the case when the modality is *accessible* (see [HoTT/Coq] for definition and proof).

Proposition 3.10
The modal universe $\mathrm{Type}^{\bigcirc}$ *is non-trivial if the type* $\bigcirc\mathbf{0}$ *is empty.*

Proof. □

The translation might be clearer than this result...

Proposition 3.11
A left exact modality $\bigcirc$ *induces a consistent type theory if and only if* $\bigcirc\mathbf{0}$ *can not be inhabited in the initial type theory. In that case, the modality is said to be consistent.*

Proof. By condition (iv) of Definition 3.1, $\bigcirc\mathbf{0}$ is an initial object of $\mathrm{Type}^{\bigcirc}$, and thus corresponds to false for modal mere proposition. As $\bigcirc\mathbf{1} = \mathbf{1}$, $\mathrm{Type}^{\bigcirc}$ is consistent when $\bigcirc\mathbf{0} \neq \mathbf{1}$, that is when there is no proof of $\bigcirc\mathbf{0}$. □

3.4 Truncated modalities

As for colimits, we define a truncated version of modalities, in order to use it in chapter 5. Basically, a truncated modality is the same as a modality, but restricted to Type_n.

Definition 3.12 : Truncated modality
Let $n \geq -1$ *be a truncation index. A left exact modality at level* n *is the data of*

(i) *A predicate* $P : \mathrm{Type}_n \to \mathrm{HProp}$

(ii) *For every* n*-truncated type* A*, a* n*-truncated type* $\bigcirc A$ *such that* $P(\bigcirc A)$

(iii) *For every* n*-truncated type* A*, a map* $\eta_A : A \to \bigcirc A$

such that

(iv) *For every* n*-truncated types* A *and* B*, if* $P(B)$ *then*

$$\begin{cases} (\bigcirc A \to B) & \to & (A \to B) \\ f & \mapsto & f \circ \eta_A \end{cases}$$

is an equivalence.

(v) for any $A : \mathrm{Type}_n$ *and* $B : A \to \mathrm{Type}_n$ *such that* $P(A)$ *and* $\prod_{x:A} P(Bx)$*, then* $P\left(\sum_{x:A} B(x)\right)$

(vi) for any $A : \mathrm{Type}_n$ *and* $x, y : A$*, if* $\bigcirc A$ *is contractible, then* $\bigcirc(x = y)$ *is contractible.*

Properties of truncated left-exact modalities described in 3.3 are still true when restricted to n-truncated types, except the one that does not make sense: $\mathrm{Type}_n^{\bigcirc}$ cannot be modal, as it is not even a n-truncated type.

3.5 Formalization

Let us discuss here about the formalization of the theory of modalities. General modalities are formalized in the Coq/HoTT library [HoTT/Coq], thanks to a huge work of Mike Shulman [Shu15]. The formalization might seem to be straightforward, but the universe levels (at least, their automatic handling by Coq) are here a great issue. Hence, we have to explicitely give the universe levels and their constraints in a large part of the library. For example, the reflector $\bigcirc$ of a modality is defined, in [HoTT/Coq] as

$$\bigcirc : \mathrm{Type}^i \to \mathrm{Type}^i;$$

it maps any universe to itself.

In chapter 5, we will need a slighlty more general definition of modality. The actual definitions stay the same, but the universes constraints we consider change. The reflector $\bigcirc$ will now have type

$$\bigcirc : \mathrm{Type}^i \to \mathrm{Type}^j, \quad i \leqslant j;$$

it maps any universe to a possibly higher one. Other components of the definition of a modality are changed in the same way.

Fortunately, this change seems small enough for all properties of modalities to be kept. Of course, the examples of modalities mapping any universe to itself are still an example of generalized modality, it just does not use the possibility to inhabit a higher universe. This has been computer-checked, it can be found at https://github.com/KevinQuirin/HoTT/tree/extended_modalities.

We would like to have the same generalization for truncated modalities. But there are a lot of new universe levels popping out, mostly because in $\mathrm{Type}_n = \sum_{T:\mathrm{Type}} \text{Is-}n\text{-type}\, T$, Is-$n$-type come with its own universes. Hence, handling "by hand" so many universes together with their constraints quickly go out of control. One idea to fix this issue could be to use *resizing rules* [Voe11], allowing h-propositions to live in the smallest universe. We could then get rid of the universes generated by Is-n-type, and treat the truncated modality exactly as generalized modalities.

3.6 Translation

As said in section 3.3, left-exact modalities allows to perform model transformation. But it can be enhanced a bit by exhibiting a *translation* of type theories, from CIC into itself, as it has been done for forcing [JTS12; Jab+16]. Let us explain here how this translation works.

Let $\bigcirc$ be an accessible left-exact modality. We describe, for each term constructor, how to build its translation. We denote $\pi_{\bigcirc}(A)$ the proof that $\bigcirc(A)$ is always modal.

- For types
$$[\mathrm{Type}] \overset{def}{=} (\mathrm{Type}^{\bigcirc}, \pi_{\mathrm{Type}^{\bigcirc}})$$
where $\pi_{\mathrm{Type}^{\bigcirc}}$ is a proof that $\mathrm{Type}^{\bigcirc}$ is itself modal. To ease the reading in what follows, we introduce the notation
$$[\![A]\!] \overset{def}{=} \pi_1\,[A]$$

- For dependent sums
$$\begin{aligned} [\textstyle\sum_{x:A} B] &\overset{def}{=} \left(\textstyle\sum_{x:[\![A]\!]} [\![B]\!], \pi_{\Sigma}^{[A],[B]}\right) \\ [(x,y)] &\overset{def}{=} ([x],[y]) \\ [\pi_i t] &\overset{def}{=} \pi_i [t] \end{aligned}$$
where $\pi_{\Sigma}^{A,B}$ is a proof that $\sum_{x:A} B$ is modal when A and B are.

- For dependent products
$$\begin{aligned} [\textstyle\prod_{x:A} B] &\overset{def}{=} \left(\textstyle\prod_{x:[\![A]\!]} [\![B]\!], \pi_{\Pi}^{[A],[B]}\right) \\ [\lambda\, x : A,\ M] &\overset{def}{=} \lambda\, x : [\![A]\!],\ [M] \\ [t\, t'] &\overset{def}{=} [t][t'] \end{aligned}$$
where $\pi_{\Pi}^{A,B}$ is a proof that $\prod_{x:A} B$ is modal when B is.

- For paths
$$\begin{aligned} [A = B] &\overset{def}{=} \left([A] = [B], \pi_{=}^{[A],[B]}\right) \\ [1] &\overset{def}{=} 1 \\ [J] &\overset{def}{=} J \end{aligned}$$
where $\pi_{=}^{A,B}$ is a proof that $A = B$ is modal when A and B are, if $A, B :$ Type, or a proof that $A = B$ is modal as soon as their type is modal if $A, B : X$.

- For positive types (we only treat the case of the sum as an example)

$$\begin{array}{lcl}
[A+B] & \stackrel{def}{=} & (\bigcirc([\![A]\!]+[\![B]\!]);\pi_{\bigcirc}([\![A]\!]+[\![B]\!])) \\
[\mathrm{in}_\ell t] & \stackrel{def}{=} & \eta(\mathrm{in}_\ell[t]) \\
[\mathrm{in}_r t] & \stackrel{def}{=} & \eta(\mathrm{in}_r[t]) \\
[\langle f,g\rangle] & \stackrel{def}{=} & \bigcirc_{\mathrm{rec}}^{[\![A]\!]+[\![B]\!]}\langle[f],[g]\rangle
\end{array}$$

- For truncations ($i \leqslant n$)

$$\begin{array}{lcl}
[\|A\|_i] & \stackrel{def}{=} & (\bigcirc\|[\![A]\!]\|_i;\pi_{\bigcirc}(\|[\![A]\!]\|_i)) \\
[|t|_i] & \stackrel{def}{=} & \eta|[t]|_i \\
[|f|_i] & \stackrel{def}{=} & \bigcirc_{\mathrm{rec}}^{\|[\![A]\!]\|_i}|[f]|_i
\end{array}$$

Let us make it more explicit on inductive types. In Coq, inductive types are all of the following form (we leave the mutual inductive types, behaving in the same way):

$$\begin{array}{c}
\mathbb{I}\ (a_1:A_1)\cdots(a_n:A_n):\mathrm{Type} \stackrel{def}{=} \\
|c_1 : \prod_{x_1:X_1^1}\cdots\prod_{x_{n_1}:X_1^{n_1}}\mathbb{I}(a_1^1,\ldots,a_1^n) \\
\vdots \\
|c_p : \prod_{x_1:X_p^1}\cdots\prod_{x_{n_1}:X_p^{n_1}}\mathbb{I}(a_p^1,\ldots,a_p^n)
\end{array}$$

with technical conditions on the X_i^j to avoid ill-defined types. For every such inductive type, we build a new one $\widehat{\mathbb{I}}$ defined by

$$\begin{array}{c}
\widehat{\mathbb{I}}\ (a_1:[\![A_1]\!])\cdots(a_n:[\![A_n]\!]):\mathrm{Type} \stackrel{def}{=} \\
|\widehat{c_1} : \prod_{x_1:\lceil X_1^1\rceil}\cdots\prod_{x_{n_1}:\lceil X_1^{n_1}\rceil}\widehat{\mathbb{I}}([a_1^1],\ldots,[a_1^n]) \\
\vdots \\
|\widehat{c_p} : \prod_{x_1:\lceil X_p^1\rceil}\cdots\prod_{x_{n_1}:\lceil X_p^{n_1}\rceil}\widehat{\mathbb{I}}([a_p^1],\ldots,[a_p^n])
\end{array}$$

where $\lceil\bullet\rceil$ is defined by

$$X = \begin{cases} \widehat{\mathbb{I}}([b_1],\ldots,[b_n]) & \text{if} \quad X \text{ is } \mathbb{I}(b_1,\ldots,b_n) \\ [\![X]\!] & \text{else} \end{cases}$$

Then, the translation of $\mathbb{I}(a_1,\dots,a_n)$ is defined as $\bigcirc(\widehat{\mathbb{I}}([a_1],\dots,[a_n]))$.

Example

Consider the following inductive

$$\begin{aligned}&\mathrm{q}\ (A : \mathrm{Type}) : \mathrm{Type} \stackrel{def}{=}\\ &\quad|\alpha : \mathrm{q}\mathbf{2}\\ &\quad|\beta : A \to \mathrm{q}\mathbf{1} \to \mathrm{q}A\end{aligned}$$

Then the inductive $\widehat{\mathrm{q}}$ is

$$\begin{aligned}&\widehat{\mathrm{q}}\ (A : \mathrm{Type}^{\bigcirc}) : \mathrm{Type} \stackrel{def}{=}\\ &\quad|\widehat{\alpha} : \mathrm{q}(\bigcirc\widehat{\mathbf{2}})\\ &\quad|\widehat{\beta} : \pi_1 A \to \widehat{\mathrm{q}}(\bigcirc\widehat{\mathbf{1}}) \to \widehat{\mathrm{q}}A\end{aligned}$$

and the translation of $\mathrm{q}A$ is thus $\bigcirc(\widehat{\mathrm{q}}[A])$.

Then, we translate contexts pointwise:

$$[\emptyset] \stackrel{def}{=} \emptyset$$
$$[\Gamma, x : A] \stackrel{def}{=} [\Gamma], x : [\![A]\!]$$

As in the forcing translation [JTS12], the main issue is that convertibility might not be preserved. For example, let $f : A \to X$ and $g : B \to X$. Then $\langle f, g\rangle(\mathrm{in}_\ell t)$ is convertible to $f\,t$, but $[\langle f, g\rangle(\mathrm{in}_\ell t)]$ reduces to $\bigcirc_{\mathrm{rec}}^{[\![A]\!]+[\![B]\!]}\langle [f], [g]\rangle(\eta(\mathrm{in}_\ell t))$, which is only equal to $[f\,t]$.

Two solutions to this problem can come to our mind:

- We could use the eliminator J of equality in the conversion rule, but this would require to use a type theory with explicit conversion in the syntax, like in [JTS12]. Such type theories has been studied in [GW; DGW13]. We do not chose this solution, and thus do not give more details.
- We can ask the modality $\bigcirc$ to be a strict modality, in the sense that retraction in the equivalence of $(- \circ \eta)$ is conversion instead of equality, like for the closed modality. That way, we keep the conversion rule, *i.e.* if $\Gamma \vdash u \equiv v$, then $[\![\Gamma]\!] \vdash [u] \equiv [v]$. That is the solution we chose.

We first want to show that the translation respects substitution:

Proposition 3.13
If $\Gamma \vdash A : \mathrm{Type}$, $\Gamma, x : A \vdash B : \mathrm{Type}$ *and* $\Gamma, x : A \vdash b : B$, *then*

$$[b[a/x]] \equiv [b][[a]/x].$$

Proof. It can be shown directly by induction on the term b.

□

The usual result we want about any translation is its soundness:

Proposition 3.14 : Soundness of the translation
Let Γ is a valid context, A a type and t a term. If $\Gamma \vdash t : A$, then

$$[\![\Gamma]\!] \vdash [t] : [\![A]\!].$$

Proof. We prove it by induction on the proof of $\Gamma \vdash t : A$. We use the name of rules as in [HoTT, Appendix A]. We note M the predicate "is modal".

- Π-FORM:

$$\dfrac{\dfrac{[\Gamma] \vdash [A] : [\![\mathrm{Type}]\!]}{[\Gamma] \vdash [\![A]\!] : \mathrm{Type}}\,\Sigma\text{-ELIM} \qquad \dfrac{[\Gamma, x : A] \vdash [B] : [\![\mathrm{Type}]\!]}{[\Gamma, x : A] \vdash [\![B]\!] : \mathrm{Type}}\,\Sigma\text{-ELIM}}{[\Gamma] \vdash \prod_{x:[\![A]\!]} [\![B]\!] : \mathrm{Type}}\,\Pi\text{-FORM}$$

  together with

$$\dfrac{\dfrac{[\Gamma] \vdash [A] : [\![\mathrm{Type}]\!]}{[\Gamma] \vdash [A]_2 : M(A)}\,\Sigma\text{-ELIM} \qquad \dfrac{[\Gamma, x : A] \vdash [B] : [\![\mathrm{Type}]\!]}{[\Gamma, x : A] \vdash [B]_2 : M(B)}\,\Sigma\text{-ELIM}}{[\Gamma] \vdash \pi_{\Pi}^{[A]_2, [B]_2} : M\left(\prod_{x:[\![A]\!]} [\![B]\!]\right)}$$

  yields

$$[\Gamma] \vdash \prod_{x:[\![A]\!]} [\![B]\!] : [\mathrm{Type}]$$

- Π-INTRO:

$$\dfrac{[\Gamma, x : A] \vdash [b] : [\![B]\!]}{[\Gamma] \vdash \lambda\, x : [\![A]\!], [b] : \prod_{x:[\![A]\!]} [\![B]\!]}\,\Pi\text{-INTRO}$$

- Π-ELIM:

$$\dfrac{[\Gamma] \vdash [f] : \prod_{x:[\![A]\!]} [\![B]\!] \qquad [\Gamma] \vdash [a] : [\![A]\!]}{[\Gamma] \vdash [f][a] : [\![B]\!][[a]/x]}\,\Pi\text{-ELIM}$$

- As the translation go through dependent sums as well, the proof trees for Σ-FORM, Σ-INTRO, Σ-ELIM and Σ-COMP are similar.

- Π-COMP:

$$\dfrac{[\Gamma, x : A] \vdash [b] : [\![B]\!] \qquad [\Gamma] \vdash [a] : [\![A]\!]}{[\Gamma] \vdash (\lambda\, x : [\![A]\!], [b])([a]) \equiv [b][[a]/x] : [\![B]\!][[a]/x]}\,\Pi\text{-COMP}$$

- $+$-FORM:

$$\dfrac{\dfrac{\dfrac{[\Gamma] \vdash [A] : [\![\mathrm{Type}]\!]}{[\Gamma] \vdash [\![A]\!] : \mathrm{Type}}\,\Sigma\text{-elim} \quad \dfrac{[\Gamma] \vdash [B] : [\![\mathrm{Type}]\!]}{[\Gamma] \vdash [\![B]\!] : \mathrm{Type}}\,\Sigma\text{-elim}}{[\Gamma] \vdash [\![A]\!] + [\![B]\!] : \mathrm{Type}}\,+\text{-form}}{[\Gamma] \vdash \bigcirc([\![A]\!] + [\![B]\!]) : [\![\mathrm{Type}]\!]}\,\pi_{\bigcirc}^{[\![A]\!]+[\![B]\!]}$$

- $+$-intro$_{1,2}$:

$$\dfrac{\dfrac{\dfrac{[\Gamma] \vdash [A] : [\![\mathrm{Type}]\!]}{[\Gamma] \vdash [\![A]\!] : \mathrm{Type}}\,\Sigma\text{-elim} \quad \dfrac{[\Gamma] \vdash [B] : [\![\mathrm{Type}]\!]}{[\Gamma] \vdash [\![B]\!] : \mathrm{Type}}\,\Sigma\text{-elim} \quad [\Gamma] \vdash [a] : [\![A]\!]}{[\Gamma] \vdash \mathrm{inl}([a]) : [\![A]\!] + [\![B]\!]}\,+\text{-intro}_1}{[\Gamma] \vdash \eta(\mathrm{inl}([a])) : [\![A + B]\!]}$$

and

$$\dfrac{\dfrac{\dfrac{[\Gamma] \vdash [A] : [\![\mathrm{Type}]\!]}{[\Gamma] \vdash [\![A]\!] : \mathrm{Type}}\,\Sigma\text{-elim} \quad \dfrac{[\Gamma] \vdash [B] : [\![\mathrm{Type}]\!]}{[\Gamma] \vdash [\![B]\!] : \mathrm{Type}}\,\Sigma\text{-elim} \quad [\Gamma] \vdash [b] : [\![B]\!]}{[\Gamma] \vdash \mathrm{inl}([b]) : [\![A]\!] + [\![B]\!]}\,+\text{-intro}_2}{[\Gamma] \vdash \eta(\mathrm{inl}([b])) : [\![A + B]\!]}$$

- $+$-elim: We treat the non-dependent case.

$$\dfrac{\dfrac{[\Gamma] \vdash [C] : [\![\mathrm{Type}]\!] \qquad [\Gamma] \vdash [f] : [\![A \to C]\!] \qquad [\Gamma] \vdash [d] : [\![B \to C]\!]}{[\Gamma] \vdash \langle [f], [g] \rangle : [\![A]\!] + [\![B]\!] \to [\![C]\!]}\,+\text{-elim}}{[\Gamma] \vdash \bigcirc_{\mathrm{rec}}^{[\![A]\!]+[\![B}\langle [f], [g] \rangle : \bigcirc([\![A]\!] + [\![B]\!]) \to [\![C]\!]}$$

- $+$-comp$_{1,2}$: Again, we only treat the non-dependent case.

$$\dfrac{\dfrac{[\Gamma] \vdash [C] : [\![\mathrm{Type}]\!] \qquad [\Gamma] \vdash [f] : [\![A \to C]\!] \qquad [\Gamma] \vdash [d] : [\![B \to C]\!] \qquad [\Gamma] \vdash [a] : [\![A]\!]}{[\Gamma] \vdash \langle [f], [g] \rangle(\mathrm{inl}[a]) \equiv [f][a] : [\![C]\!]}\,+\text{-comp}_1}{[\Gamma] \vdash \bigcirc_{\mathrm{rec}}^{[\![A]\!]+[\![B]\!]}\langle [f], [g] \rangle(\eta(\mathrm{inl}[a])) \equiv [f][a] : [\![C]\!]}\,\bigcirc_{\mathrm{strict}}$$

$+$-comp$_2$ is done in the same way.

- $=$-form:

$$\dfrac{\dfrac{\dfrac{[\Gamma] \vdash [A] : [\![\mathrm{Type}]\!]}{[\Gamma] \vdash [\![A]\!] : \mathrm{Type}}\,\Sigma\text{-elim} \quad [\Gamma] \vdash [a], [b] : [\![A]\!]}{[\Gamma] \vdash [a] = [b] : \mathrm{Type}}\,=\text{-form} \quad \dfrac{\dfrac{[\Gamma] \vdash [A] : [\![\mathrm{Type}]\!]}{[\Gamma] \vdash [A]_2 : M(A)}\,\Sigma\text{-elim}}{[\Gamma] \vdash \pi_{=}^{[a],[b]} : M([a] = [b])}}{[\Gamma] \vdash [a] = [b] : [\![\mathrm{Type}]\!]}\,\Sigma\text{-intro}$$

- $=$-intro:

$$\frac{[\Gamma] \vdash [A] : [\![\mathrm{Type}]\!] \qquad [\Gamma] \vdash [a] : [\![A]\!]}{[\Gamma] \vdash 1_{[a]} : [a] = [a]} =\text{-INTRO}$$

- $=$-ELIM: We only treat transport.

$$\frac{\dfrac{[\Gamma, x : A] \vdash [P] : [\![\mathrm{Type}]\!]}{[\Gamma, x : A] \vdash [\![P]\!] : \mathrm{Type}}\,\Sigma\text{-ELIM} \qquad [\Gamma] \vdash [a], [b] : [\![A]\!] \qquad [\Gamma] \vdash [p] : [a] = [b] \qquad [\Gamma] \vdash [u] : [\![P\,a]\!]}{[\Gamma] \vdash \mathrm{transport}^{[\![P]\!]}_{[p]}[u] : [P][b]} =$$

□

This allows to state the following theorem

Theorem 3.15
Let $\bigcirc$ *be a modality on* Type *such that*

- $\bigcirc$ *is left-exact*
- $\mathrm{Type}^{\bigcirc}$ *is itself modal*
- *For all* A, B, f, x, $\bigcirc^{\beta}_{\mathrm{rec}}(A, B, f, x) = 1$.

Then $\bigcirc$ *induces a new type theory.*

NOTA

Note that, for any (accessible) modality $\bigcirc$, we can define an equivalent strict modality, by defining an inductive type $\bigcirc^S$ generated by

$$\eta^S \quad : \quad \textstyle\prod_{A:\mathrm{Type}} A \to \bigcirc^S A$$

with an axiom asserting that all $\bigcirc^S A$ are $\bigcirc$-modal, and an induction principle restricted to $\bigcirc$-modals types

$$\textstyle\prod_{A:\mathrm{Type}} \prod_{B:\bigcirc^S A \to \mathrm{Type}^{\bigcirc}} \prod_{a:A} B(\eta^S_A a) \to \prod_{a:\bigcirc^S A} B\,a.$$

As in [Shu11], we can change the axiom by the equivalent

$$\prod_{A:\mathrm{Type}} \mathrm{IsEquiv}(\eta_A : A \to \bigcirc A),$$

and unfold the definition of IsEquiv to see that $\bigcirc^S$ is a valid HIT. We are actually building $\bigcirc^S$ as the *localization* [HTT, Definition 5.2.7.2] of $\bigcirc$-modal types. This idea is developed in [Shu12].

Then, it is straightforward to see that $\bigcirc^S$ is a strict modality, equivalent to $\bigcirc$.

According to the previous remark, the classical situation fulfilling the requirements is when the modality $\bigcirc$ is left-exact and accessible. One example is the closed modality [HoTT/Coq, Modalities/Closed.v].

Note that univalence axiom remains true in the new theory:

Proposition 3.16
Let $\bigcirc$ be as in theorem 3.15. Then the univalence axiom remains true in the new type theory.

Proof.

Lemma 3.17
We begin by showing that for any arrow $f : A \to B$, $[\mathrm{IsEquiv}(f)] = \mathrm{IsEquiv}[f]$.

Proof. Let $A, B : \mathrm{Type}$ and $f : A \to B$. Then

$$\mathrm{IsEquiv}(f) \overset{def}{=} \sum_{g:B\to A} \sum_{r:\prod_{y:B} f(g(y))=y} \sum_{s:\prod_{x:A} g(f(x))=x} \prod_{x:A} \mathrm{ap}_f r(x) = s(fx).$$

As the translation go through dependent product and sums, and through path elimination, it is straightforward that $[\mathrm{IsEquiv}(f)] = \mathrm{IsEquiv}[f]$. ◇

Now,

$$\begin{aligned}[\mathrm{idtoequiv}_{A,B}] &= [\lambda\,(p : A = B),\ \mathrm{transport}^p_{A\simeq\bullet}(\mathrm{id})] \\ &= \lambda\,(p : [\![A]\!] = [\![B]\!]),\ \mathrm{transport}^p_{[\![A]\!]\simeq\bullet}(\mathrm{id}) \\ &= \mathrm{idtoequiv}_{[\![A]\!],[\![B]\!]}\end{aligned}$$

As univalence is true in the original type theory, we have $A : \mathrm{Type}, B : \mathrm{Type} \vdash \mathrm{IsEquiv}(\mathrm{idtoequiv}_{A,B}$. By the soundness theorem, we have

$$A : \mathrm{Type}^{\bigcirc}, B : \mathrm{Type}^{\bigcirc} \vdash \mathrm{IsEquiv}(\mathrm{idtoequiv}_{A_1,B_1}).$$

□

Nota

We note that if the modality is not left-exact (or not accessible), like truncations modalities, then $\mathrm{Type}^{\bigcirc}$ is not itself modal. It is although still possible to write a translation, but we can only define it on a type theory with only one universe. Indeed, the judgement $\Gamma \vdash \mathrm{Type}^i : \mathrm{Type}^j$ cannot be expressed, and thus cannot be translated to $[\![\Gamma]\!] \vdash [\mathrm{Type}^i] : [\![\mathrm{Type}^j]\!]$.

An implementation of this translation has been made, in the form of a Coq plugin, available at https://github.com/KevinQuirin/translation-mod/. We give here a brief description. For each module, there is a table T containing a list of pairs consisting of constants c (resp. inductive type

i) together with its translation $[c]$ (resp. another inductive type $[i]$). Each time we need the translation of a constant, the plugin read this table to find it. Then, we add two new commands in Coq: `Modal Definition` and `Modal Translate`.

`Modal Definition` allows to give a definition in the reflective subuniverse;

```
Modal Definition foo :  bar using ○
```

open a new goal of type $[\texttt{bar}]$, waits for the user to give a proof, and define a new term $\texttt{foo}^m : [\texttt{bar}]$ and a new constant $\texttt{foo} : \texttt{bar}$ at the `Defined` command. Then, it adds in T a new pair $(\texttt{foo}, \texttt{foo}^m)$.

`Modal Translate` allows to translate automatically a previously existing constant or inductive type. If $\texttt{foo} \stackrel{def}{=} \texttt{qux} : \texttt{bar}$ is a constant,

```
Modal Translate foo using ○
```

computes the value of $[\texttt{foo}]$, and add in T the pair $(\texttt{foo}, [\texttt{foo}])$. If $I(x_1 : A_1) \cdots (x_n : A_n) : \text{Type}$ is an inductive type with p constructors $C_i : T_i$, then the plugin builds a new inductive type $I^m(x_1 : [A_1]) \cdots (x_n : [A_n]) : \text{Type}$ with constructors $C_i^m : [T_i]$, and add (I, I^m) in the table T. Then, the translation of $I(t_1, \cdots, t_n)$ will be $\bigcirc(I^m([t_1], \cdots, [t_n]))$.

What remains to be done

Bibliography

[DGW13] Floris van Doorn, Herman Geuvers, and Freek Wiedijk. "Explicit convertibility proofs in pure type systems". In: *Proceedings of the Eighth ACM SIGPLAN international workshop on Logical frameworks & meta-languages: theory & practice*. ACM. 2013, pp. 25–36 (cit. on p. 35).

[GW] Herman Geuvers and Freek Wiedijk. "A logical framework with explicit conversions". In: (cit. on p. 35).

[HoTT] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. IAS: http://homotopytypetheory.org/book, 2013 (cit. on pp. 28, 36).

[HoTT/Coq] The Univalent Foundations Program. *Coq HoTT library*. URL: https://github.com/HoTT/HoTT/ (cit. on pp. 31, 32, 39).

[HTT] Jacob Lurie. *Higher topos theory*. Annals of mathematics studies. Princeton, N.J., Oxford: Princeton University Press, 2009 (cit. on p. 38).

[Jab+16] Guilhem Jaber et al. "The Definitional Side of the Forcing". In: *LICS*. New York, United States, May 2016. DOI: 10.1145/http://dx.doi.org/10.1145/2933575.2935320. URL: https://hal.archives-ouvertes.fr/hal-01319066 (cit. on p. 33).

[JTS12] Guilhem Jaber, Nicolas Tabareau, and Matthieu Sozeau. "Extending type theory with forcing". In: *Logic in Computer Science (LICS), 2012 27th Annual IEEE Symposium on*. IEEE. 2012, pp. 395–404 (cit. on pp. 33, 35).

[Kun] Kenneth Kunen. *Set theory*. Ed. by North-Holland (cit. on p. 25).

[Mog91] Eugenio Moggi. "Notions of Computation and Monads". In: *Information and Computation* 93 (1991), pp. 55–92 (cit. on p. 25).

[Shu11] Mike Shulman. *Localization as an Inductive Definition*. Blog post. 2011. URL: https://homotopytypetheory.org/2011/12/06/inductive-localization/ (cit. on p. 38).

[Shu12] Mike Shulman. *All Modalities are HITs*. Blog post. 2012. URL: https://homotopytypetheory.org/2012/11/19/all-modalities-are-hits/ (cit. on p. 38).

[Shu15] Mike Shulman. *Module for Modalities*. Blog post. 2015. URL: http://homotopytypetheory.org/2015/07/05/modules-for-modalities/ (cit. on p. 32).

[Voe11] Vladimir Voevodsky. *Resising Rules - their use and semantic justification*. 2011 (cit. on p. 32).