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

Abstract

The main goal of this thesis is to define an extension of Gödel not-not translation to all truncated types, in the setting of homotopy type theory. This goal will use some existing theories, like Lawvere-Tierney sheaves theory in toposes, we will adapt in the setting of homotopy type theory. In particular, we will define a Lawvere-Tierney sheafification functor, which is the main theorem presented in this thesis.

To define it, we will need some concepts, either already defined in type theory, either not existing yet. In particular, we will define a theory of colimits over graphs as well as their truncated version, and the notion of truncated modalities, based on the existing definition of modalities.

Almost all the result presented in this thesis are formalized with the proof assistant Coq together with the library [HoTT/Coq].

Résumé

Le but principal de cette thèse est de définir une extension de la traduction de double-négation de Gödel à tous les types tronqués, dans le contexte de la théorie des types homotopique. Ce but utilisera des théories déjà existantes, comme la théorie des faisceaux de Lawvere-Tierney, que nous adapterons à la théorie des types homotopiques. En particulier, on définira le fonction de faisceautisation de Lawvere-Tierney, qui est le principal théorème présenté dans cette thèse.

Pour le définir, nous aurons besoin de concepts soit déjà définis en théorie des types, soit non existants pour l'instant. En particulier, on définira une théorie des colimits sur des graphes, ainsi que leur version tronquée, et une notion de modalités tronquées basée sur la définition existante de modalité.

Presque tous les résultats présentés dans cette thèse sont formalisée avec l'assistant de preuve Coq, muni de la librairie [HoTT/Coq].

Acknowlegdments

Write the acknowledgments

Contents

1 Résumé en français

1.1 Introduction

La théorie des types homotopique est une branche nouvelle des mathématiques et de l'informatique, exhibant un lien fort, mais surprenant entre la théorie des ω-catégories et la théorie de types. Ce domaine se situe donc à la frontière entre les mathématiques pures et l'informatique. Un des but de la recherche sur ce sujet est d'utiliser la théorie des types homotopique comme une nouvelle fondation des mathématiques, remplaçant par exemple la théorie des ensembles de Zermelo-Frænkel. Ses liens forts avec la théorie des types donneraient aux mathématiciens la possibilité de formaliser leurs travaux avec un assistant de preuve comme Coq [The12], Agda [Nor07] ou Lean [Mou+15]. En effet, les erreurs dans les articles de recherche en mathématiques semblent inévitables [Voe14], et une preuve vérifiée par ordinateur peut inspirer plus confiance qu'une preuve vérifiée par un humain. Les exemples les plus célèbres de preuves vérifiées par ordinateur sont le *théorème des quatre couleurs* (pour colorier une carte telle qu'aucuns pays voisins aient la même couleur, il suffit de quatre couleurs) par Gonthier et Werner [Gon08] avec Coq, le *théorème de Feit-Thomson* (tout groupe fini d'ordre impair est résoluble) par Gonthier et al. [Gon+13] avec Coq, la preuve originelle du *théorème de Jordan* (toute courbe simple continue fermée divise le plan en une partie bornée "intérieure" et une partie non bornée "extérieure") par Hales [Hal07] avec Mizar, ou la *conjecture de Kepler* (les façons les plus compactes d'empiler des sphères sont les empilements clos cubiques et hexagonaux) par Hales et al. [Hal+15] avec Isabelle et HOL Light.

Un des avantages de la théorie des types par rapport à la théorie des ensembles est la propriété de calculabilité de la théorie des types : tout terme est identifié avec sa forme normale. Ainsi, un assistant de preuve permet à l'utilisateur de simplifier automatiquement toutes les expressions, alors qu'une preuve sur papier requiert de faire toutes ces calculs à la main. La théorie des ensembles ne partage pas cette propriété de calculabilité, et n'est donc pas aussi pratique à utiliser comme base théorique pour un assistant de preuve. Cependant, cette propriété de calculabilité nous empêche d'utiliser des propriétés classiques, telles que le tiers exclu ; en général, il est impossible de prouver qu'une proposition est soit vraie, soit fausse.

L'ingrédient principal de la théorie des types homotopique, faisant le lien entre les mathématiques et l'informatique, est l'isomorphisme de Curry-

Howard : on peut indifféremment parler de preuve ou de programme, les deux décrivent les mêmes objets *via* une correspondance. Par exemple, en théorie de types, la séquence de symboles $A \to B$ peut être vue comme le type des programmes prenant un argument de type A et produisant une sortie de type B ou comme le type des preuves que A implique B. Une chose qu'implique cette correspondance est qu'il peut exister plusieurs preuves de "A implique B", puisqu'il peut y avoir plusieurs manières de produire une sortie de type B à partir d'une entrée de type A. Cette propriété est appelée *proof-relevance*, alors que ZFC est considérée comme *proof-irrelevant* : si un lemme a été prouvé, la façon de le prouver peut être oublié, elle n'a aucune importance.

À part le manque de propriétés classique, un problème de la théorie des types est la notion d'égalité. On a deux possibilités :

- une égalité intentionnelle, ou définitionnelle ; deux objets sont égaux s'ils sont définis de la même manière, *i.e.* si l'un peut être échangé avec l'autre sans changer le sens. Par exemple, le nombre naturel 1 et le successeur du nombre naturel 0 sont intentionnellement égaux. En théorie des types, on ajoute des règles à cette égalité comme la règle β $((\lambda\ x,\ f\ x)y = f\ y)$ and η.
- une égalité extensionnelle, ou propositionelle ; deux objets sont considérés égaux s'ils se comportent de la même façon. Par exemple, étant donnés deux nombres naturels a and b, $a + b$ et $b + a$ sont extensionnellement mais pas intentionnellement égaux ; on a besoin de le prouver.

En théorie des ensembles, on utilise traditionnellement une égalité extensionnelle, en affirmant que deux ensembles sont égaux si et seulement s'ils ont les mêmes éléments. En théorie des types, l'égalité intentionnelle est une notion meta-théorique ; seul le type-checker (comme Coq) peut y accéder. On ne peut pas l'exprimer dans la théorie elle-même car on sait que la théorie des types extensionnelle est indécidable [Hof95] (étant donnés un terme p et un type P, il peut être indécidable de vérifier que p est bien une preuve de P). L'égalité propositionnelle est un concept interne à la théorie, définie comme un type inductif inductive type

$$\mathrm{Id}(A : \mathrm{Type})(a : A) : A \to \mathrm{Type}$$

généré par un seul constructeur

$$\mathrm{idpath} : \mathrm{Id}_A(a,a),$$

et le type $\mathrm{Id}(A,a,b)$ sera noté $a =_A b$ ou $a = b$. Ce type identité n'est en fait pas satisfaisant. L'idée de Martin-Löf était de copier l'égalité mathématique, mais le type identité n'y arrive pas. En effet, un problème avec cette égalité est que les types $a = b$ peuvent être habités de différents manières – au moins, il n'est pas prouvable que pour tous $p, q : a = b$, $p = q$; cette propriété, appelée UIP (unicité des preuves d'égalité) a été prouvée dans [HS96] être indépendante de la théorie des types intentionnelle. Un autre problème est que cette égalité

est définie pour tous les types, et ne se comporte pas bien vis-à-vis de certains constructeurs de types ; par exemple, l'extensionalité fonctionnelle, affirmant que deux fonctions sont égales dès qu'elles sont égales point-à-point, ne peut pas être montrée.

Cependant, il ne faut pas jeter les types identité. Vers 2006, Vladimir Voevodsky, et Steve Awodey et Michal Warren [AW09] ont donné indépendamment une nouvelle interprétation des types identité : les types sont maintenant vus comme des espaces topologiques, les habitants des types comme des points, et un élément $p : \mathrm{Id}(A, a, b)$ peut être lu comme

> Dans l'espace A, p est une homotopie (ou un chemin continu) entre les points a et b.

Avec cette interprétation, il paraît normal de ne pas satisfaire UIP : il peut y avoir plusieurs (*i.e.* non homotopiques) chemins entre deux points (on peut penser à un beignet). Le second problème a été résolu vers 2009, quand Vladimir Voevodsky a énoncé l'axiome d'univalence : deux types sont égaux exactement quand ils sont isomorphes. De façon surprenante, cette axiome implique la compatibilité des types identité avec certains constructeurs de types : il implique l'extensionalité fonctionnelle, et il semble qu'il implique aussi que le type identité sur les *streams* coïncide avec la bisimulation [Lic].

Le projet originel de Voevodsky [Voe10] était de donner un outil aux mathématiciens pour qu'il puissent vérifier leurs preuves sur ordinateur. La théorie des types homotopique semble être une bonne base pour ça, mais il manque le *tiers-exclu*, un des principes préférés des mathématiciens :

> Toute proposition est soit vraie, soit fausse.

Le but principal de cette thèse est d'ajouter ce principe à la théorie des types homotopique, sans perdre certaines propriétés (décidabilité, canonicité, constructivisme).

Commençons par remarquer qu'on sait déjà transformer une logique intuitionniste en une logique classique, avec la traduction de Gödel-Gentzen définie dans la figure 1.1

$$x^N \overset{def}{=} \neg\neg x \text{ quand } x \text{ est atomique}$$

$$(\phi \wedge \psi)^N \overset{def}{=} \phi^N \wedge \psi^N \qquad (\phi \vee \psi)^N \overset{def}{=} \neg(\neg\phi^N \wedge \neg\psi^N)$$

$$(\phi \to \psi)^N \overset{def}{=} \phi^N \to \psi^N \qquad (\neg\psi)^N \overset{def}{=} \neg\phi^N$$

$$(\forall\, x,\, \phi)^N \overset{def}{=} \forall\, x,\, \psi^N \qquad (\exists\, x,\, \phi)^N \overset{def}{=} \neg\forall\, x\, \neg\phi^N$$

Figure 1.1 : Traduction de Gödel-Gentzen

Un théorème de correction affirme qu'une formule ϕ est classiquement prouvable si et seulement si ϕ^N est intuiitionnistiquement prouvable. Bien

que cette traduction ne fonctionne qu'avec la logique, la même idée peut être utilisée pour toute la théorie des types, comme dans [JTS12 ; Jab+16]. L'idée derrière une traduction est, en partant d'une théorie source (compliquée) $\mathcal{S}$, de traduire tous les termes t de $\mathcal{S}$ en des termes $[t]$ d'une théorie source $\mathcal{T}$, supposée sue consistante. La propriété fondamentale d'une traduction est sa correction, ie si on peut prouver un théorème de correction affirmant que si un terme x est de type X dans $\mathcal{S}$, alors la traduction $[x]$ de x est de type $[\![X]\!]$, où $[\![\cdot]\!]$ est la traduction des types. Cette propriété, avec une preuve que $[\![\mathbf{0}]\!]$ n'est pas habité dans $\mathcal{T}$, assure que la théorie $\mathcal{S}$ est consistante. On peut dire qu'une traduction correcte est un moyen de donner un nom dans la théorie cible $\mathcal{T}$ aux objets de $\mathcal{S}$ inconnus de $\mathcal{T}$.

Les théoriciens de ensembles peuvent remarquer que c'est très proche de la méthode de *forcing*, inventée en 1962 par Paul Cohen [Coh66]. Son application historique, et la plus connue, la preuve d'indépendance de la négation de l'hypothèse du continu avec ZFC. De la même façon que les traductions, le forcing ne peut montrer que des résultats de consistance relative, *e.g.* ZFC+¬HC est consistant si ZFC est consistent. Cette méthode est aujourd'hui un ingrédient clef des théoriciens de ensembles.

Nous savons déjà qu'adapter des résultats de théorie des ensembles à la théorie des types n'est pas facile, ces deux théories étant très différents. La théorie des types est plus proche de la théorie des topoi, et la théorie des types homotopique encore plus proche de la théorie des topoi supérieurs. Heureusement, Myles Tierney a donné en 1972 un équivalent du forcing en théorie des topoi [Tie72], en utilisant la notion de faisceaux. À l'origine, les faisceaux n'existaient que sur des topoi de préfaisceaux (les topoi de foncteurs d'une catégorie $\mathbf{C}$ vers la catégorie **Sets**). Ces faisceaux sont appelés faisceaux de Grothendieck, et correspondent aux objets F tels que toute fonction $X \to F$ peut être définie de façon équivalence soit sur X tout entier, soit sur tous les objets d'un recouvrement ouvert de X. Ce concept a été étendu par William Lawvere et Myles Tierney, autorisant les objets de n'importe quel topos à être des faisceaux. Ils correspondent alors aux objets F tels que toute fonction $X \to F$ peut être définie de façon équivalente soit sur X tout entier, soit sur un sous-objet dense de X. L'existence d'un *foncteur de faisceautisation* de n'importe quel topos $\mathcal{T}$ vers son topos de faisceau $\mathrm{Sh}(\mathcal{T})$, adjoint à gauche de l'inclusion, permet de construire une topos satisfaisant plus de propriétés que le topos de départ. Les faisceaux proviennent d'une topologie fixée au départ, qui peut être vue comme un opérateur sur la "logique" du topos (le classifiant des sous-objets), idempotent, préservant $\top$ et commutant avec les produits. La double négation est en fait une topologie sur n'importe quel topos $\mathcal{T}$, et alors le topos $\mathrm{Sh}_{\neg\neg}(\mathcal{T})$ satisfait de bonnes propriétés il est booléen (*i.e.* satisfait le tiers exclu), et la négation de l'hypothèse du continu est vraie dans ce topos [MM92].

En théorie des topoi supérieurs, ce $(\infty,1)$-foncteur de faisceautisation n'a été défini dans [HTT] que dans le cas des faisceaux de Grothendieck, laissant la théorie de Lawvere-Tierney inexplorée. Il y a donc un défi double dans

notre quête de tiers-exclu en théorie des types homotopique : le premier est de formaliser le résultat de la théorie des topoi, et le second est de l'étendre au cas de la théorie des types homotopique, en utilisant des résultats de la théorie des $(\infty,1)$-topoi.

But de la thèse Le but principal de cette thèse est de donner une définition du foncteur de faisceautisation de Lawvere-Tierney dans le contexte de la théorie des types homotopique. Pour cela, on aura d'abord besoin d'une théorie des colimites en théorie des types homotopique. Puis, comme notre définition de la faisceautisation sera faite inductivement sur le niveau de troncation, on définira une version tronquée ce ces colimites, et une version tronquée des modalités exactes à gauche. Presque tous ces résultats sont vérifiés par ordinateur par l'assistant de preuve Coq ; la plupart d'entre eux sont disponibles sur mon compte Github https://github.com/KevinQuirin.

Notre étude approfondie des modalités nous a aussi amené à définir une traduction de théorie des types associée à une modalité exacte à gauche, et à écrire une extension pour Coq pour manipuler automatiquement cette traduction.

1.2 Théorie des types homotopique

Dans cette section, nous allons décrire brièvement le cadre dans lequel on se place, la théorie des types homotopique. La définition compacte de cette théorie pourrait être

$$\mathrm{HoTT} = \mathrm{MLTT} + \mathrm{UA} + \mathrm{HIT}$$

où MLTT est la théorie des types de Martin-Löf (ou théorie des types dépendants), UA est l'axiome d'univalence et HIT est l'abréviation des types inductifs supérieurs. Pour une description plus complète de MLTT, le lecteur pourra se référer à [Hof97] ou [HoTT].

Dans la théorie des ensembles de Zermelo-Frankæl, la formule logique la plus basique est

$$x \in E$$

où x et E sont des ensembles. En théorie des types dépendants, un jugement similaire serait

$$a : A$$

qui doit être lu comme "a est de type A". La différence principale avec la relation d'appartenance est qu'un élément a n'appartient qu'à un et un seul type, alors qu'il est possible d'écrire $x \in E$ et $x \in F$ en théorie des ensembles (c'est la définition de $x \in E \cap F$).

La théorie des types dépendants est basée sur la correspondance de Curry-Howard, ou principe de "proposition comme des types". En effet, on ne fait pas de différence entre les proposition et les types ; $a : A$ sera lu indifféremment "a est de type A" si A est vu comme un type ou "a est une preuve de A" quand A est vu comme une proposition.

On ne rappelle pas dans ce résumé les règles d'introduction et d'élimination des types, mais on donne le tableau suivant rappelant les différents types existants :

Nom	Notation	Propositions comme des types
Zéro	$\mathbf{0}$	$\bot$
Un	$\mathbf{1}$	$\top$
Coproduit, sum	$A + B$	$A \vee B$
Fonction	$A \to B$	$A \Rightarrow B$
Fonction dépendante, type Pi	$\prod_{x:A} B\,x$	$\forall x, B\,x$
Produit	$A \times B$	$A \wedge B$
Somme dépendante, type Sigma	$\sum_{x:A} B\,x$	$\exists x, B\,x$
Égalité, identité, chemin	$a =_A b$	$a = b$

Revenons simplement sur les types identité. Ils sont très utiles pour affirmer l'égalité propositionnelle entre des objets. On note qu'ils ne caractérisent pas l'égalité jugementale, qu'on considère comme appartenant à la métathéorie. Les types identités donnent à tout type une structure d'ω-groupoïde, *i.e.* la structure d'une ω-catégorie où toutes les flèches sont inversibles. En particulier, ils vérifient la réflexion, la transitivité, la symétrie, l'associativité, le pentagone de Maclane, *etc.*

Les types identités se comportent bien vis-à-vis des fonctions. Si $f : A \to B$ est une fonction, alors pour tous $x, y : A$, il existe une fonction

$$\mathrm{ap}_f : x = y \to f(x) = f(y)$$

compatible avec la structure d'ω-groupoïde.

Avant de lier les type identités avec les types d'équivalence, définissons d'abord les équivalences.

Définition 1.1
Soient A, B : Type *et* $f : A \to B$. *On dit que* f *est une équivalence, noté* $\mathrm{IsEquiv}(f)$ *s'il existe*

- *une fonction* $g : B \to A$, *appelée inverse de* f
- *un terme* $\mathrm{retr}_f : \prod_{x:A} g(f(x)) = x$, *appelé la rétraction de l'équivalence*

- *un terme* $\mathrm{sect}_f : \prod_{x:A} f(g(x)) = x$, *appelé la section de l'équivalence*
- *un terme* $\mathrm{adj}_f : \prod_{x:A} \mathrm{ap}_f \mathrm{retr}_f x = \mathrm{sect}_f(f\,x)$, *appelé l'adjonction de l'équivalence.*

On dit que A et B sont équivalents, noté $A \simeq B$ s'il existe $f : A \to B$ qui soit une équivalence.

On note que si $A = B$, alors il existe une équivalence canonique entre A et B, et on note $\mathrm{idtoequiv} : A = B \to A \simeq B$. On peut alors énoncer l'axiome d'univalence :

Axiome 1.2
Pour tous types A et B, la fonction

$$\mathrm{idtoequiv} : A = B \to A \simeq B$$

est une équivalence.

Avec cet axiome, il semble que les types identité sont compatibles avec tous les constructeurs de types :

- l'égalité dans les produits correspond à l'égalité des composantes
- l'égalité dans les sommes dépendantes correspond à l'égalité des composantes, la deuxième transportée par la première
- l'égalité dans les fonctions (dépendantes ou non) correspond à l'égalité point à point

Pour obtenir la théorie des types homotopiques, il reste à rajouter un moyen de construire des types dont on contrôle – en partie – les espaces de chemins itérés : les types inductifs supérieurs (HIT). Les HIT fonctionnent de la même manière que les types inductifs de Coq, mais on peut aussi rajouter des constructeurs pour les espaces de chemins.

Exemple

Le premier exemple est le cercle $\mathbb{S}^1$. Il est généré par les constructeurs

$$\left|\begin{array}{lcl} \mathrm{base} & : & \mathbb{S}^1 \\ \mathrm{loop} & : & \mathrm{base} = \mathrm{base} \end{array}\right.$$

On peut le représenter par

Comme pour les types inductifs, on donne aussi des éliminateurs pour les HIT.

Exemple

Donnons l'éliminateur non-dépendant de $\mathbb{S}^1$. Si P est un type, $b : P$ et $p : b = b$, alors il existe une fonction

$$\mathbb{S}^1_{\text{rec}} : \mathbb{S}^1 \to P$$

tel que $\mathbb{S}^1_{\text{rec}}(\text{base}) \equiv b$ et $\text{ap}_{\mathbb{S}^1_{\text{rec}}}(\text{loop}) = p$.

L'éliminateur dépendant est un peu plus compliqué. Si $P : \mathbb{S}^1 \to \text{Type}$ est une famille de type sur $\mathbb{S}^1$, $b : (P\text{base})$ et $p : \text{transport}^{\text{loop}}_P(b) = b$, alors il existe un terme

$$\mathbb{S}^1_{\text{ind}} : \prod_{x:\mathbb{S}^1} P\,x$$

tel que $\mathbb{S}^1_{\text{ind}}(\mathit{base}) \equiv b$ et $\text{ap}_{\mathbb{S}^1_{\text{ind}}}(\text{loop}) = p$.

1.3 Colimites

On voit dans [AKL15] que la présence des sommes dépendantes en théorie des types permet de construire des limites sur des graphes. Cependant, alors que ce n'est qu'une notion duale, on ne sait pas traiter le cas des colimites dans MLTT. On va voir dans cette partie que les types inductifs supérieurs permettent de gérer les colimites au-dessus de graphes, et même au-dessus de catégories dans certains cas.

Commençons par rappeler les définitions de graphes et diagrammes.

Définition 1.3 : Graphe
Un graphe *G est la donnée de*

- *un type G_0 de sommets ;*
- *pour tous $i, j : G_0$, un type $G_1(i, j)$ d'arêtes.*

Définition 1.4 : Diagramme
Un diagramme *D sur un graphe G est la donnée de*

- *pour tous $i : G_0$, un type $D_0(i)$;*
- *pour tous $i, j : G_0$ et tous $\phi : G_1(i, j)$, une flèche $D_1(\phi) : D_0(i) \to D_0(j)$*

Comme dans le contexte catégorique, on va essayer de définir une colimite comme un type qui forme un cocone sur le graphe désiré, de façon universelle. Dans la suite, on se fixe un graphe G et un diagramme D au-dessus de G.

Définition 1.5
Soit Q un type. Un cocone sur D dans Q est la donnée de flèches $q_i : D_i \to Q$, et pour tous $i, j : G$ et $g : G(i,j)$, une homotopie $q_j \circ D(g) \sim q_i$.

On peut postcomposer les cocones par des flèches. Plus précisément, si Q et Q' sont des types, $f : Q \to Q'$ et C un cocone sur D dans Q, alors il existe un cocone sur D dans Q' qui consiste à postcomposer toutes les flèches de C par f. Cela donne une flèche

$$\text{postcompose}_{\text{cocone}} : \text{cocone}_D(Q) \to (Q' : \text{Type}) \to (Q \to Q') \to \text{cocone}_D(Q')$$

L'autre sens correspond à notre définition de colimite :

Définition 1.6 : Colimite
Un type Q est une colimite de D s'il existe un cocone C sur C dans Q, qui est universel, i.e. *tel que* $\text{postcompose}_{\text{cocone}}(C, Q')$ *soit une équivalence.*

Les colimites sont compatibles avec différentes opérations sur les diagrammes, que nous n'expliciterons pas ici. On notera simplement que deux diagrammes équivalents ont des colimites équivalentes, donnant par suite l'unicité à équivalence près des colimites.

Regardons un cas particulier de colimites qui nous intéressera plus tard : les constructions que Van Doorn et de Boulier. Elles généralisent le théorème de théorie des topoi affirmant que tout épimorphisme est la colimite de sa kernel pair. En théorie des topoi supérieurs, le résultat reste vrai en remplaçant "kernel pair" par "nerf de Čech". Cependant, on ne sait pas définir en théorie des types les nerf de Čech, qui sont une version particulière d'objets simpliciaux. On va donc donner un autre diagramme dépendant d'une fonction dont la colimite est l'image de la fonction. On commence par la construction de Van Doorn, qui correspond à une fonction $A \to \mathbf{1}$, et donnant la (-1)-troncation comme colimite d'un diagramme.

Proposition 1.7 : Construction de Van Doorn
Soit A : Type. On définit le type inductif supérieur TA comme la colimite (le coégaliseur) de

$$A \times A \underset{\pi_2}{\overset{\pi_1}{\rightrightarrows}} A \ .$$

Alors la colimite de

$$A \xrightarrow{q} TA \xrightarrow{q} TTA \xrightarrow{q} TTTA \xrightarrow{q} \dots$$

est $\|A\|_{-1}$.

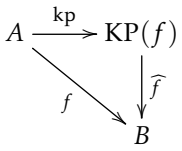
Maintenant, soient A, B : Type et $f : A \to B$. On définit la kernel pair de f comme la colimite de

$$A \times_B A \underset{\pi_2}{\overset{\pi_1}{\rightrightarrows}} A \ .$$

en d'autres termes, $\mathrm{KP}(f)$ est le type inductif supérieur généré par

$$\left| \begin{array}{lcl} \mathrm{kp} & : & A \to \mathrm{KP}(f) \\ \alpha & : & \prod_{x,y:A} f\,x = f\,y \to kp\,x = \mathrm{kp}\,y \end{array} \right.$$

En utilisant l'éliminateur des colimites, on peut construire une fonction $\widehat{f}$: $\mathrm{KP}(f) \to B$, telle que le digramme suivant commute



On peut alors construire $KP(\widehat{f})$ et une fonction $\widehat{\widehat{f}} : \mathrm{KP}(\widehat{f}) \to B$, *etc.* On a le résultat suivant

Proposition 1.8 : Construction de Boulier
Pour tout $f : A \to B$, $\mathrm{Im}(f)$ est la colimite de la kernel pair itérée de f

$$A \longrightarrow \mathrm{KP}(f) \longrightarrow \mathrm{KP}\left(\widehat{f}\right) \longrightarrow \mathrm{KP}\left(\widehat{\widehat{f}}\right) \longrightarrow \cdots$$

En particulier, si f est une surjection, la colimite de ce diagramme est B.

Un problème de ces deux constructions est qu'ils ne préservent pas le niveau de troncation (resp. les plongements). Si A est un HProp, TA peut avoir un niveau de troncation élevé (resp. si f est un plongement, $\widehat{f}$ ne l'est pas forcément). Ce problème peut être résolu en considérant des graphes avec compositions, et en définissant les deux types suivants

$$TA \left| \begin{array}{lcl} q & : & A \to TA \\ \alpha & : & \prod_{x,y:A} q\,x = q\,y \\ \alpha_1 & : & \prod_{x:A} \alpha(x,x) = 1 \end{array} \right. \quad ; \quad \mathrm{KP}(f) \left| \begin{array}{lcl} \mathrm{kp} & : & A \to \mathrm{KP}(f) \\ \alpha & : & \prod_{x,y:A} f\,x = f\,y \to q\,x = q\,y \\ \alpha_1 & : & \prod_{x:A} \alpha(x,x,1) = 1 \end{array} \right.$$

Les deux résultats énoncés précédemment restent vrais. Cependant, les niveaux d'homotopie plus élevés ne sont pas préservés. Comme pour construire le foncteur de faisceautisation, nous travaillerons à niveau de troncation fixé, il nous faut un moyen de tronquer les colimites.

On appelera n-colimite d'un diagramme D un type Q qui forme un cocone sur D et qui est universel par rapport à tous les n-types. On a alors le résultat

Lemme 1.9
Soit D un diagramme et Q une colimite de D. Alors $\|Q\|_n$ est une n-colimite de $\|D\|_n$, le diagramme où on a n-tronqué tous les types de D.

1.4 Modalités

Comme dit dans l'introduction, le but principal de notre travail est de construire, à partir d'un modèle $\mathfrak{M}$ de la théorie des types homotopique, un autre modèle $\mathfrak{M}'$ qui satisfait de nouveaux principes. On va utiliser un analogue aux modèles internes [Kun] de la théorie des ensembles, représentés ici par des modalités exactes à gauche.

Résumé modalités

1.5 Faisceaux

Résumé faisceaux

1.6 Conclusion

Commençons par un résumé de la thèse. La théorie des types homotopique est un nouveau domaine de recherche, et se compose de la théorie des types de Martin-Löf où on voit les types identité comme des homotopies, à laquelle on ajoute l'axiome d'univalence, liant les équivalences et les égalités, et les types inductifs supérieurs, permettant de construire des types avec des égalités non-triviales. Il semble exister un lien fort entre cette théorie et celle des topoi supérieurs. Plus précisément, il semble qu'on puisse voir la théorie des types homotopique comme le langage interne des $(\infty,1)$-topoi.

En théorie des types homotopiques, les types sont classifiés par leur niveau de troncation, représentant la complexité de ses espaces de boucles itérés. En particulier, si X est $(n+1)$-tronqué, alors tous les $x = y$ avec $x, y : X$ sont n-tronqués. On utilise cette propriété pour construire un opérateur sur tous les types tronqués par induction sur le niveau de troncation.

L'opérateur qu'on veut construire est une modalité. Les modalités sont une version généralisée des localisations, qui sont elles-mêmes un moyen de caractériser de façon équivalence la notion de sous-topos ; cette équivalence est toujours vraie en théorie des topos supérieurs [HTT, Section 6.2.2]. En théorie des types homotopique, une modalité est un opérateur $\bigcirc$ sur Type, muni d'unités $\eta : \prod_{X:\mathrm{Type}} X \to \bigcirc X$ satisfaisant de bonnes propriétés. Elles peuvent simplement être vue comme des monades idempotentes. En se basant sur l'équivalence entre la théorie des types et les $(\infty,1)$-topoi, on peut conjecturer que ces modalités – en fait, les modalités accessibles et exacte à gauche – induisent des sous-théories de types réflexives. Dans cette thèse, on veut décrire une théorie des types classique (*i.e.* qui satisfait le principe du tiers-exclu) comme

uns sous-théorie de la théorie des types homotopique, en utilisant une modalité. On sait déjà que c'est possible en théorie des topoi : en prenant n'importe quel topos $\mathcal{T}$, et la topologie de Lawvere-Tierney de la double négation, on peut construire le topos $\mathrm{Sh}_{\neg\neg}(\mathcal{T})$, qui est booléen.

L'idée principale de notre travail est de remarquer que les topologies de Lawvere-Tierney sur un topos, qui sont des opérateurs sur le classifiant des sous-objets, peuvent être vues en théorie des types homotopique comme des modalités sur HProp, la deuxième couche de la stratification des types. De plus, le foncteur de faisceautisation correspond à étendre cette modalité tronquée à HSet, la couche suivante. On a donc cru possible d'étendre à nouveau à la couche suivante, *etc.* pour finalement donner une modalité sur tous les types tronqués. En fait, *modulo* quelque changements dans plusieurs preuves – en particulier la preuve impliquant des *kernel pair* de flèches – et quelque sophistications dans les preuves, la méthode décrite dans le cadre des topoi peut être répétée indéfiniment pour construire une modalité sur tous les niveaux de notre stratification.

Malheureusement, la gestion actuelle des univers par Coq ne nous permet pas de formaliser complètement ce résultat ; cependant, une grosse partie est vérifiée par ordinateur. Certaines parties ne peuvent être vérifiées qu'en utilisant l'option (inconsistante) `type-in-type` de Coq, autorisant à avoir $\mathrm{Type}^i : \mathrm{Type}^i$, mais la définition complète ne peut pas être vérifiée complètement.

2 Introduction

> In anticipation of the coming of our overlords computers, we redo math as computers understand it.
>
> Andrej Bauer

Homotopy type theory is a brand new branch of mathematics and computer science, exhibiting a strong, but surprising link between the theory of ω-categories and type theory. This topic hence lives at the borderline between pure mathematics and computer science. One of the goals of researches on this topic is to use homotopy type theory as a new foundation for mathematics, replacing for example Zermelo-Frænkel set theory. Its strong links with type theory would allow mathematicians to formalize their work with a proof assistant such as Coq [The12], Agda [Nor07] or Lean [Mou+15]. Indeed, errors in mathematical research papers seem to be inevitable [Voe14], and a computer-checked proof might be more trustable than a human-checked proof. The most famous examples of computer-checked results are the *Four Colour Theorem* (to color a map such that any adjacent countries does not have the same color, four colors are sufficient) by Gonthier and Werner [Gon08] in Coq, the *Feit-Thomson Theorem* (every finite group of odd order is solvable) by Gonthier and al. [Gon+13] in Coq, the original proof of *Jordan curve theorem* (any continuous simple closed curve divides the plane into an "interior" bounded region and an "exterior" unbounded region) by Hales [Hal07] in Mizar, or the *Kepler conjecture* (the most compact ways to arrange spheres are the cubic and hexagonal close packings) by Hales and al. [Hal+15] in Isabelle and HOL Light.

One advantage of type theory over set theory is the computation property of type theory: any term is identified with its normal form. Thus, a proof assistant allows its user to simplify automatically all expressions, while a proof on paper requires all computations to be done "by hand". Set theory does not share this computation property, and is thus not convenient to use as a formal basis for a proof assistant. However, this computation property prevents us to use classical facts, such as excluded middle ; in the general case, it is not provable that a proposition is either true or false.

The main ingredient in homotopy type theory, linking mathematics and computer science, is the Curry-Howard isomorphism: one can speak equivalently about proofs or about programs, they describe the same objects *via* a

correspondence. For example, in type theory, the sequence of symbols $A \to B$ can be seen as the type of programs taking an argument of type A and producing an output of type B as well as the types of proofs that A implies B. Something implied by this correspondence is that there might exist different proofs of "A implies B", since there are probably several ways to construct an output of type B from an input of type A. This property is called *proof-relevance*, while ZFC is considered as *proof-irrelevant*: if a lemma has been proved, the way it was proved can be forgotten, as it does not matter at all.

Other than the lack of classical facts, one issue with type theory is the notion of equality. Two possibilities arise:

- an intentional, or definitional, equality ; two objects are equal if they are defined in the same way, *i.e.* if one can be exchanged with the other without changing the meaning. For example, the natural number 1 and the successor of the natural number 0 are intentionally equal. In type theory, we add some rules to this equality such as β $((\lambda\ x, f\ x)y = f\ y)$ and η.
- an extensional, or propositional, equality ; two objects are considered equal if they behave in the same way. For example, given two abstract natural numbers a and b, $a + b$ and $b + a$ are extensionally, but not intentionally equal, *i.e.* we need a proof of this.

In set theory, we usually use an extensional equality, asserting that two sets are equal if they have the same elements. In type theory, the intentional equality is a meta-theoretic notion ; only the type-checker (like Coq) can access it. It cannot be expressed in the theory itself, as it is known that extensional type theory is not decidable [Hof95] (given and term p and a type P, it might be undecidable to check if p is indeed a proof of P). The propositional equality is an internal concept defined as an inductive type

$$\mathrm{Id}(A : \mathrm{Type})(a : A) : A \to \mathrm{Type}$$

generated by only one constructor

$$\mathrm{idpath} : \mathrm{Id}_A(a,a),$$

and the type $\mathrm{Id}(A,a,b)$ will be denoted $a =_A b$ or $a = b$. This identity type is actually not satisfactory. The idea of Martin-Löf was to mimic mathematical equality, where identity types fail. Indeed, one issue with this equality is that types $a = b$ can be inhabited in several ways – at least, it is not provable that for all $p, q : a = b$, $p = q$; this property, called uniqueness of identity proofs (UIP) has been proven in [HS96] independent of intentional type theory. Another issue is that this equality is defined above all types, and does not behave well with some type constructors ; for example, functional extensionality, asserting that two functions are equal as soon as they are pointwise equal, cannot be derived.

Nevertheless, we should not throw away identity types. Around 2006, Vladimir Voevodsky, and Steve Awodey and Michael Warren [AW09] gave independently a new interpretation of identity types: types are now seen as topological spaces, inhabitants of types as points, and an element $p : \mathrm{Id}(A, a, b)$ can be read as

> In the space A, p is an homotopy (or a continuous path) between points a and b.

Under this interpretation, it seems normal not to satisfy UIP: there can be several (*i.e.* non-homotopic) paths between two points (think of a doughnut). The second issue was solved around 2009, when Vladimir Voevodsky stated the *univalence axiom*: two types are equal exactly when they are isomorphic. It surprisingly implies compatibility of identity paths with some type constructors: it implies functional extensionality, and it seems to imply that identity types of streams coincide with bisimulation [Lic].

The original project of Voevodsky [Voe10] was to give a tool so that mathematicians can check their proofs with the help of computers. Homotopy type theory seems to be a good setting for this, but it lacks the *law of excluded middle*, one of the mathematician's favourite thing:

> Any proposition is either true or false.

The main goal of this thesis is to add this principle to homotopy type theory, without losing all desired properties (decidability, canonicity, constructivity).

Note that we already know how to change an intuitionistic logic into a classical one, through the Gödel-Gentzen translation defined in figure 2.1. A

$$x^N \overset{def}{=} \neg\neg x \text{ when } x \text{ is atomic}$$

$$(\phi \wedge \psi)^N \overset{def}{=} \phi^N \wedge \psi^N \qquad (\phi \vee \psi)^N \overset{def}{=} \neg(\neg\phi^N \wedge \neg\psi^N)$$

$$(\phi \to \psi)^N \overset{def}{=} \phi^N \to \psi^N \qquad (\neg\psi)^N \overset{def}{=} \neg\phi^N$$

$$(\forall\ x,\ \phi)^N \overset{def}{=} \forall\ x,\ \psi^N \qquad (\exists\ x,\ \phi)^N \overset{def}{=} \neg\forall\ x\ \neg\phi^N$$

Figure 2.1: Gödel-Gentzen translation

soundness theorem states that a formula ϕ is classically provable if and only if ϕ^N is intuitionistically provable. Although this translation only works with the logic, the same idea can be applied to the whole type theory, as it is done in [JTS12; Jab+16]. The idea behind a translation is, from a source (complex) theory $\mathcal{S}$, to translate every term t of $\mathcal{S}$ into a term $[t]$ of a target theory $\mathcal{T}$, known to be consistent. The desired property of a translation is its soundness, *i.e.* if we can prove a soundness theorem asserting that if a term x is of type X in $\mathcal{S}$, then the translation $[x]$ of x is of type $[\![X]\!]$, where $[\![\cdot]\!]$ stands for the

translation of types. Such a statement, together with a proof that $[\![\mathbf{0}]\!]$ is not inhabited in $\mathcal{T}$, ensures that the theory $\mathcal{S}$ is consistent. One could say that a sound translation is a way to give a name in the target theory $\mathcal{T}$ to objects of $\mathcal{S}$ unknown to $\mathcal{T}$.

Set theorists will notice that this is very close to the method of forcing, invented in 1962 by Paul Cohen [Coh66]. Its historical and most famous application is the proof of the Independence of the negation of the continuum hypothesis with ZFC. In the same way, forcing can only prove relative consistency, *e.g.* ZFC+¬CH is consistent if ZFC is consistent. This method is now one of the most used ingredient of set theorists.

But we know that adapting things of set theory to type theory is not easy, as those are very different theories. Type theory is known to be closer to topos theory, and even to higher topos theory for homotopy type theory. Fortunately, Myles Tierney gave a topos-theoretic counterpart of forcing in 1972 [Tie72], through the notion of sheaves. Originally, sheaves only existed on presheaf topos (the topos of functor from a category $\mathbf{C}$ to the category **Sets**). These sheaves are called Grothendieck sheaves, and corresponds to objects F such that any function $X \to F$ can be defined equivalently on whole X or on each subsets of an open cover of X. This concept had been extended by William Lawvere and Myles Tierney, allowing objects of any topos to be sheaves. They correspond to object F such any function $X \to F$ can be defined equivalently on whole X or on any dense subobject of X. The existence of a *sheafification functor* from any topos $\mathcal{T}$ to the topos of sheaves $\mathrm{Sh}(\mathcal{T})$, left adjoint to the inclusion functor, allows to build a topos satisfying more properties than the base topos. Sheaves come from a fixed topology, which is an operator on the "logic" of the topos (the subobject classifier), idempotent, preserving $\top$ and commuting with products. Double negation is actually a topology on any topos $\mathcal{T}$, and the topos of sheaves $\mathrm{Sh}_{\neg\neg}(\mathcal{T})$ satisfies good properties: it is boolean, *i.e.* its internal logic satisfies the law of excluded middle, and the negation of the continuum hypothesis holds in this topos [MM92].

In higher topos theory, this sheafification $(\infty,1)$-functor has only been defined in [HTT] for Grothendieck sheaves, leaving the theory of Lawvere-Tierney sheaves unexplored. There is thus a double challenge in our quest for excluded middle in homotopy type theory: the first is to formalize the topos theoretic result, and see how to extend it to the setting of homotopy type theory, using if needed $(\infty,1)$-topos' concepts.

Aims of the thesis The main goal of this thesis is to give a definition of a Lawvere-Tierney sheafification functor in the setting of homotopy type theory. In order to do this, we need first to develop a theory of colimits in homotopy type theory. Then, as our definition of sheafification is done inductively on the truncation levels, we need to define a truncated version of the just defined theory of colimits, as well as a truncated version of left-exact modalities. All these developments have been computer-checked by

the proof assistant Coq; most of them are available on my Github account https://github.com/KevinQuirin.

Our deep study of modalities also leads us to define the translation of type theories associated to a (left-exact) modality, and write a Coq plugin to handle automatically that translation.

Plan of the thesis Let us describe the contents of this thesis. Chapter 3 recalls the basic definitions in homotopy type theory. It is mainly based on [HoTT][1], and serves more as a way for us to introduce notations to be consistent in the whole thesis. If the reader is not supposed to know anything about homotopy type theory before reading this thesis, this short introduction might be not enough to understand this setting, and they are strongly encouraged to take a look at [HoTT] before.

Chapter 4 introduces in its first part the theory of modalities as explained in [HoTT]. Then, we describe what we will call a *truncated modality*, which is a restricted version of modalities. Finally, we exhibit the translation of type theories induced by a left-exact modality.

In Chapter 5, we describe a basic theory of colimits over graphs, and discuss an extension defining colimits over "graphs with compositions". This chapter, in a large part, has been formalized by Simon Boulier in a library available at https://github.com/SimonBoulier/hott-colimits.

The central (and last) chapter of this thesis is Chapter 6. It describes our construction of the Lawvere-Tierney sheafification functor, which is a way to extend the not-not Gödel translation, valid only on h-propositions, to all truncated types. This result is the main contribution of the thesis. This chapter uses almost all the theory defined in previous chapters, and thus can hardly be read on its own. This section as been (almost) fully formalized, in a library available at https://github.com/KevinQuirin/sheafification.

[1] Note that, to ease the reading, some references are shortcutted by their usual names: [HoTT] is the Homotopy Type Theory book, [HoTT/Coq] is the Coq/HoTT library, [HTT] is Lurie's monograph Higher topos theory, *etc.*

3 Homotopy type theory

> Mathematics is the art of giving the same name to different objects.
>
> Henri Poincaré

This chapter is an overview of the setting we will use in the rest of the thesis, homotopy type theory. The first part describes the formal system *dependent type theory*, which is a formal basis for homotopy type theory. This description is not intended to form a complete introduction to type theory accessible by anyone, but rather to clarify the formal system we use. A neophyte willing to read this thesis should read first a complete introduction to type theory, [Hof97] for an extensive one, but [HoTT] might be more accessible, and sufficient for the thesis. As we will see, we can view homotopy type theory as in

$$\text{HoTT} = \text{MLTT} + \text{UA} + \text{HIT}.$$

UA stands for *univalence axiom*, and is introduced in the second part of this chapter. HIT is the usual abbreviation for *higher inductive types*, which is a way, extensively discussed in [HoTT], to build new type, by giving constructors for the type as well as for its identity types. General principles will not be discussed here ; we rather present usual examples of HIT in section 3.3.

Finally, the last section of this chapter presents our point of view on identity paths, leading to the terminology *homotopy* type theory. This section also introduces the notion of *truncation level*, which plays a central role in the thesis.

3.1 Dependent type theory

In Zermelo-Frankæl set theory, the most basic assertion is

$$x \in E$$

where x and E are sets. In dependent type theory, a similar judgement can be

$$a : A,$$

to be read as "a is of type A". The main difference with membership relation is that an element a has one and only one type, while we can say $x \in E$ and $x \in F$ for the same element x in set theory (it is the definition of $x \in E \cap F$).

Dependent (or Martin-Lóf, due to its inventor [Mar98]) type theory is based on the Curry-Howard correspondance [How80], or "propositions as types" principle. Indeed, we do not need to make a difference between types an propositions. Hence $a : A$ will be read "a is of type A" when A is seen as a type, and "a is a proof of A" when A is seen as a proposition. In the rest of this section 3.1, we present the different types used to build dependent type theory. It is not intended to be complete ; for example we sometimes only give the non-dependent elimination rules to ease the reading. The reader can refer to [NPS01] or [HoTT] for a more detailed introduction.

3.1.1 Universes

In dependent type theory, types are also terms of a universe Type. Of course, we want the universe Type to be itself a type, and the Russel's paradox[1] is close here. We solve the problem by using a cumulative hierarchy of universes

$$\mathrm{Type}^0 : \mathrm{Type}^1 : \mathrm{Type}^2 : \cdots$$

where every universe Type^i is of type Type^{i+1}. Cumulativity means that if $A : \mathrm{Type}^k$ and $k < m$, then $A : \mathrm{Type}^m$. The handling of universes level can be done automatically[2] by proof assistants such as Coq, and we will thus use only a "type of type" Type, to be understood in a polymorphic way.

3.1.2 Empty and Unit types

The first two types we will see are the Empty type (denoted $\mathbf{0}$) and the Unit type (denoted $\mathbf{1}$). These are respectively the types with zero and one elements (named $\star$). Those two types are dual to each other:

- having a term of type $\mathbf{0}$ in the context allows to prove anything, while having a term of type $\mathbf{1}$ in the context is useless
- dually, giving a term of type $\mathbf{1}$ is trivial, while giving a term of type $\mathbf{0}$ is impossible (if the theory if consistent).

Here are the (non-trivial) introduction and elimition rules for these types:

$$\frac{\Gamma \vdash x : \mathbf{0} \qquad X : \mathrm{Type}}{\Gamma \vdash X}\ \mathbf{0}\text{-ELIM} \qquad \frac{}{\Gamma \vdash \star : \mathbf{1}}\ \mathbf{1}\text{-INTRO}$$

Under the propositions-as-types principle, $\mathbf{0}$ is the type always false, and $\mathbf{1}$ the type always true. With a categorical point of view, $\mathbf{0}$ is an initial object and $\mathbf{1}$ is a terminal object.

[1] In type theory, the argument is harder to prove, and is called Girard's paradox.

[2] We will see in chapters 4 and 6 that we sometimes need to "help" Coq

3.1.3 Coproduct

The coproduct of A and B, noted $A + B$, is seen as the disjoint sum of A and B. It is described by the inductive type generated by

$$\left|\begin{array}{lcl} \text{inl} & : & A \to A + B \\ \text{inr} & : & B \to A + B \end{array}\right.$$

The introduction and elimination rules of coproduct are:

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash \text{inl}\, a : A + B}\ +\text{-INTRO}_L \qquad \frac{\Gamma \vdash b : B}{\Gamma \vdash \text{inr}\, b : A + B}\ +\text{-INTRO}_R$$

$$\frac{\Gamma \vdash p : A + B \qquad \Gamma, x : A \vdash c_A : C \qquad \Gamma, x : B \vdash c_B : C}{\Gamma, \vdash \text{sum\_rect}(p, c_A, c_B) : C}\ +\text{-ELIM}$$

Under the propositions-as-types principle, $A + B$ is seen as the disjunction of A and B.

3.1.4 Dependent product

One of the things both mathematicians and computer scientists love to do is to define functions. If A and B are types, one can consider the type of functions from A to B, taking an inhabitant of type A (called the source type) and giving an inhabitant of type B (the target type). What is new in dependent type theory is that the target type is allowed to depend on the argument of the function.

EXAMPLE

For example, one can consider a function taking a natural number n and giving a natural number greater than n. The target source depends indeed of n.

The type of dependent functions with source type A and target type Bx is called "dependent product over B" (or "pi-type over B"), and will be denoted $\prod_{x:A} Bx$ or $(x : A) \to (Ba)$. When B does not depend on A, we just say "arrow type" and note it $A \to B$.

The introduction and elimination rules of dependent products are:

$$\frac{\Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda\,(x : A), b : \prod_{x:A} B}\ \prod\text{-INTRO} \qquad \frac{\Gamma \vdash f : \prod_{x:A} B \qquad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B[a/x]}\ \prod\text{-ELIM}$$

Under the propositions-as-types, type of non-dependent functions $A \to B$ is seen as implication $A \Rightarrow B$, and type of dependent functions $\prod_{x:A} Bx$ is seen as universally quantified formulas $\forall x, Bx$.

3.1.5 Dependent sum

If A and B are two types, we would like to define the type of pairs (a, b), where $a : A$ and $b : B$. The resulting type is called the product of A and B, noted $A \times B$.

As for functions, dependent type theory allows the second type to depend on the first type. Thus, the type of pairs where the first element x is in type A and the second element y is in type $B\,x$ is called "dependent sum over B" (or sigma-type over B), noted $\sum_{a:A} B\,a$.

The introduction and elimination rules are:

$$\frac{\Gamma, x : A \vdash B : \text{Type} \qquad \Gamma \vdash x : A \qquad \Gamma \vdash b : B[a/x]}{\Gamma \vdash (a,b) : \sum_{x:a} B}\ \sum\text{-INTRO}$$

$$\frac{\Gamma \vdash p : \sum_{x:A} B}{\Gamma \vdash \pi_1 p : A}\ \sum\text{-INTRO}_1 \qquad \frac{\Gamma \vdash p : \sum_{x:A} B}{\Gamma \vdash \pi_2 p : B[\pi_1 p/x]}\ \sum\text{-INTRO}_2$$

The projections π_1 and π_2 will sometimes be noted, when applied to a term u, u_1 and u_2.

NOTA

We note that the terminology might be confusing: the dependent generalization of products are dependent sums, while dependent products are generalization of functions.

Under the propositions-as-types principle, $A \times B$ is seen as the conjonction $A \wedge B$, and $\sum_{x:A} B\,a$ is seen as the existentially quantified formula $\exists x\, B\, x$.

If A and B are types, $f : A \to B$ and $b : B$, the particular dependent sum $\sum_{a:A} f\,a = b$ is called the fiber of f over b, noted $\text{fib}_f(b)$. It describes the inverse image of b by f.

3.1.6 Inductive types

Dependent type theory actually allows us to define any inductive type. An inductive type is a type defined only by its introduction rules, in a free way. Its elimination rules are automatically determined. We have already seen examples of inductive types: Empty, Unit, Coproduct.

The most basic example of inductive types might be the type $\mathbb{N}$ of naturals. Its introduction rules are

$$\frac{}{0 : \mathbb{N}}\ \mathbb{N}\text{-INTRO}_0 \qquad \frac{\Gamma \vdash n : \mathbb{N}}{\Gamma \vdash S\,n : \mathbb{N}}\ \mathbb{N}\text{-INTRO}_S$$

We also say that its *constructors* are $0 : \mathbb{N}$ and $S : \mathbb{N} \to \mathbb{N}$: $\mathbb{N}$ is thus the free monoid generated by 0 and S. The elimination rule for $\mathbb{N}$ is the famous induction principle

$$\frac{\begin{array}{ll}\Gamma, x : \mathbb{N} \vdash P : \text{Type} & \Gamma \vdash c_0 : C[0/x] \\ \Gamma \vdash n : \mathbb{N} & \Gamma, n : \mathbb{N}, y : C \vdash c_S : C[S\,n/x]\end{array}}{\Gamma \vdash \text{nat\_ind}(C, c_0, c_S(n,y), n) : C[n/x]}\ \mathbb{N}\text{-ELIM}$$

This elimination rule allows us to define basic operators on natural numbers: addition, multiplication, order, *etc.*

For a reader interested by the theory of general inductive types, we refer him to [AGS12].

3.1.7 Paths type

One of the most powerful tool in dependent type theory might be the identity types. They allow us to talk about propositional equality between inhabitants of a type. The identity type over A will be noted $a =_A b$ or $a = b$ if A can be inferred from context . What is great about identity type over A is that its definition does not depend on the type A. If $a, b : A$, $a =_A b$ is defined as the inductive type whose only constructor is

$$\text{idpath} : \prod_{a:A} a =_A a$$

(idpath_x will sometimes be just noted 1_x, or even 1).

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash \text{idpath}_a : a =_A a} =\text{-INTRO}$$

$$\frac{\begin{array}{cc}\Gamma \vdash a, b : A & \Gamma, x : A, y : A, p : x =_A y \vdash P : \text{Type} \\ \Gamma \vdash q : a =_A b & \Gamma, z : A \vdash w : P[z, z, 1/x, y, p]\end{array}}{\Gamma \vdash \text{path\_ind}_P^q(w) : P[a, b, q/x, y, p]} =\text{-ELIM}$$

The elimination principle should be looked closer. Lets simplify to understand how it works: suppose that the type family P depends only on two objects $x, y : A$. Then if C is reflexive, *i.e.* if $C(x,x)$ is always inhabited, then whenever $x = y$, $C(x,y)$ is inhabited too.

Path-induction, also known as "J principle", allow us to state the following:

Lemma 3.1 : [HoTT, Lemma 2.3.1]
Let A : Type *and* $P : A \to \text{Type}$ *a type family over* A. *Then if* $p : x =_A y$, *there is a function* $\text{transport}_P^p : P\,x \to P\,y$.

Proof. Let's do this proof to use path-induction for the first time. Let Q be the type family indexed by $x, y : A$

$$Q(x,y) := P\,x \to P\,y.$$

Then by path-induction, it suffices to define $Q(x,x)$. The latter is clearly inhabited by $\text{idmap}_{P\,x}$. □

Path-induction might be the most powerful tool we can use. For example, most of the lemmas we will see in section 3.4 can be proved using path-induction.

3.1.8 Summary

We can summarize the situation in the following array:

Name	Notation	Proposition-as-types
Empty	$\mathbf{0}$	$\bot$
Unit	$\mathbf{1}$	$\top$
Coproduct, sum	$A + B$	$A \vee B$
Function	$A \to B$	$A \Rightarrow B$
Dependent function, pi-type	$\prod_{x:A} B\,x$	$\forall x, B\,x$
Product	$A \times B$	$A \wedge B$
Dependent sum, sigma-type	$\sum_{x:A} B\,x$	$\exists x, B\,x$

3.2 Identity types and Univalence axiom

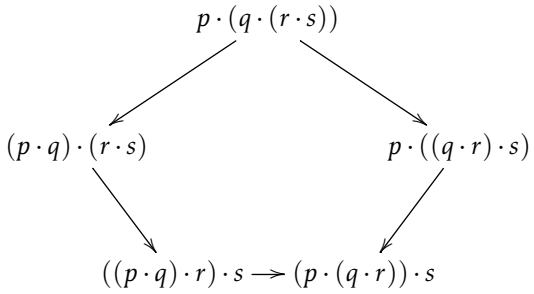
Identity types are very useful to assert propositional equalities between object. Note that it does not characterize *judgemental* equality, which we consider here as belonging to the meta-theory (some theories, as Voevodsky's Homotopy Type System [HTS], implemented in proof assistant Andromeda [m31]). One can prove that identity types give each type a structure of a ω-groupoid, *i.e.* the structure of a ω-category where all arrows are invertible. For example, identity types over a type A satisfies:

- Reflexivity: for all $x : A$,
$$1_x : x = x$$
- Transitivity: for all $x, y, z : A$, $p : x = y$ and $q : y = z$,
$$p \cdot q : x = z$$
- Symmetry: for all $x, y : A$ and $p : x = y$, $p^{-1} : y = x$
- Reflexivity and symmetry behave well together: for all $x, y : A$ and $p : x = y$,
$$p \cdot p^{-1} = 1_x \text{ and } p^{-1} \cdot p = 1_y$$
Note that these paths are respectively in types $x = x$ and $y = y$.
- Reflexivity and associativity behave well together: for all $x, y : A$ and $p : x = y$,
$$p \cdot 1 = p \text{ and } 1 \cdot p = p$$

- Associativity: for all $w,x,y,z : A$ and $p : w = x$, $q : x = y$, $r : y = z$,

$$p \cdot (q \cdot r) = (p \cdot q) \cdot r$$

- Maclane pentagon: for all $v,w,x,y,z : A$, $p : v = w$, $q : w = x$, $r : x = y$ and $s : y = z$, the following diagram involving associativities commutes



- The coherences goes on and on.

A thing one can notice here is that we work in a proof-relevant setting. The types $x = y$ are just types, and thus can be inhabited by different objects. It might sound unfamiliar for mathematicians, but we will give in section 3.4 an interpretation of identity types justifying proof-relevance.

Identity types also have a good behavior with respect to function. Let A, B : Type and $f : A \to B$; then

- For all $x, y : A$, there is a map $\mathrm{ap}_f : x = y \to f(x) = f(y)$
- It is coherent with inverse: $\mathrm{ap}_f p^{-1} = (\mathrm{ap}_f p)^{-1}$
- It actually is coherent with the whole ω-groupoid structure. We refer to [HoTT] for more detailed results.

The fact that the definition of identity types does not depend on the type might seem strange, but it actually allows to catch the equality one would want ; for example, we can characterize the equalities between dependent pairs:

Lemma 3.2 : Paths in dependent sums
Let A : Type and $B : A \to$ Type. Then, for any $u, v : \sum_{x:A} B\,a$, we have a term

$$\mathrm{path}_\Sigma : \sum_{p:u_1 =_A v_1} \mathrm{transport}_p^p\, u_2 = v_2 \to u = v.$$

This characterization goes even further, as we can prove that path_Σ is an equivalence. Equivalences are very important objects in homotopy type theory, and are defined as

Definition 3.3 : Equivalence
Let A, B : Type *and* $f : A \to B$. *Wa say that* f *is an equivalence, noted* $\mathrm{IsEquiv}(f)$, *if there are:*

- *a map* $g : B \to A$, *called the inverse of* f
- *a term* $\mathrm{retr}_f : \prod_{x:A} g(f(x)) = x$, *called the retraction of the equivalence*
- *a term* $\mathrm{sect}_f : \prod_{x:A} f(g(x)) = x$, *called the section of the equivalence*
- *a term* $\mathrm{adj}_f : \prod_{x:A} \mathrm{ap}_f \mathrm{retr}_f x = \mathrm{sect}_f(f x)$, *called the adjunction of the equivalence*

We say that A *and* B *are equivalent, noted* $A \simeq B$, *if there is a function* $f : A \to B$ *which is an equivalence.*

Hence, path_Σ allows us to prove equalities in a dependent sum $\sum_{x:A} B a$, but its inverse allows us to prove, from an equality in the dependent sum, equalities in A and in $B a$.

Unfortunately, identity types does not allow to characterize equalities for all types. One example is paths in dependent products. In usual mathematics, one would like pointwise equality to be a sufficient (actually, a necessary and sufficient) property to state equality between function. In type theory, that would be phrased

$$\prod_{x:A} f x = g x \to f = g. \tag{3.1}$$

But one can prove (see [Str93] for a semantical proof) that this property, called *functional extensionality* cannot be proved from rules of dependent type theory. All we can say is that the backward function of 3.1 can be defined, called happly:

$$\mathrm{happly} : f = g \to \prod_{x:A} f x = g x.$$

Functional extensionality will thus be stated as the axiom:

Axiom 3.4 : Functional extensionality
For any A : Type *and* $B : A \to$ Type, *the arrow* happly *is an equivalence.*

In a way, that could solve our problem. But dependent products are not the only types for which we cannot characterize identity types; the same problem arises with identity types of the universe. An answer was proposed first by Martin Hofmann and Thomas Streicher in 1996 in [HS96], and later (around 2005) by Vladimir Voevodsky: the univalence axiom. As for functional extensionality, it asserts that a certain arrow is an equivalence.

Axiom 3.5 : Univalence axiom
For any types A, B : Type*, the function*

$$\text{idtoequiv} : A = B \to A \simeq B$$

is an equivalence.

What is great about univalence axiom is that is seems to imply all other characterization of identity types. For example:

Lemma 3.6 : [BL11][Lic]
Univalence axiom implies functional extensionality.

Another example the reader can think of is the type of streams (and more generaly coindutive) extensionality: we want two streams (infinite list) to be equal when they are pointwise equal. It seems that univalence axiom implies it [Lic].

The main issue with univalence is that it clearly oblige us to work in a proof-relevant setting ; indeed, there are for examples two distinct proofs of $\mathbf{1} + \mathbf{1} = \mathbf{1} + \mathbf{1}$: the identity, and the function swapping left and right.

3.3 Higher Inductives Types

As seen in section 3.1.6, one can define a type by giving only its constructors ; but we also saw in section 3.2 that our setting is proof-relevant. It means that a type is characterized not only by its objects, but also by its identity types between these objects, and identity types of these identity types, *etc.* Hence, we would like to be able, like for inductive types, to define a type by giving constructors for the objects, and constructors for the identity types, *etc.* Then, the constructed type is the type freely generated by the constructors, and whose identity types are freely generated by the constructors, *etc.* Unlike inductive types, it is not known if elimination rules of higher inductive type can easily (and automatically) be infered from the constructors. At the moment, we prefer to express them explicitly. As a general theory of hogher inductive types would be very hard, we will only give some fundamental examples.

3.3.1 The circle

The most basic example of higher inductive type is the circle $\mathbb{S}^1$. The circle consists of just a point, with a non-trivial path above this point. It is defined as the higher inductive type generated by

$$\left| \begin{array}{lcl} \text{base} & : & \mathbb{S}^1 \\ \text{loop} & : & \text{base} = \text{base} \end{array} \right.$$

It can be pictured as

As said, we give the elimination principle of the circle. We start by the non-dependent eliminator.

Lemma 3.7 : Non-dependent elimination of $\mathbb{S}^1$
Let P be a type. If $b : P$ and $p : b = b$, then there is a map

$$\mathbb{S}^1_{\text{rec}} : \mathbb{S}^1 \to P$$

such that $\mathbb{S}^1_{\text{rec}}(\text{base}) \equiv b$ (judgementally) and $\text{ap}_{\mathbb{S}^1_{\text{rec}}}(\text{loop}) = p$ (propositionally).

It says that if in P you can find a "copy" of $\mathbb{S}^1$, then there is a "good" function $\mathbb{S}^1 \to P$. The dependent eliminator is a bit more complicated, but still understandable.

Lemma 3.8 : Dependent elimination of $\mathbb{S}^1$
Let $P : \mathbb{S}^1 \to \text{Type}$ be a type family over $\mathbb{S}^1$. If $b : P(\text{base})$ and $p : \text{transport}^{\text{loop}}_P(b) = b$, then there is a term

$$\mathbb{S}^1_{\text{ind}} : \prod_{x:\mathbb{S}^1} P\,x$$

such that $\mathbb{S}^1_{\text{ind}}(\text{base}) \equiv b$ and $\text{ap}_{\mathbb{S}^1_{\text{ind}}}(\text{loop}) = p$.

The most famous result about the circle is the computation of its homotopy group $\pi_1(\mathbb{S}^1)$.

Proposition 3.9 : Fundamental group of $\mathbb{S}^1$ [LS13]
There is an equivalence

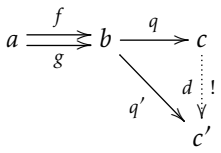
$$(\text{base} = \text{base}) \simeq \mathbb{Z}$$

where $\mathbb{Z} \overset{def}{=} \mathbb{N} + \mathbb{N} + \mathbf{1}$.

3.3.2 Coequalizers

Another example of higher inductive type is the computation of coequalizers. In category theory, a coequalizer of two objects a and b with arrow $f, g \in \text{Hom}(a,b)$ is a $c \in \text{Obj}$ together with an arrow $q \in \text{Hom}(b,c)$ such

that $a \underset{g}{\overset{f}{\rightrightarrows}} b \xrightarrow{q} c$ commutes, and which is universal with respect to this property, in the sense that for any other object $c' \in \mathrm{Obj}$ with $q' \in \mathrm{Hom}(b,c')$, there is an unique arrow $d : \mathrm{Hom}(c,c')$. It might be seen as the diagram



In homotopy type theory, the coequalizer of two functions $f,g : A \to B$ is defined as the higher inductive type $\mathrm{Coeq}^{f,g}$ generated by

$$\left| \begin{array}{lcl} q & : & B \to \mathrm{Coeq}^{f,g} \\ \alpha & : & \prod_{x:A} q \circ f(x) = q \circ g(x) \end{array} \right.$$

Its elimination principles are:

Lemma 3.10 : Elimination of coequalizer
Let A, B, f, g be as above.

- *Let $P : Type$. If there are terms $q' : B \to P$ and $\alpha' : \prod_{x:A} q' \circ f(x) = q' \circ g(x)$, then there is a map*

$$\mathrm{Coeq}^{f,g}_{\mathrm{rec}} : \mathrm{Coeq}^{f,g} \to P$$

such that for all $b : B$, $\mathrm{Coeq}^{f,g}_{\mathrm{rec}}(q\,b) \equiv q'\,b$ and $\mathrm{ap}_{\mathrm{Coeq}^{f,g}_{\mathrm{rec}}}(\alpha\,b) = \alpha'\,b$.

- *Let $P : \mathrm{Coeq}^{f,g} \to \mathrm{Type}$. If there are terms $q' : \prod_{b:B} P(q\,b)$ and $\alpha' : \prod_{b:B} \mathrm{transport}^{\alpha\,b}_{P}(q' \circ f(b)) = q' \circ g(b)$, then there is a dependent map*

$$\mathrm{Coeq}^{f,g}_{\mathrm{ind}} : \prod_{x:\mathrm{Coeq}^{f,g}} P\,x$$

such that for all $b : B$, $\mathrm{Coeq}^{f,g}_{\mathrm{ind}}(q\,b) \equiv q'\,b$ and $\mathrm{ap}_{\mathrm{Coeq}^{f,g}_{\mathrm{ind}}}(\alpha\,b) = \alpha'\,b$.

It satisfies the desired universal property in the following sense:

- If Q is a type and $\varphi : \mathrm{Coeq}^{f,g} \to Q$, then one can define a map $\psi : B \to Q$ such that for all $b : B$, $q \circ f(b) = q \circ g(b)$.
- One can prove that the map $\phi \mapsto \psi$ is an equivalence: from any other type such that the diagram commutes, there is an unique arrow $\mathrm{Coeq}^{f,g} \to Q$.

Some results about coequalizers in homotopy type theory can be found in the form of Coq code in the HoTT/Coq library [HoTT/Coq].

3.3.3 Suspension

There is a way to see any type A as the identity type of another type: the suspension of A, noted ΣA. It is the higher inductive type generated by

$$\left|\begin{array}{lcl} N & : & \Sigma A \\ S & : & \Sigma A \\ \text{merid} & : & A \to (N = S) \end{array}\right.$$

Its elimination principle are:

Lemma 3.11 : Elimination of suspension
Let A be a type.

- *Let P : Type. If there are $n, s : B$ and $m : A \to (n = s)$, then there is a map*

$$\Sigma^A_{\text{rec}} : \Sigma A \to B$$

such that $\Sigma^A_{\text{rec}}(N) \equiv n$, $\Sigma^A_{\text{rec}}(S) \equiv s$ and for all $a : A$, $\text{ap}_{\Sigma^A_{\text{rec}}}(\text{merid}(a)) = m(a)$

- *Let $P : \Sigma A \to$ Type. If there are $n : P(N)$, $s : P(S)$ and $m : \prod_{a:A} \text{transport}^{\text{merid}(a)}_P n = s$, then there is a dependent map*

$$\Sigma^A_{\text{ind}} : \prod_{a:A} P\,a$$

such that $\Sigma^A_{\text{ind}}(S) \equiv s$, $\Sigma^A_{\text{ind}}(N) \equiv n$ and $\text{ap}_{\Sigma^A_{\text{ind}}}(\text{merid}(a)) = m(a)$.

The first thing we can notice is that $\Sigma(\mathbf{1} + \mathbf{1}) \simeq \mathbb{S}^1$ [HoTT, Lemma 6.5.1]. Actually, we can define the sequence of n-spheres inductively by

$$\begin{array}{ll} \mathbb{S}^0 & \overset{def}{=} \mathbf{1} + \mathbf{1} \\ \mathbb{S}^{n+1} & \overset{def}{=} \Sigma\mathbb{S}^n \end{array}$$

3.4 Introduction to homotopy type theory

It is now time to really introduce homotopy type theory: it is the sum of sections 3.1, 3.2 and 3.3. In other words, homotopy type theory is just dependent type theory, augmented with univalence axiom and higher inductive types. We promised in section 3.2 an interpretation of identity types. In homotopy type theory, we view types as topological spaces and inhabitants of types as points of these spaces. Then, if $x, y : A$, the identity type $x = y$ is viewed as the topological space of paths (or continuous maps $f : [0,1] \to A$ such that $f(0) = x$ and $f(1) = y$) between points x and y in A. As said, $x = y$ is itself a topological space, thus we can iterate the analogy. A path $r : p = q$ where $p, q : x = y$ are themselves paths is called a *homotopy* ; a path between paths

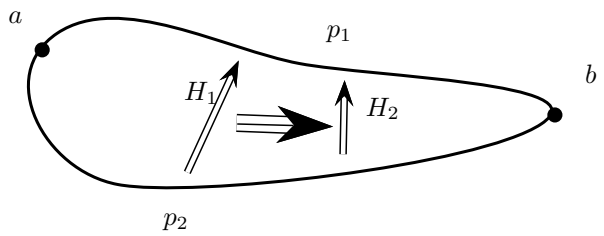


Figure 3.1: Paths: a and b are points, p_1 and p_2 paths, H_1 and H_2 homotopies.

between paths is called a 2-*homotopy*, *etc.* The situation can be depicted as in figure 3.1. Indeed, that interpretation allows to explain all properties of identity types.

- The equality 1_x is just the constant path

$$1_x : \begin{array}{ccc} [0,1] & \longrightarrow & A \\ t & \longmapsto & x \end{array}$$

- The inverse $\bullet^{-1}$ is the function changing a path f into

$$g : \begin{array}{ccc} [0,1] & \longrightarrow & A \\ t & \longmapsto & f(1-t) \end{array}$$

- The concatenation $\bullet \cdot \bullet$ is the function changing paths f and g into

$$h : \begin{array}{ccc} [0,1] & \longrightarrow & A \\ t & \longmapsto & \begin{cases} f(2t) & \text{if } t \in [0,1/2] \\ g(2t-1) & \text{if } t \in [1/2,1] \end{cases} \end{array}$$

- A path between paths f and g between points x and y is a continuous function

$$H : \begin{array}{ccc} [0,1] & \longrightarrow & A \\ (u,v) & \longmapsto & H(u,v) \end{array}$$

  such that $H(t,0) = f(t)$, $H(t,1) = g(t)$, $H(0,s) = x$ and $H(1,s) = y$. A homotopy is then a continuous deformation of f into g, fixing the ending points x and y.

The table 3.1 summarize the three points of view (type theoretic, groupoidal, homotopy theoretic). In the rest of the thesis, we will use any of the following name (*e.g.* we will talk about "points of a type", "path betwenn inhabitants", *etc.*).

What homotopy theorists love to do is to compute fundamental groups (*i.e.* the group of paths between two points) of different spaces ([WX10] [Hut11]). We can then categorized spaces with the level at which their homotopy groups become trivial. Voevodsky has realized that this notion admits a compact inductive definition internal to type theory, given by

Type theory	Homotopy theory	ω-groupoid
Type	Topological space	ω-groupoid
Inhabitant	Point	Object
Equality	Path	Morphism
idpath_x	Constant path	Identity morphism
$\bullet^{-1}$	Inverse path	Inverse morphism
$\bullet \cdot \bullet$	Concatenation of paths	Composition of morphisms
Equality between equalities	Homotopy	2-morphism

Table 3.1: Three points of view about HoTT

Definition 3.12 : Truncated types
Is-n-type *is defined by induction on* $n \geqslant -2$*:*

- Is-(-2)-type(X) *if* X *is a contractible type,* i.e. X *is inhabited by* $c : X$*, and every other point in* X *is connected to* c*.*
- Is-$(n+1)$-type$(X) \overset{def}{=} \prod_{x,y:X}$ Is-n-type$(x = y)$.

Then, $\text{Type}_n \overset{def}{=} \sum_{X:\text{Type}}$ Is-n-type(X).

For the first values of n, there are different names: (-2)-truncated types are called *contractible types*, (-1)-truncated types are called *h-propositions*, 0-truncated types are called *h-sets*. Following this, Is-(-2)-type is just Contr, Is-(-1)-type is just IsHProp and Type_{-1} is HProp, and Is-0-type is just IsHSet and Type_0 is HSet. An explanation of this terminology might be helpful. Contractible types are types, inhabited by a center c, with paths between any point and c. A kind of magic thing about contractible types is the lemma

Lemma 3.13
If A : Type *is contractible, then for any* $x, y : A$*, the type* $x = y$ *is contractible.*

Inductively, it means that paths types of any level over a contractible type is contractible. It can be seen as the fact that a contractible type contains only one point c, for which there is only one path $c = c$, *etc.* The canonical example of contractible type is $\mathbf{1}$, and actually, any contractible type is equivalent to $\mathbf{1}$. Then, h-propositions are types where any two points are connected by a path. The only difference with contractible types is that we allow the type not to be inhabited. H-propositions are then proof-irrelevant types, in the sense that under the propositions-as-types principle, any points x and y in an HProp are equal, with a unique equality, which is thus irrelevant.

Lemma 3.14 : Example of h-propositions
The following types are h-propositions:

- $\mathbf{0}, \mathbf{1}$
- $\mathrm{IsEquiv}(f)$ *for any* $A, B : \mathrm{Type}$ *and* $f : A \to B$.
- $\text{Is-}n\text{-type}(A)$ *for any type* $A : \mathrm{Type}$ *and truncation index* n.

Now, by definition, h-sets are types for which identity types are in HProp. The world of h-sets can thus be seen as the "usual mathematical" world, as it satisfies proof-irrelevance.

The n-truncated types are stable under a lot of operations, as:

Lemma 3.15
Let $n \geqslant -2$ *be a truncation index. Then*

- *If* $A : \mathrm{Type}_n$ *and* $x, y : A$, *then* $\text{Is-}n\text{-type}(x = y)$.
- *If* $A : \mathrm{Type}_n$ *and* $B : A \to \mathrm{Type}_n$, *then* $\text{Is-}n\text{-type}\,(\sum_{x:A} B\,a)$.
- *If* $A : \mathrm{Type}$ *and* $B : A \to \mathrm{Type}_n$, *then* $\text{Is-}n\text{-type}\,(\prod_{x:A} B\,a)$. *Note that the source type needs not to be truncated.*
- *If* $X : \mathrm{Type}_n$, $Y : \mathrm{Type}$ *and* $X \simeq Y$, *then* $\text{Is-}n\text{-type}(Y)$.
- *If* $X : \mathrm{Type}_n$, *then* $\text{Is-}(n+1)\text{-type}(X)$.
- *We have* $\text{Is-}(n+1)\text{-type}\,\mathrm{Type}_n$.

Nota

Some types are not n-truncated for any n [HoTT, Example 8.8.6] ; it is highly suspected for example that the sphere $\mathbb{S}^2$ is one of these ∞-truncated types (it is at least true in homotopy theory).

This stratification of types induces a stratification of functions. A function $f : A \to B$ is said to be n-truncated if all its homotopy fibers $\mathrm{fib}_f(b)$ are n-truncated. If $B : \mathrm{Type}$, a type A together with a n-truncated map $f : A \to B$ will be called a n-subobject of B.

As in [RS13], we can define a sequence of subobject classifiers, namely, Type_n classifies n-subobjects of a type B, in the sens that there is an equivalence

$$\Xi : \sum_{A:\mathrm{Type}} \sum_{f:A\to B} \prod_{b:B} \text{Is-}n\text{-type}\ \mathrm{fib}_f(b) \xrightarrow{\sim} (B \to \mathrm{Type}_n)$$

such that the diagram

$$\begin{array}{ccc} A & \xrightarrow{t_f} & \mathrm{Type}_n^\bullet \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \pi_1} \\ B & \xrightarrow[\chi_f]{} & \mathrm{Type}_n \end{array}$$

is a pullback for any f with n-truncated homotopy fibers where $\mathrm{Type}_n^\bullet \overset{def}{=} \sum_{A:\mathrm{Type}} A$ is the universe of pointed n-truncated types and

$$t_f = \lambda a,\ (\mathrm{fib}_f(f(a)),(a,\mathrm{idpath})).$$

We will use the equivalence Ξ to define n-subobjects of a type B either as a n-truncated map $A \to B$, either as a characteristic map $\chi : B \to \mathrm{Type}_n$.

3.4.1 Truncations

We present here a way to change any type into a n-truncated type, using truncations. The interested reader can read Nicolai Kraus' PhD thesis [Kra15] consecrated to truncation levels in HoTT.

Let $n \geqslant -1$ be a truncation index. The n-truncation of a type A is the higher inductive type $\|A\|_n$ generated by

$$\left|\begin{array}{lcl} \mathrm{tr}_n & : & A \to \|A\|_n \\ \alpha_{\mathrm{tr}}^n & : & \text{Is-}n\text{-type}(\|A\|_n) \end{array}\right.$$

If $a : A$, $\mathrm{tr}_n(a)$ will be noted $|a|_n$. The elimination principles are:

Lemma 3.16 : Elimination principle of truncations
Let A : Type and $n \geqslant -1$ be a truncation index.

- *If P : Type such that* Is-n-type(P) *and $f : A \to P$, then there is a map*

$$|f|_n : \|A\|_n \to P$$

such that for all $a : A$, $|f|_n(|a|_n) \equiv f(a)$.

- *If $P : \|A\|_n \to$ Type such that $\prod_{x:\|A\|_n}$* Is-n-type$(P\,x)$ *and $f : \prod_{a:A} P(|a|_n)$, then there is a dependent map*

$$|f|_n : \prod_{x:\|A\|_n} P\,x$$

such that for all $a : A$, $|f|_n(|a|_n) \equiv f(a)$.

Basically, this induction principle says that $\|A\|_n$ has contains as much data about A than A itself, but it can only be used to define a n-truncated type. It can be expressed as the following universal property:

Lemma 3.17 : Universal property of truncations
Let $A : \mathrm{Type}$ *and* $P : \mathrm{Type}_n$. *Then the map*

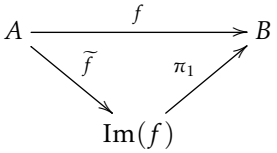
$$\mathrm{precompose}_{\mathrm{tr}_n} : \begin{array}{ccc} A \to P & \longrightarrow & \|A\|_n \to P \\ f & \longmapsto & f \circ \mathrm{tr}_n \end{array}$$

is an equivalence.

We will see in chapter 4 that all $(\text{Is-}n\text{-type}, \| \bullet \|_n)$ define *modalities*. In particular, every truncation index n yields a factorization system:

Proposition 3.18
Let $n \geqslant -1$ *a truncation index. Then any map* $f : A \to B$ *can be factored as a* n*-connected map followed by a* n*-truncated map, where a map is* n*-connected if for all* $b : B$, $\|\mathrm{fib}_f(b)\|_n$ *is contractible.*

In the rest of this thesis, we will only use this result with $n = -1$. In that case, (-1)-truncated functions are called *embeddings*, and (-1)-connected functions are called *surjections*. The factorization of a map $f : A \to B$ is given by



$$\mathrm{Im}(f) \overset{def}{=} \sum_{b:B} \left\| \sum_{a:A} f\, a = b \right\|_{-1}$$
$$\widetilde{f} \overset{def}{=} \lambda a : A, \left(f\, a, \left| (a, 1_b) \right|_{-1} \right)$$

4 Higher modalities

> There is no one fundamental logical notion of *necessity*, not consequently of *possibility*. If this conclusion is valid, the subject of modality ought to be vanished from logic, since propositions are simply true or false.
>
> Bertrand Russel

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* [Gol03] and *modal type theories* [Mog91]. At the time we write this thesis, the only reference for modalities is [HoTT, Section 7.7]. Most of the results presented here either are taken from the book, either are paper versions of results formalized in [HoTT/Coq, Folder `Modalities`].

Egbert Rijke, Mike Shulman and Bas Spitters are writing a paper [RSS] extending greatly what is said in the HoTT book, and probably in this thesis.

4.1 Modalities

Definition 4.1
A left exact modality is the data of

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

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

(vi) for any A : 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 $\bigcirc_{\text{rec}} : (A \to B) \to (\bigcirc A \to B)$, and its computation rule $\bigcirc^{\beta}_{\text{rec}} : \prod_{f:A\to B} \prod_{x:A} \bigcirc_{\text{rec}}(f)(\eta_A x) = fx$.

If $\bigcirc$ is a modality, the type of modal types will be denoted $\text{Type}^{\bigcirc}$. 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 4.2 : 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 4.3
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)* $\text{Type}^{\bigcirc}$ *is closed under dependent products,* i.e. *$\prod_{x:A} Bx$ is modal as soon as all Bx are modal.*

•*(R) For any types A and B, the map*

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

is an equivalence.

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

- •*(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 $A, B : \mathrm{Type}$ are modal, then so are* $\mathrm{Is}\text{-}n\text{-}\mathrm{type}\, A$, *$A \simeq B$ and* $\mathrm{IsEquiv}\, f$ *for all $f : A \to B$.*
- •*(L) If $A : \mathrm{Type}_n$, then $\bigcirc A : \mathrm{Type}_n$.*
- •*(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 4.4
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 4.5
Let A, B : 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 4.6
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 $T : \mathrm{HProp}$. 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 4.7
For any modality, Type_{-1} *is closed under the reflector* $\bigcirc$*,* i.e.

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

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

Lemma 4.8
For any left-exact modality, all Type_p *are closed under the reflector* $\bigcirc$*,* i.e.

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

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.

4.2 Examples of modalities

4.2.1 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 $\eta_A \overset{def}{=} \mathrm{idmap}$. 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.

4.2.2 Truncations

The first class of non-trivial examples might be the *truncations* modalities, seen in 3.4.1.

4.2.3 Double negation modality

Proposition 4.9

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 6 is to extend this modality into a better one.

4.3 Truncated modalities

As for colimits, we define a truncated version of modalities, in order to use it in chapter 6. Basically, a truncated modality is the same as a modality, but restricted to Type_n.

Definition 4.10 : 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 4.3 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.

4.4 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 6, 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.

4.5 Translation

In this section, we will explain how left-exact modalities allows to perform model transformation, just as forcing does. Actually, we can to even better 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

$$[\mathrm{Type}] \stackrel{def}{=} (\mathrm{Type}^{\bigcirc}, \pi_{\mathrm{Type}^{\bigcirc}})$$

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

$$[\![A]\!] \stackrel{def}{=} \pi_1\,[A]$$

- For dependent sums

$$\begin{aligned}
[\textstyle\sum_{x:A} B] &\stackrel{def}{=} \left(\textstyle\sum_{x:[\![A]\!]} [\![B]\!], \pi_{\Sigma}^{[A],[B]}\right)\\
[(x,y)] &\stackrel{def}{=} ([x],[y])\\
[\pi_i t] &\stackrel{def}{=} \pi_i[t]
\end{aligned}$$

where $\pi_{\Sigma}^{A,B}$ is a proof that $\sum_{x:A} B$ is modal when A and B are.

- For dependent products

$$\begin{aligned}
[\textstyle\prod_{x:A} B] &\stackrel{def}{=} \left(\textstyle\prod_{x:[\![A]\!]} [\![B]\!], \pi_{\Pi}^{[A],[B]}\right)\\
[\lambda\, x : A,\ M] &\stackrel{def}{=} \lambda\, x : [\![A]\!],\ [M]\\
[t\, t'] &\stackrel{def}{=} [t][t']
\end{aligned}$$

where $\pi_{\Pi}^{A,B}$ is a proof that $\prod_{x:A} B$ is modal when B is.

- For paths

$$\begin{aligned}
[A = B] &\stackrel{def}{=} \left([A] = [B], \pi_{=}^{[A],[B]}\right)\\
[1] &\stackrel{def}{=} 1\\
[J] &\stackrel{def}{=} J
\end{aligned}$$

where $\pi_{=}^{A,B}$ is a proof that $A = B$ is modal when A and B are, if $A, B : \mathrm{Type}$, 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)

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

- For truncations ($i \leqslant n$)

$$\begin{aligned}
[\|A\|_i] &\overset{def}{=} (\bigcirc\|[\![A]\!]\|_i ; \pi_\bigcirc(\|[\![A]\!]\|_i)) \\
[|t|_i] &\overset{def}{=} \eta|[t]|_i \\
[|f|_i] &\overset{def}{=} \bigcirc^{\|[\![A]\!]\|_i}_{\text{rec}}|[f]|_i
\end{aligned}$$

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):

$$\begin{aligned}
&\mathbb{I}\ (a_1 : A_1)\cdots(a_n : A_n) : \text{Type} \overset{def}{=} \\
&|c_1 : \prod_{x_1 : X_1^1} \cdots \prod_{x_{n_1} : X_1^{n_1}} \mathbb{I}(a_1^1, \ldots, a_1^n) \\
&\qquad\vdots \\
&|c_p : \prod_{x_1 : X_p^1} \cdots \prod_{x_{n_1} : X_p^{n_1}} \mathbb{I}(a_p^1, \ldots, a_p^n)
\end{aligned}$$

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

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

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 $\mathbb{I}(a_1, \ldots, a_n)$ is defined as $\bigcirc(\widehat{\mathbb{I}}([a_1], \ldots, [a_n]))$.

Example

Consider the following inductive

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

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 4.11
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 4.12 : 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:

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

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

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

and

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

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

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

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

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

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

- $=$-form:

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

- $=$-intro:

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

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

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

□

This allows to state the following theorem

Theorem 4.13
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

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

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

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

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 4.14
Let $\bigcirc$ *be as in theorem 4.13. Then the univalence axiom remains true in the new type theory.*

Proof.

Lemma 4.15
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 `[bar]`, waits for the user to give a proof, and define a new term $\mathtt{foo}^m$: `[bar]` and a new constant `foo : bar` at the `Defined` command. Then, it adds in T a new pair $(\mathtt{foo}, \mathtt{foo}^m)$.

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

```
Modal Translate foo using ○
```

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

5 Colimits

A comathematician is a device
turning cotheorems into ffee.

Co-Alfréd Rényi

As seen in chapter 3, adding sigma-types to type theory results in adding limits over graphs in the underlying category, and adding higher inductives types results in adding colimits over graphs. If limits has been extensively studied in [AKL15], theory of colimits was not completely treated.

The following is conjoint work with Simon Boulier and Nicolas Tabareau, helped by precious discussions with Egbert Rijke. The sections 5.1 and 5.2 are extended version of the blog post [Bou16].

5.1 Colimits over graphs

As colimits are just dual to limits, it seems that it would be very easy to translate the work on limits to colimits. Although, even if it might be because we are more habituated to manipulate sigma-types than higher inductive types, it seems way harder.

5.1.1 Definitions

Let's recall the definitions of graphs and diagrams over graphs, introduced in [AKL15].

Definition 5.1 : Graph
A graph *G is the data of*

- *a type G_0 of vertices ;*
- *for any $i, j : G_0$, a type $G_1(i, j)$ of edges.*

Definition 5.2 : Diagram
A diagram *D over a graph G is the data of*

- *for any $i : G_0$, a type $D_0(i)$;*
- *for any $i, j : G_0$ and all $\phi : G_1(i, j)$, a map $D_1(\phi) : D_0(i) \to D_0(j)$*

When the context is clear, G_0 will be simply denoted G, $G_1(i,j)$ will be noted $G(i,j)$, $D_0(i)$ will be noted $D(i)$ or D_i, and $D_1(\phi)$ will be noted $D_{i,j}(\phi)$ or simply $D(\phi)$ (i and j can be inferred from ϕ).
Examples :

- One can consider the following graph, namely the graph of (co)equalizers

  $$\bullet \rightrightarrows \bullet$$

  Here, $G_0 = 2$, $G_1(\top,\bot) = 2$ and other $G_1(i,j)$ are empty.

  A diagram over this graph consists of two types A and B, and two maps $f,g : A \to B$, producing the diagram

  $$A \underset{g}{\overset{f}{\rightrightarrows}} B$$

- The graph of the mapping telescope is

  $$\bullet \longrightarrow \bullet \longrightarrow \cdots$$

  In other words, $G_0 = \mathbb{N}$ and $G_1(i,i+1) = \mathbf{1}$.

  A diagram over the mapping telescope is a sequence of types $P : \mathbb{N} \to$ Type together with arrows $f_n : P_n \to P_{n+1}$:

  $$P_0 \xrightarrow{f_0} P_1 \xrightarrow{f_1} \cdots$$

What we would like now would be to define the colimits of these diagrams over graphs, that would satisfy type theoretic versions of usual properties: it should make the diagram commute, and be universal with respect to this property. From now on, let G be a graph and D a diagram over this graph.

The commutation of the diagram is easy: the colimit should be the tip of a cocone.

Definition 5.3 : Cocone
Let Q be a type. A cocone over D into Q is the data of arrows $q_i : D_i \to Q$, and for any $i,j : G$ and $g : G(i,j)$, an homotopy $q_j \circ D(g) \sim q_i$.

If Q and Q' are type with an arrow $f : Q \to Q'$, and if C is a cocone over D into Q, one can easily build a cocone on D into Q' by postcomposing all maps of the cocone by f, giving a map

$$\text{postcompose}_{\text{cocone}} : \text{cocone}_D(Q) \to (Q' : \text{Type}) \to (Q \to Q') \to \text{cocone}_D(Q')$$

The other way around (from a cocone into Q', give a map $Q \to Q'$) is exactly the second condition we seek:

Definition 5.4 : Universality of a cocone
Let Q be a type, and C be a cocone over D into Q. C is said universal if for any type Q', $\mathrm{postcompose}_{\mathrm{cocone}}(C, Q')$ *is an equivalence.*

We can finally define what it means for Q to be a colimit of D.

Definition 5.5 : Colimit
A type Q is said to be a colimit of D if there is a cocone C over D into Q, which is universal.

Example

Let A, B be types and $f, g : A \to B$. Let Q be the HIT generated by $\left|\begin{array}{lcl} q & : & B \to Q \\ \alpha & : & q \circ f \sim q \circ g \end{array}\right.$. Then Q is a colimit of the coequalizer diagram associated to A, B, f, g. We say that Q is a coequalizer of f and g.

Note that for any diagram D, one can build a free colimit of D, namely the higher inductive type $\mathrm{colim}(D)$ generated by

$$\left|\begin{array}{lcl} \mathrm{colim} & : & \prod_{i:G} D_i \to \mathrm{colim}(D) \\ \alpha_{\mathrm{colim}} & : & \prod_{ij:G} \prod_{g:G(i,j)} \prod_{x:D_i} \mathrm{colim}_j \circ D(g) \sim \mathrm{colim}_i \end{array}\right.$$

5.1.2 Properties of colimits

Diagrams can be thought as functors from type of graphs to Type, and hence one can define morphisms between diagrams as natural transformation.

Definition 5.6 : Morphism of diagram
Let D and D' be two diagrams over the same graph G. A morphism of diagram m between D and D' is the data of

- *for all $i : G$, a map $m_i : D_i \to D'_i$*
- *for all $g : G(i, j)$, a path $D'(g) \circ m_i = m_j \circ D(g)$*

This definition leads to obvious definition of identity morphism of diagrams, and composition of morphisms of diagrams.

Then, two diagrams D and D' are said to be equivalent if there is a morphism of diagrams m between D and D' such that all D_i are equivalences. It can easily be checked that the morphism m^{-1} given by inverting all m_i's satisfies $m \circ m^{-1} = \mathrm{id}$ and $m^{-1} \circ m = \mathrm{id}$.

In the previous section, we defined the map $\mathrm{postcompose}_{\mathrm{cocone}}$ changing a cocone into Q and a map $Q \to Q'$ into a cocone into Q'. We now define a map

$\text{precompose}_{\text{cocone}}$ taking a morphism of diagrams m between D and D', and a cocone over D' into X, and giving a cocone over D into X:

Definition 5.7
Let m be a morphism of diagrams between D and D', and X a type. Any cocone over D' into X can be changed into a cocone over D into X, by precomposing all maps in the cocone by the m_i's:

$$\text{precompose}_{\text{cocone}} : \text{Hom}(D, D') \to \text{cocone}_{D'}(X) \to \text{cocone}_D(X).$$

Precomposition by a morphism of diagrams and postcomposition by a morphism are compatible with composition and identities, in the sense that

$$\begin{aligned}
\text{precompose}_{\text{cocone}}(m \circ m') &= (\text{precompose}_{\text{cocone}}\, m) \circ (\text{precompose}_{\text{cocone}}\, m') \\
\text{precompose}_{\text{cocone}}(\text{idmap}) &= \text{idmap} \\
\text{postcompose}_{\text{cocone}}(\varphi \circ \varphi') &= (\text{postcompose}_{\text{cocone}}\, \varphi) \circ (\text{postcompose}_{\text{cocone}}\, \varphi') \\
\text{postcompose}_{\text{cocone}}(\text{idmap}) &= \text{idmap}
\end{aligned}$$

These properties allows us to express functoriality properties of colimits. If m is a morphism between diagrams D and D', then $\text{postcompose}^{-1}_{\text{cocone}} \circ \text{precompose}_{\text{cocone}}$ is a map $Q \to Q'$, where Q (resp. Q') is a colimit of D (resp. D'). One can even check that if m is an equivalence of diagrams, then the produced map $Q \to Q'$ is an equivalence of types. This is the lemma:

Lemma 5.8
Let D and D' be two equivalent diagrams, with respective colimits Q and Q'. Then $Q \simeq Q'$.

In particular, identity of diagrams being an equivalence, it asserts that the colimit of a diagram is unique (up to equivalence). From now on, as we supposed since chapter 3 the univalence axiom, we will say *the* colimit of a diagram.

One more interesting property of colimits is that it is stable by dependent sums. More precisely, let X be a type, and D_x a diagram over a graph G for all $x : X$. We want to link the colimits Q_x of diagrams D_x with the colimit of $\sum_{x:X} D_x$.

Definition 5.9 : Dependent diagram
Let G be a graph, X a type and D_x a diagram over G for all $x : X$. The diagram $\sum_{x:X} D_x$ is the diagram defined by

- $\left(\sum_{x:X} D_x\right)(i) = \sum_{x:X} D_x(i)$

- $(\sum_{x:X} D_x)(\phi) = (\mathrm{idmap}, D_x(\phi))$

Note that a family C_x of cocones over D_x can be extended to a cocone $\sum_{x:X} C_x$ on $\sum_{x:X} D_x$. Then, the expected result is true

Proposition 5.10
If, for all $x : X$, Q_x is the colimit of D_x, then $\sum_{x:X} Q_x$ is the colimit of $\sum_{x:X} D_x$.

5.1.3 Truncated colimits

As said in the introduction, we now give a truncated version of colimits. Colimits actually behave well with respect to truncations. Indeed, if D is a diagram and P a colimit of D, then $\|P\|_n$ is the n-colimit of the n-truncated diagram $\|D\|_n$. Let's make it more precise.

Definition 5.11 : Truncation of a diagram
Let D be a diagram over a graph G, and n a truncation index. Then the diagram $\|D\|_n$ is the diagram over G defined by

- $(\|D\|_n)_0(i) \overset{def}{=} \|D_0(i)\|_n : \mathrm{Type}_n$
- $(\|D\|_n)_1(\phi) \overset{def}{=} |D_1(\phi)|_n : \|D_0(i)\|_n \to \|D_0(j)\|_n$

Definition 5.12
Let D be a diagram over a graph G, P be a type, and C a cocone over D into P. C is said n-universal if for any $Q : \mathrm{Type}_n$, $\mathrm{postcompose}_{\mathrm{cocone}}(C, Q)$ is an equivalence.

Then, P is said to be a n-colimit of D if there is a cocone C over D into Q which is n-universal.

We can now give the fundamental proposition linking colimit and n-colimit.

Proposition 5.13
Let D be a diagram, and $P : \mathrm{Type}$. Then, if P is a colimit of D, $\|P\|_n$ is a n-colimit of $\|D\|_n$.

The proof of this is really straightforward: a cocone over D into P can be changed equivalently into a cocone over $\|D\|_n$ into $\|P\|_n$, using the elimination principle 3.16 of truncations, and then we can show that the following

diagram commutes for any $X : \mathrm{Type}_n$

$$\begin{array}{ccc} \|P\|_n \to X & \longrightarrow & \mathrm{cocone}(\|D\|_n, X) \\ \downarrow \wr & & \uparrow \wr \\ P \to X & \xrightarrow{\sim} & \mathrm{cocone}(D, X) \end{array}$$

Nota

This result does not hold for limits. If it were true, then applying it to the following equalizer diagram

$$A \underset{\lambda\_, y}{\overset{f}{\rightrightarrows}} B$$

with $A, B : \mathrm{Type}$, $f : A \to B$ and $y : B$ would lead to an equivalence

$$\left\| \sum_{a:A} f\,a = y \right\|_n \simeq \sum_{a:\|A\|_n} |f|_n\,a = |y|_n,$$

proving left-exactness of n-truncation.

5.1.4 Towards highly coherent colimits

In category theory, a diagram over a graph G in a category $\mathcal{C}$ is a functor $G \to \mathcal{C}$. Our definition of diagrams doesn't reflect this definition, as we don't take care of composition of morphisms in the category (actually, we only define a map $G \to \mathcal{C}$). Diagrams over a category are defined in [HoTT, Exercise 7.16]. We will here define these diagrams, and the associated colimits.

Nota

This new definition of diagrams is not sufficient to reflect what we would want to express. If we view homotopy type theory as ω-topos theory, what we want is rather a ω-functor from a graph into the ω-topos. That is an important open problem in homotopy type theory (even defining simplicial types is an open problem). It requires to handle an infinity of coherence levels, with finite data.

However, we could define, for any fixed n, diagrams and colimits coherent up to coherence level n (but that would be a real nightmare).

The main goal of this section is to define the colimit of the coequalizer with degeneracy δ

$$A \underset{g}{\overset{f}{\rightrightarrows}} B \qquad \delta : A \to (f = g)$$

and compositions $c_f : f \circ \delta = \mathrm{idmap}$ and $c_g : g \circ \delta = \mathrm{idmap}$. We will use this coequalizer to define the Boulier's construction in next section 5.2.

If we only take the previous definition of colimits, what we miss is the commutation of the following diagram

$$\begin{array}{ccc} q \circ f \circ \delta & \xrightarrow{q \circ c_f} & q \\ {\scriptstyle \alpha \circ \delta}\downarrow & \nearrow_{q \circ c_g} & \\ q \circ g \circ \delta & & \end{array}$$

Our idea to define the colimit of this diagram with degeneracy is to add it directly into the definition of the higher inductive type:

$$\left|\begin{array}{lcl} \mathrm{colim} & : & B \to Q \\ \mathrm{colim}_\alpha & : & \prod_{x:A} f\,x = g\,x \\ \mathrm{colim}^\circ_\alpha & : & \prod_{y:B} \mathrm{colim}_\alpha(\delta\,y) = c_f(y) \cdot c_g(y)^{-1} \end{array}\right.$$

Nota

We indeed see here why this idea cannot be extended to a fully coherent notion of colimits: we cannot define a higher inductive type with an infinity of constructors. Hopefully, adding this first level of coherence will be sufficient for our applications.

The new constructor $\mathrm{colim}^\circ_\alpha$ indeed asserts the commutation of the desired diagram. We can view this higher inductive type as the same type as the colimit over the coequalizer graph, but which does not add already existing paths.

One can easily (but very technically) check that properties in section 5.1.2 are still satisfied by this new notion of colimit.

5.2 Van Doorn's and Boulier's constructions

In topos theory, there is a result that we would want to use in chapter 6:

Lemma 5.14 : [MM92, p. IV.7.8]
In a topos $\mathcal{E}$, if $f \in \mathrm{Hom}_{\mathcal{E}}(A,B)$ is an epimorphism, then the colimit of

$$A \times_B A \underset{\pi_2}{\overset{\pi_1}{\rightrightarrows}} A$$

is B. The pullback $A \times_B A$ is called the kernel pair of f.

Unfortunately, this result fails in higher topos ; the kernel pair should be replaced by the Čech nerve of f.

$$\ldots \;\substack{\rightarrow\\ \rightarrow\\ \rightarrow\\ \rightarrow}\; A \times_B A \times_B A \;\substack{\rightarrow\\ \rightarrow\\ \rightarrow}\; A \times_B A \rightrightarrows A \xrightarrow[\mathrm{colim}]{f} B$$

The issue we face in homotopy type theory is that the definition of the Čech nerve, and in general of simplicial objects is a hard open problem. It involves an infinite tower of coherences, and we do not know how to handle this. However, there is a way to define a diagram depending on a map f, which colimit is $\mathrm{Im}(f)$.

The starting point of the construction is Floris Van Doorn's construction of proposition truncation [Van16].

Proposition 5.15 : Van Doorn's construction
Let A : Type. *We define the higher inductive type* TA *as the coequalizer of*

$$A \times A \underset{\pi_2}{\overset{\pi_1}{\rightrightarrows}} A\ .$$

The colimit of the diagram

$$A \xrightarrow{q} TA \xrightarrow{q} TTA \xrightarrow{q} TTTA \xrightarrow{q} \dots$$

is $\|A\|_{-1}$.

Let's compare the direct definitions of $\|A\|_{-1}$ and TA

$$\|A\|_{-1} \left| \begin{array}{lcl} \mathrm{tr} & : & A \to \|A\|_{-1} \\ \alpha_{\mathrm{tr}} & : & \prod_{x,y:\|A\|_{-1}} x = y \end{array} \right. \qquad TA \left| \begin{array}{lcl} q & : & A \to TA \\ \alpha & : & \prod_{x,y:A} q\,x = q\,y \end{array} \right.$$

The definitions are almost the same, except that the path constructor of TA quantifies over A, while the one of $\|A\|_{-1}$ quantifies over $\|A\|_{-1}$ itself: such a higher inductive type is a *recursive* inductive type. Thus, proposition 5.15 allows us to build the truncation in a non-recursive way. The counterpart is that we have to iterate the construction. In [Bou16], we found a way to generalize a bit this result. The main idea is that iterating the kernel pair construction will result in a diagram of colimit $\mathrm{Im}(f)$.

Now, let A, B : Type and $f : A \to B$. We define the kernel pair of f as the coequalizer of

$$A \times_B A \underset{\pi_2}{\overset{\pi_1}{\rightrightarrows}} A\ .$$

In other words, $\mathrm{KP}(f)$ is the higher inductive type generated by

$$\left| \begin{array}{lcl} \mathrm{kp} & : & A \to \mathrm{KP}(f) \\ \alpha & : & \displaystyle\prod_{x,y:A} f\,x = f\,y \to kp\,x = \mathrm{kp}\,y \end{array} \right.$$

Using the eliminator of coequalizers, one can build a map $\widehat{f} : \mathrm{KP}(f) \to B$, such that the following commutes

$$\begin{array}{ccc} A & \xrightarrow{\mathrm{kp}} & \mathrm{KP}(f) \\ & \searrow^{f} & \downarrow{\scriptstyle \widehat{f}} \\ & & B \end{array}$$

Then we can build $KP(\widehat{f})$ and build a map $\widehat{\widehat{f}} : \mathrm{KP}(\widehat{f}) \to B$, *etc.* We have the following result

Proposition 5.16 : Boulier's construction
For any $f : A \to B$, $\mathrm{Im}(f)$ is the colimit of the iterated kernel pair diagram of f

$$A \longrightarrow \mathrm{KP}(f) \longrightarrow \mathrm{KP}\left(\widehat{f}\right) \longrightarrow \mathrm{KP}\left(\widehat{\widehat{f}}\right) \longrightarrow \cdots$$

In particular, if f is a surjection, the colimit of this diagram is B.

Proof. The main idea of the proof is the equivalence between the diagrams

$$A \longrightarrow \mathrm{KP}(f) \longrightarrow \mathrm{KP}\left(\widehat{f}\right) \longrightarrow \cdots$$

$$\sum_{y:B} \mathrm{fib}_f(y) \longrightarrow \sum_{y:B} T(\mathrm{fib}_f(y)) \longrightarrow \sum_{y:B} TT(\mathrm{fib}_f(y)) \longrightarrow \cdots$$

Let's begin by showing the first non-trivial equivalence:

$$s : \mathrm{KP}(f) \simeq \sum_{y:B} T(\mathrm{fib}_f(y)). \tag{5.1}$$

$\mathrm{KP}(f)$ is the colimit of $A \times_B A \overset{\pi_1}{\underset{\pi_2}{\rightrightarrows}} A$, and $\sum_{y:B} T(\mathrm{fib}_f(y))$ is the colimit of $\sum_{y:B} \mathrm{fib}_f(y) \times \mathrm{fib}_f(y) \rightrightarrows \sum_{y:B} \mathrm{fib}_f(y)$. As the two diagrams are equivalent, their colimits are equivalent. We will need the following fact, easily checked

$$\pi_1 \circ s = \widehat{f}.$$

Now, let's prove the other equivalences. We need the following lemma

Lemma 5.17
Let X, Y : Type and $\varphi : X \to Y$. Then $\mathrm{fib}_{\widehat{\varphi}}(y) \simeq T(\mathrm{fib}_{\varphi}(y))$.

Proof. We have the following sequence of equivalences:

$$\begin{aligned}
\mathrm{fib}_{\widehat{\varphi}}(y) &\overset{def}{=} \sum_{x:\mathrm{KP}(\varphi)} \widehat{\varphi}\,x = y \\
&\simeq \sum_{x:\sum_{y:B} T(\mathrm{fib}_\varphi(y))} \widehat{\varphi} \circ s^{-1}(x) = y \quad \text{by 5.1} \\
&\simeq \sum_{x:\sum_{y:B} T(\mathrm{fib}_\varphi(y))} \pi_1(x) = y \\
&\simeq T(\mathrm{fib}_\varphi(y))
\end{aligned}$$

◇

Then, using the sum-of-fibers property, we can change the iterated kernel pair of f into

$$\sum_{y:B} \mathrm{fib}_f(y) \longrightarrow \sum_{y:B} \mathrm{fib}_{\widehat{f}}(y) \longrightarrow \sum_{y:B} \mathrm{fib}_{\widehat{\widehat{f}}}(y) \longrightarrow \cdots$$

With the just proved lemma, and a bit of induction, we can prove the desired equivalence of diagrams.

As colimits are stable under dependent sum, we know that the colimit of the diagram is thus $\sum_{y:B} Q\,y$, where Q is the colimit of

$$\mathrm{fib}_f(y) \longrightarrow T(\mathrm{fib}_f(y)) \longrightarrow TT(\mathrm{fib}_f(y)) \longrightarrow \cdots$$

But proposition 5.15 asserts that $Q \simeq \|\mathrm{fib}_f(y)\|_{-1}$, and the result is proved. □

The main issue with Van Doorn's construction is that is does not preserve truncations levels at all. For example, when computing $\|\mathbf{1}\|_{-1}$, the first step is $T\mathbf{1} \simeq \mathbb{S}^1$, which is a 1-type. Asking for preservation of all truncation levels along the diagram might be too much, but the least we could ask is that when starting with a $P : \mathrm{HProp}$, the diagram should be the constant diagram $P \to P \to P \to \cdots$. The Boulier counterpart of this is, when starting with an embedding f, all $\widehat{f}$ are embeddings.

This can be achieve by changing operators T and KP, by asking them to preserve identities:

$$TA \left| \begin{array}{lcl} q & : & A \to TA \\ \alpha & : & \prod_{x,y:A} q\,x = q\,y \\ \alpha_1 & : & \prod_{x:A} \alpha(x,x) = 1 \end{array} \right. \quad ; \quad \mathrm{KP}(f) \left| \begin{array}{lcl} \mathrm{kp} & : & A \to \mathrm{KP}(f) \\ \alpha & : & \prod_{x,y:A} f\,x = f\,y \to q\,x = q\,y \\ \alpha_1 & : & \prod_{x:A} \alpha(x,x,1) = 1 \end{array} \right.$$

This new KP can be thought of as the coequalizer preserving the "degeneracy" δ, as in section 5.1.4:

$$A \times_B A \underset{\pi_2}{\overset{\pi_1}{\rightrightarrows}} A \qquad \delta : A \to A \times_B A$$

Then, the following is still true

Proposition 5.18 : Boulier's construction
For any $f : A \to B$, $\mathrm{Im}(f)$ is the colimit of the iterated kernel pair diagram of f

$$A \longrightarrow \mathrm{KP}(f) \longrightarrow \mathrm{KP}\left(\widehat{f}\right) \longrightarrow \mathrm{KP}\left(\widehat{\widehat{f}}\right) \longrightarrow \cdots$$

Moreover, if f is an embedding, $\widehat{f}$ also is.

The proof is almost the same as for proposition 5.16, except that the new constructors in the higher inductives types introduce another level of coherence, which is very technical to handle.

5.3 Formalization

All this chapter has been formalized with Coq, defining a library for colimits on top of the Coq/HoTT library [HoTT/Coq], based on the formalization of limits in [AKL15] for the basic definitions of graph, diagram, *etc.*

The root folder of the library formalizes results about general colimits over graphs, with the following key results:

- `Colimit_prod.is_colimit_prod` shows that if D is a diagram, and $A \times D$ the diagram where each node X of D is changed into $X \times D$, then $A \times \mathrm{colim} D$ is a colimit of $A \times D$.
- `Colimit_Sigma.is_colimit_sigma` shows the commutation of colimits with dependent sums, as in proposition 5.10.
- `Colimit_trunc.tr_colimit` shows the proposition 5.13 about truncated colimits.

The second part of the library, in folder `IteratedKernelPair`, formalizes the section 5.2. We note that the formalization for the kernel pair preserving degeneracy involves a lot of manipulation of paths between paths.

6 Sheaves in homotopy type theory

> Reductio ad absurdum, which Euclid loved so much, is one of a mathematician's finest weapons. It is a far finer gambit than any chess gambit: a chess player may offer the sacrifice of a pawn or even a piece, but a mathematician offers the game.
>
> Godfrey Harold Hardy

In topos theory, sheafification can be seen as a way to transform a topos into another one. It is used, for example, to build, from any topos $\mathcal{T}$, a boolean topos (*i.e.* satisfying the excluded middle property) satisfying the axiom of choice and negating the continuum hypothesis [MM92, Theorem VI.2.1]. This is actually an adaptation of a slightly older method, in set theory, to change a model $\mathfrak{M}$ of ZFC into a model $\mathfrak{M}[G]$ of ZFC, satisfying other principles, called *forcing*. It's most famous application is the proof of consistency of ZFC with the negation of the continuum hypothesis, by Paul Cohen [Coh66], answering (neither negatively nor positively) the first Hilbert's problem. Indeed, Gödel proved in 1938 the consistency of ZFC with continuum hypothesis [Göd38] using the constructible model $\mathfrak{L}$. The global idea of this technique is to add to the theory ZFC partial information about the witness of ¬CH. Then, supposing that ZFC is coherent, it is provable that ZFC together with a finite number of approximation of the desired object is still consistent. Then, the compactness theorem allows to prove the consistency of ZFC with *all* approximations, *i.e.* with a witness of ¬HC.

Then, forcing has been adapted to the setting of topos theory by Myles Tierney [Tie72], through the notion of sheaves. Note that, in topos theory, there are two different kind of sheaves: Grothendieck sheaves, which only exists on a presheaf topos, and Lawvere-Tierney sheaves. One can show that Lawvere-Tierney sheaves, when considered on a presheaf topos, are exactly the Grothendieck sheaves; thus, Lawvere-Tierney sheaves can be seen as a generalization of Grothendieck sheaves. Given a topos $\mathcal{T}$, one can build another topos – the topos of sheaves $\mathrm{Sh}(\mathcal{T})$ – together with geometric embedding from $\mathrm{Sh}(\mathcal{T})$ to $\mathcal{T}$ called sheafification. Depending on the sheaves we chose to treat, the topos $\mathrm{Sh}(\mathcal{T})$ satisfies new principles. The construction of

the geometric embedding is done in [MM92, Section V.3], and briefly recalled in section 6.1.

The development of higher topos theory (and more generally, higher category theory) leads to wonder if a notion of sheafification still exists in this setting. This question is answered positively in [HTT], where the author build a sheafification functor, but only for Grothendieck sheaves. Surprisingly, sheafification in a higher topos is just an iteration of the process of sheafification in topos theory. It seems that Lawvere-Tierney sheaves were not considered in this new setting.

Similar questions have been considered around the Curry-Howard isomorphism, to extend a programming language close to type theory with new logical or computational principle while keeping consistency automatically. For instance, much efforts have been done to provide a computational content to the law of excluded middle in order to define a constructive version of classical logic. This has lead to various calculi, with most notably the $\lambda\mu$-calculus of Parigot [Par93], but this line of work has not appeared to be fruitful to define a new version of type theory with classical principles. Other works have tried to extend continuation-passing-style (CPS) transformation to type theory, but they have been faced with the difficulty that the CPS transformation is incompatible with (full) dependent sums [BU02], which puts emphasis on the tedious link between the axiom of choice and the law of excluded middle in type theory. Nevertheless the axiom of choice has shown to be realizable by computational meaning in a classical setting by techniques turning around the notion of (modified) bar induction [BBC98], Krivine's realizability [Kri03] and even more recently with restriction on elimination of dependent sums and lazy evaluation [Her12]. The work on forcing in type theory [JTS12; Jab+16] also gives a computational meaning to a type theory enriched with new logical or computational principle.

Section 6.2 presents a definition of the sheafification functor in the setting of homotopy type theory. Actually, this construction is entirely complementary to forcing in type theory, as forcing corresponds to the presheaf construction while Lawvere-Tierney sheafification corresponds to the topological transformation that allows to go from the presheaf construction to the sheaf construction.

6.1 Sheaves in topoi

In this section, we will rather work in an arbitrary topos rather in type theory. The next section will present a generalisation of the results presented here.

Let us fix for the whole section a topos $\mathcal{E}$, with subobject classifier Ω. A *Lawvere-Tierney topology* on $\mathcal{E}$ is a way to modify slightly truth values of $\mathcal{E}$. It allows to speak about *locally true* things instead of *true* things.

Definition 6.1 : Lawvere-Tierney topology [MM92]
A Lawvere-Tierney topology is an endomorphism $j : \Omega \to \Omega$ preserving $\top$ ($j \top = \top$), idempotent ($j \circ j = j$) and commuting with products ($j \circ \wedge = \wedge \circ (j, j)$).

A classical example of Lawvere-Tierney topology is given by double negation. Other examples are given by Grothedieck topologies, in the sense

Theorem 6.2 : [MM92, Theorem V.1.2]
Every Grothendieck topology J on a small category $\mathbf{C}$ determines a Lawvere-Tierney topology j on the presheaf topos $\mathbf{Sets}^{\mathbf{C}^{\mathbf{op}}}$.

Any Lawvere-Tierney topology j on $\mathcal{E}$ induces a closure operator $A \mapsto \overline{A}$ on subobjects. If we see a subobject A of E as a characteristic function χ_A, the closure $\overline{A}$ corresponds to the subobject of E whose characteristic function is

$$\chi_{\overline{A}} = j \circ \chi_A.$$

A subobject A of E is said to be dense when $\overline{A} = E$.

Then, we are interested in objects of $\mathcal{E}$ for which it is impossible to make a distinction between objects and their dense subobjects, *i.e.* for which "true" and "locally true" coincide. Such objects are called *sheaves*, and are defined as

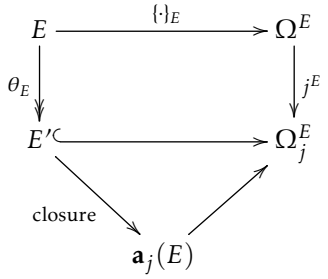
Definition 6.3 : Sheaves[MM92, Section V.2]
On object F of $\mathcal{E}$ is a sheaf (or j-sheaf) if for every dense monomorphism m : $A \hookrightarrow E$ in $\mathcal{E}$, the canonical map $\mathrm{Hom}_{\mathcal{E}}(E, F) \to \mathrm{Hom}_{\mathcal{E}}(A, F)$ is an isomorphism.

One can show that $\mathrm{Sh}_{\mathcal{E}}$, the full sub-category of $\mathcal{E}$ given by sheaves, is again a topos, with classifying object

$$\Omega_j = \{P \in \Omega \mid jP = P\}.$$

Lawvere-Tierney sheafification is a way to build a left adjoint $\mathbf{a}_j$ to the inclusion $\mathcal{E} \hookrightarrow \mathrm{Sh}_(\mathcal{E})$, exhibiting $\mathrm{Sh}_(\mathcal{E})$ as a reflective subcategory of $\mathcal{E}$. In particular, that implies that logical principles valid in $\mathcal{E}$ are still valid in $\mathrm{Sh}_(\mathcal{E})$.

For any object E of $\mathcal{E}$, $\mathbf{a}_j(E)$ is defined as in the following diagram



The proof that $\mathbf{a}_j$ defines a left adjoint to the inclusion can be found in [MM92].

One classical example of use of sheafification is the construction, from any topos, of a boolean topos negating the continuum hypothesis. More precisely:

Theorem 6.4 : Negation of CH [MM92, Theorem VI.2.1]
There exists a Boolean topos satisfying the axiom of choice, in which the continuum hypothesis fails.

The proof actually follows almost exactly the famous proof of the construction by Paul Cohen of a model of ZFC negating the continuum hypothesis [Coh66]. Together with the model of constructible sets $\mathfrak{L}$ by Kurt Gödel [Göd40], it proves that CH is independent of ZFC, solving first Hilbert's problem.

6.2 Sheaves in homotopy type theory

The idea of this section is to consider sheafification in topoi as only the first step towards sheafification in type theory. We note that axioms for a Lawvere-Tierney topology on the subobject classifier Ω of a topos are very close to those of a modality on Ω. We will extensively use this idea, applying it to every subobject classifier Type_n we described in 3.4. The subobject classifier Ω of a topos is seen as the *truth values* of the topos, which corresponds to the type HProp in our setting ; the topos is considered proof irrelevant, corresponding to our HSet. Sheafification in topoi is thus a way, when translated to the setting of homotopy type theory, to build, from a left-exact modality on HProp, a left-exact modality on HSet. Our hope in this section is to iterate this construction by applying it to the subobject classifier HSet equipped with a left-exact modality, to build a new left-exact modality on Type_1, and so on.

The first thing we can note is that such a construction will not allow to reach every type: it is known that there exists types with no finite truncation level [HoTT, Example 8.8.6]. Even worse, some types are not even the limit of its successive truncations, even in a hypercomplete setting [MV98]. It suggests that defining a sheafification functor for all truncated types won't give (at least easily) a sheafification functor on whole Type. Another issue that can be pointed is the complexity of proofs. If, in a topos-theoretic setting, everything is proof-irrelevant, it won't be the case for higher settings, forcing us to prove results that were previously true on the nose. This will oblige us to write long and technical proofs of coherence, and more deeply, to modify completely some lemmas, such as Proposition [MM92, Theorem IV.7.8], stating that epimorphisms are coequalizers of their kernel pair.

The main idea is thus to follow as closely as possible the topos-theoretic construction, and change it as few times as possible to make it work in our higher setting.

Note that the principles we want to add are added directly from the HProp level, the extension to all truncated types is automatic. The choice of the left-exact modality on HProp is thus crucial. For the rest of the section, we fix one, noted $\bigcirc_{-1}$. The reader can think of the double negation $\bigcirc_{\neg\neg}$ defined in 4.2.3. We will define, by induction on the truncation level, left-exact modalities on all Type_n, as in the following theorem.

Theorem 6.5
The sequence defined by induction by

$$\begin{aligned}
&\bigcirc : \forall\ (n : nat),\ \mathrm{Type}_n \to \mathrm{Type}_n \\
&\bigcirc_{-1}\ (T) \stackrel{def}{=} j\,T \\
&\bigcirc_{n+1}(T) \stackrel{def}{=} \sum_{u:T\to \mathrm{Type}_n^{\bigcirc}} \bigcirc_{-1} \left\| \sum_{a:T} u = (\lambda t,\ \bigcirc_n (a = t)) \right\|
\end{aligned}$$

defines a sequence of left-exact modalities, coherent with each others in the sense that the following diagram commutes for any $P : \mathrm{Type}_n$, where $\hat{P}$ is P seen as an inhabitant of Type_{n+1}.

$$\begin{array}{ccc}
P & \xrightarrow{\sim} & \widehat{P} \\
{\scriptstyle \eta_n}\downarrow & & \downarrow{\scriptstyle \eta_{n+1}} \\
\bigcirc_n P & \xrightarrow{\sim} & \bigcirc_{n+1}\widehat{P}
\end{array}$$

6.2.1 Sheaf theory

Let n be a truncation index greater that -1, and $\bigcirc_n$ be the left-exact modality given by our induction hypothesis. As in the topos-theoretic setting, we will define what it means for a type to be a n-sheaf (or just "sheaf" if the context is clear), and consider the reflective subuniverses of these sheaves ; the reflector will exactly be the sheafification functor. The main issue to give the "good" definition is the choice of the subobject classifier in which dense subobjects will be chosen: two choices appears, HProp and Type_n ; we will actually use both. What guided our choice is the crucial property that the type of all n-sheaves has to be a $(n+1)$-sheaf.

From the modality $\bigcirc_n$, one can build a *closure operator*.

Definition 6.6 : (`cloture`,`closed`,`EnJ`)
Let E be a type.

- *The* closure *of a subobject of E with n-truncated homotopy fibers (or n-subobject of E, for short), classified by $\chi : E \to \mathrm{Type}_n$, is the subobject of E classified by $\bigcirc_n \circ \chi$.*

- *An n-subobject of E classified by χ is said to be* closed in E *if it is equal to its closure,* i.e. *if $\chi = \bigcirc_n \circ \chi$.*
- *An n-subobject of E classified by χ is said to be* dense in E *if its closure is E,* i.e. *if $\bigcirc_n \circ \chi = \lambda e, \mathbf{1}$*

Topos-theoretic sheaves are characterized by a property of existence and uniqueness, which will be translated, as usual, into a proof that a certain function is an equivalence.

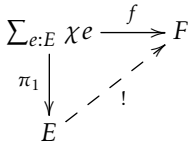
Definition 6.7 : Restriction (`E_to_χmono_map`, `E_to_χ_map`)
Let E, F : Type *and $\chi : E \to$* Type*. We define the* restriction map Φ_E^χ *as*

$$\Phi_E^\chi : \begin{array}{ccc} E \to F & \longrightarrow & \sum_{e:E} \chi e \to F \\ f & \longmapsto & f \circ \pi_1 \end{array} .$$

Here, we need to distinguish between dense (-1)-subobjects, that will be used in the definition of sheaves, and dense n-subobjects, that will be used in the definition of separated types.

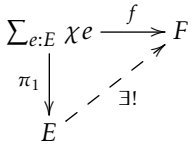
Definition 6.8 : Separated Type (`separated`)
A type F in Type_{n+1} *is* separated *if for any type E, and all dense n-subobject of E classified by χ, Φ_E^χ is an embedding.*

With topos theory point of view, it means that given a map $\sum_{e:E} \chi e \to F$, if there is an extension $\tilde{f} : E \to F$, then it is unique, as in



Definition 6.9 : Sheaf (`Snsheaf_struct`)
A type F of Type_{n+1} *is a* $(n+1)$-sheaf *if it is separated, and for any type E and all dense (-1)-subobject of E classified by χ, Φ_E^χ is an equivalence.*

In topos-theoretic words, it means that given a map $f : \sum_{e:E} \chi e \to F$, one can extend it uniquely to $\tilde{f} : E \to F$, as in



Note that these definitions are almost the same as the ones in [MM92]. The main difference is that separated is defined for n-subobjects, while sheaf only for (-1)-subobjects. It might seem bizarre to make such a distinction, but the following proposition gives a better understanding of the situation.

Proposition 6.10 : (nj_paths_separated)
A type F is Type_{n+1} *is separated if, and only if all its path types are n-modal, ie*

$$\prod_{x,y:F} (\bigcirc_n(x=y)) = (x=y).$$

A $(n+1)$-sheaf is hence just a type satisying the usual property of sheaves (*i.e.* existence of uniqueness of arrow extension from dense (-1)-subobjects), with the condition that all its path types are n-sheaves. It is a way to force the compatibility of the modalities we are defining.

One can check that the property IsSeparated (resp. IsSheaf) is HProp: given a $X : \mathrm{Type}_{n+1}$, there is at most one way for it to be separated (resp. a sheaf). In particular, when needed to prove equality between two sheaves, it suffices to show the equality between the underlying types.

As said earlier, these definitions allow us to prove the fundalemental property that the type of all n-sheaves is itself a $(n+1)$-sheaf (this can be viewed as an equivalent definition of left-exactness).

Proposition 6.11 : (nType_j_Type_is_SnType_j_Type)
$\mathrm{Type}_n^{\bigcirc}$ *is a $(n+1)$-sheaf.*

Proof. We have two things to prove here : separation, and sheafness.

- Let E : Type and $\chi : E \to \mathrm{Type}$, dense in E. Let $\phi_1, \phi_2 : E \to \mathrm{Type}_n^{\bigcirc}$, such that $\phi_1 \circ \pi_1 = \phi_2 \circ \pi_1$ and let $x : E$. We show $\phi_1(x) = \phi_2(x)$ using univalence.

  As χ is dense, we have a term $m_x : \bigcirc_n(\chi\, x)$. But as $\phi_2(x)$ is modal, we can obtain a term $h_x : \chi\, x$. As ϕ_1 and ϕ_2 are equal on $\sum_{e:E} \chi\, e$, we have an arrow $\phi_1(x) \to \phi_2(x)$. The same method leads to an arrow $\phi_2(x) \to \phi_1(x)$, and one can prove that they are each other inverse.

- Now, we prove that $\mathrm{Type}_n^{\bigcirc}$ is a sheaf. Let E : Type and $\chi : E \to \mathrm{HProp}$, dense in E. Let $f : \sum_{e:E} \chi\, e \to \mathrm{Type}_n^{\bigcirc}$. We want to extend f into a map $E \to \mathrm{Type}_n^{\bigcirc}$.

  We define g as $g(e) = \bigcirc_n\Big(\mathrm{fib}_\phi(e)\Big)$, where

$$\phi : \sum_{b:\sum_{e:E} \chi\, e} (f\, b) \to E$$

defined by $\phi(x) = (x_1)_1$. Using the following lemma, one can prove that the map $f \mapsto g$ defines an inverse of Φ_E^χ.

Lemma 6.12 : (`nj_fibers_compose`)
Let $A, B, C : \mathrm{Type}_n$, $f : A \to B$ and $g : B \to C$. Then if all fibers of f and g are n-truncated, then

$$\prod_{c:C} \left(\bigcirc_n(\mathrm{fib}_{g\circ f}(c)) \right) \simeq \bigcirc_n \left(\sum_{w:\mathrm{fib}_g(c)} \bigcirc_n(\mathrm{fib}_f(w_1)) \right).$$

Proof. This is just a modal counterpart of the property characterizing fibers of composition of function. ◇

□

Another fundamental property on sheaves we will need is that the type of (dependent) functions is a sheaf as soon as its codomain is a sheaf.

Proposition 6.13 : (`dep_prod_SnType_j_Type`)
If $A : \mathrm{Type}_{n+1}$ and $B : A \to \mathrm{Type}_{n+1}$ such that for any $a : A$, $(B\ a)$ is a sheaf, then $\prod_{a:A} B\,a$ is a sheaf.

Proof. Again, when proving equivalences, we will only define the maps. The proofs of section and retraction are technical, not really interesting, and present in the formalisation.

- *Separation:* Let $E : \mathrm{Type}$ and $\chi : E \to \mathrm{Type}_n$ dense in E. Let $\phi_1, \phi_2 : E \to \prod_{a:A} B\,a$ equal on $\sum_{e:E} \chi\, e$ *i.e.* such that $\phi_1 \circ \pi_1 = \phi_2 \circ \pi_1$. Then for any $a : A$, $(\lambda x : E,\ \phi_1(x,a))$ and $(\lambda x : E,\ \phi_2(x,a))$ coincide on $\sum_{e:E}(\chi\, e)$, and as $B\,a$ is separated, they coincide also on all E.
- *Sheaf:* Let $E : \mathrm{Type}$, $\chi : E \to \mathrm{HProp}$ dense in E and $f : \sum_{e:E} \chi\, e \to \prod_{a:A} B\,a$. Let $a : A$; the map $(\lambda x,\ f(x,a))$ is valued in the sheaf $B\,a$, so it can be extended to all E, allowing f to be extended to all E.

□

6.2.2 Sheafification

The sheafification process will be defined in two steps. The first one will build, from any $T : \mathrm{Type}_{n+1}$, a separated object $\square_{n+1} T : \mathrm{Type}_{n+1}$; one can show that $\square_{n+1}$ defines a modality on Type_{n+1}. The second step will build, from any separated type $T : \mathrm{Type}_{n+1}$, a sheaf $\bigcirc_{n+1}(T)$; one can show that $\bigcirc_{n+1}$ is indeed the left-exact modality we are searching.

Let n be a fixed truncation index, and $\bigcirc_n$ a left-exact modality on Type_n, compatible with $\bigcirc_{-1}$ as in

Condition 6.14
For any mere proposition P (where $\widehat{P}$ is P seen as a Type_n), $\bigcirc_n \widehat{P} = \bigcirc_{-1} P$ and the following coherence diagram commutes

$$\begin{array}{ccc} P & \xrightarrow{\sim} & \widehat{P} \\ {\scriptstyle \eta_{-1}}\downarrow & & \downarrow{\scriptstyle \eta_n} \\ \bigcirc_{-1} P & \xrightarrow{\sim} & \bigcirc_n \widehat{P} \end{array}$$

From types to separated types

Let $T : \mathrm{Type}_{n+1}$. We define $\square_{n+1}\, T$ as the image of $\bigcirc_n^T \circ \{\cdot\}_T$, as in

$$\begin{array}{ccc} T & \xrightarrow{\{\cdot\}_T} & (\mathrm{Type}_n)^T \\ {\scriptstyle \mu_T}\downarrow & & \downarrow{\scriptstyle \bigcirc_n^T} \\ \square_{n+1}\, T & \longrightarrow & \left(\mathrm{Type}_n^{\bigcirc}\right)^T \end{array},$$

where $\{\cdot\}_T$ is the singleton map $\lambda(t : T),\ \lambda(t' : T),\ t = t'$. $\square_{n+1}\, T$ can be given explicitly by

$$\begin{aligned} \square_{n+1}\, T &\overset{def}{=} \mathrm{Im}(\lambda\, t : T,\ \lambda\, t',\ \bigcirc_n(t = t')) \\ &\overset{def}{=} \sum_{u : T \to \mathrm{Type}_n^{\bigcirc}} \left\| \sum_{a:A} (\lambda t,\ \bigcirc_n(a = t)) = u \right\|. \end{aligned}$$

This corresponds to the free separated object used in the topos-theoretic construction, but using $\mathrm{Type}_n^{\bigcirc}$ instead of the j-subobject classifier Ω_j.

Proposition 6.15 : (`separated_Type_is_separated`)
For any $T : \mathrm{Type}_{n+1}$, $\square_{n+1}\, T$ is separated.

Proof. We use the following lemma:

Lemma 6.16 : (`separated_mono_is_separated`)
A $(n+1)$-truncated type T with an embedding $f : T \to U$ into a separated $(n+1)$-truncated type U is itself separated.

Proof. Let $E : \mathrm{Type}$ and $\chi : E \to \mathrm{Type}_n$ dense in E. Let $\phi_1, \phi_2 : \sum_{e:E} \chi\, e \to T$ such that $\phi_1 \circ \pi_1 \sim \phi_2 \circ \pi_1$. Postcomposing by f yields an homotopy $f \circ \phi_1 \circ \pi_1 \sim$

$f \circ \phi_2 \circ \pi_1$. As $f \circ \phi_1, f \circ \phi_2 : \sum_{e:E} \chi\, e \to U$, and U is separated, we can deduce $f \circ \phi_1 \sim f \circ \phi_2$. As f is an embedding, $\phi_1 \sim \phi_2$. ◇

As $\square_{n+1} T$ embeds in $\left(\text{Type}_n^{\bigcirc}\right)^T$, we only have to show that the latter is separated. But it is the case because $\text{Type}_n^{\bigcirc}$ is a sheaf (by Proposition 6.11) and a function type is a sheaf as soon as its codomain is a sheaf (by Proposition 6.13). □

We will now show that $\square_{n+1}$ defines a modality, with unit map μ. The left-exactness of $\bigcirc_{n+1}$ will come from the second part of the process. The first thing to show that $\square_{n+1} T$ is universal among separated type below T. In the topos-theoretic sheafification, it comes easily from the fact that epimorphims are coequalizers of their kernel pairs. As it is not true anymore in our setting, we will use its generalization, proposition 5.18. Here is a sketch of the proof: as μ_T is a surjection (it is defined by the surjection-embedding factorization), $\square_{n+1} T$ is the colimit of its iterated kernel pair. Hence, for any type Q defining a cocone on $\text{KP}(\mu_T)$, there is a unique arrow $\square_{n+1} T \to Q$. What remains to show is any separated type Q defines a cocone on $\text{KP}(\mu_T)$; we will actually show that any separated type Q defines a cocone on $\|\text{KP}(\mu_T)\|_{n+1}$, which is enough. We do it by defining another diagram $\mathring{T}$, equivalent to $\|\text{KP}(\mu_T)\|_{n+1}$, for which it is easy to define a cocone into any separated type Q.

This comes from the following construction which connects $\square_{n+1} T$ to the colimit of the iterated kernel pair of μ_T.

Definition 6.17 : (`0Tid`)
Let X : Type. Let $\mathring{T}_X$ be the higher inductive type generated by

- $\mathring{t} : \|X\|_{n+1} \to \mathring{T}_X$
- $\mathring{\alpha} : \forall a\, b : \|X\|_{n+1},\ \bigcirc(a = b) \to \mathring{t}(a) = \mathring{t}(b)$
- $\mathring{\alpha}_1 : \forall a : \|X\|_{n+1},\ \mathring{\alpha}(a, a, \eta_{a=a} 1) = 1$

We view $\mathring{T}$ as the coequalizer of

$$\sum_{a,b:\|X\|_{n+1}} \bigcirc(a = b) \underset{\pi_2}{\overset{\pi_1}{\rightrightarrows}} \|X\|_{n+1}$$

preserving $\eta_{a=a} 1$.

We consider the diagram $\mathring{T}$:

$$\|X\|_{n+1} \longrightarrow \|\mathring{T}_X\|_{n+1} \longrightarrow \|\mathring{T}_{\mathring{T}_X}\|_{n+1} \longrightarrow \cdots$$

The main result we want about $\mathring{T}$ is the following:

Lemma 6.18 : (`separation_colimit_0Ttelescope`)
Let T : Type_{n+1}. Then $\square_{n+1} T$ is the $(n+1)$-colimit of the diagram $\mathring{T}$.

The key point of the proof is that diagrams $\mathring{T}$ and $\|\mathrm{KP}(\mu_T)\|_{n+1}$ are equivalent. We will need the following lemma:

Lemma 6.19 : (0T_0mono_sep)
Let $A, S : \mathrm{Type}_{n+1}$, S separated, and $f : A \to S$. Then if

$$\forall a, b : A,\ f(a) = f(b) \simeq \bigcirc(a = b), \tag{6.1}$$

then

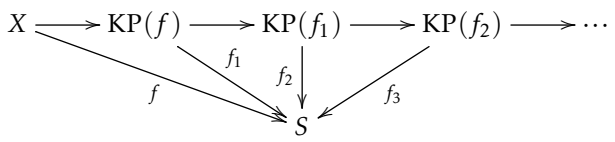
$$\forall a, b : \|\mathrm{KP}_f\|_{n+1},\ |\tilde{f}|_{n+1}(a) = |\tilde{f}|_{n+1}(b) \simeq \bigcirc(a = b).$$

Sketch of proof. By induction on truncation, we need to show that

$$\forall a, b : \mathrm{KP}_f,\ \tilde{f}(|a|_{n+1}) = \tilde{f}(|b|_{n+1}) \simeq \bigcirc(|a|_{n+1} = |b|_{n+1}).$$

We use the encode-decode [HoTT, Section 8.9] method to characterize $\tilde{f}(|a|_{n+1}) = x$, and the result follows. We refer to the formalization for details. □

This lemmas allows to prove that, in the iterated kernel pair diagram of f



if f satisties 6.1, then each $|f_i|_{n+1}$ does.

Nota

It is clear that if A and B are equivalent types, and $\forall a, b : A,\ f(a) = f(b) \simeq \bigcirc(a = b)$, then

$$\mathrm{Coeq}_1\left(\sum_{a,b:A} fa = fb \underset{\pi_2}{\overset{\pi_1}{\rightrightarrows}} A\right) \simeq \mathrm{Coeq}_1\left(\sum_{a,b:B} \bigcirc(a = b) \underset{\pi_2}{\overset{\pi_1}{\rightrightarrows}} B\right)$$

Proof of lemma 6.18. As said, it suffices to show that $\|C(\mu_T)\|_{n+1} = \mathring{T}$.

$$\begin{array}{ccccccc}
\|\mathrm{KP}^0(\mu_T)\|_{n+1} & \longrightarrow & \|\mathrm{KP}^1(\mu_T)\|_{n+1} & \longrightarrow & \|\mathrm{KP}^2(\mu_T)\|_{n+1} & \longrightarrow & \cdots \\
\downarrow \wr & & \downarrow \wr & & \downarrow \wr & & \\
\mathring{T}_0 & \longrightarrow & \mathring{T}_1 & \longrightarrow & \mathring{T}_2 & \longrightarrow & \cdots
\end{array}$$

The first equivalence is trivial. Let's then start with the second. What we need to show is

$$\|\mathrm{KP}(\mu_T)\|_{n+1} \simeq \|\mathring{T}_T\|_{n+1},$$

i.e.

$$\mathrm{Coeq}_1\left(\sum_{a,b:T} \mu_T a = \mu_T b \overset{\pi_1}{\underset{\pi_2}{\rightrightarrows}} T\right) \simeq \mathrm{Coeq}_1\left(\sum_{a,b:T} \bigcirc(a=b) \overset{\pi_1}{\underset{\pi_2}{\rightrightarrows}} T\right).$$

By the previous remark, it suffices to show that μ_T satisfies condition (6.1), *i.e.* $\forall a,b:T, \bigcirc_n(a=b) = (\mu_T a = \mu_T b)$. By univalence, we want arrows in both ways, forming an equivalence.

- Suppose $p : (\mu_T a = \mu_T b)$. Then projecting p along first components yields $q : \prod_{t:T} \bigcirc_n(a=t) = \bigcirc_n(b=t)$. Taking for example $t = b$, we deduce $\bigcirc_n(a=b) = \bigcirc_n(b=b)$, and the latter is inhabited by $\eta_{b=b}1$.
- Suppose now $p : \bigcirc_n(a=b)$. Let ι be the first projection from $\square_{n+1} T \to (T \to \mathrm{Type}_n^{\bigcirc})$. ι is an embedding, thus it suffices to prove $\iota(\mu_T a) = \iota(\mu_T b)$, *i.e.* $\prod_{t:T} \bigcirc_n(a=t) = \bigcirc_n(b=t)$. The latter remains true by univalence.

The fact that these two form an equivalence is technical, we refer to the formalization for an explicit proof.

Let's show the other equivalences by induction. Suppose that, for a given $i : \mathbb{N}$, $\|\mathrm{KP}^i(\mu_T)\|_{n+1} \simeq \mathring{T}_i$. We want to prove $\|\mathrm{KP}^{i+1}(\mu_T)\|_{n+1} \simeq \mathring{T}_{i+1}$, *i.e.*

$$\left\|\mathrm{Coeq}_1\left(\sum_{a,b:\mathrm{KP}^i(\mu_T)} f_i a = f_i b \overset{\pi_1}{\underset{\pi_2}{\rightrightarrows}} \mathrm{KP}^i(\mu_T)\right)\right\|_{n+1}$$
$$\simeq \left\|\mathrm{Coeq}_1\left(\sum_{a,b:\|\mathring{T}_i\|_{n+1}} \bigcirc(a=b) \overset{\pi_1}{\underset{\pi_2}{\rightrightarrows}} \|\mathring{T}_i\|_{n+1}\right)\right\|_{n+1}$$

where f_i is the map $\mathrm{KP}^i(\mu_T) \to \square_{n+1} T$. But lemma 6.19 just asserted that f_i satisfies 6.1, hence the previous nota yields the result.

One would need to show that, modulo these equivalences, the arrows of the two diagrams are equal. We leave that to the reader, who can refer to the formalization if needed. □

Now, let Q be any separated Type_{n+1}, and $f : X \to Q$. Then the following diagram commutes

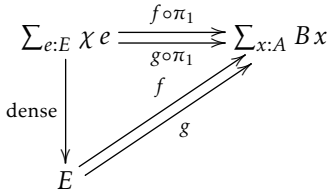
$$\begin{array}{ccccccc} \|X\|_{n+1} & \longrightarrow & \|\mathring{T}_X\|_{n+1} & \longrightarrow & \|\mathring{T}_{\mathring{T}_X}\|_{n+1} & \longrightarrow & \cdots \\ & \searrow & \downarrow & \swarrow & & & \\ & & Q & & & & \end{array}$$

But we know (lemma 6.18) that $\square_{n+1} T$ is the $(n+1)$-colimit of the diagram $\mathring{T}$, thus there is an universal arrow $\square_{n+1} T \to Q$. This is enough to state the following proposition.

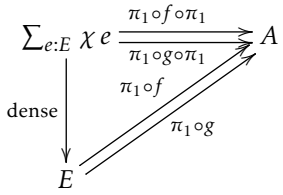
Proposition 6.20 : (separation_reflective_subuniverse)
$(\square_{n+1}, \mu)$ *defines a reflective subuniverse on* Type_{n+1}.

To show that $\square_{n+1}$ is a modality, it remains to show that separation is a property stable under sigma-types. Let $A : \mathrm{Type}_{n+1}$ be a separated type and $B : A \to \mathrm{Type}_{n+1}$ be a family of separated types. We want to show that $\sum_{x:A} Bx$ is separated. Let E be a type, and $\chi : E \to \mathrm{Type}_n$ a dense subobject of E.

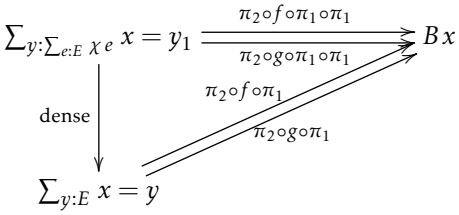
Let f, g be two maps from $\sum_{e:E} \chi e$ to $\sum_{x:A} Bx$, equal when precomposed with π_1.



We can restrict the previous diagram to



and as A is separated, $\pi_1 \circ f = \pi_1 \circ g$. For the second components, let $x : E$. Notice that $\sum_{y:E} x = y$ has a dense n-subobject, $\sum_{y:\sum_{e:E} \chi e} x = y_1$:



Using the separation property of Bx, one can show that second components, transported correctly along the first components equality, are equal. The complete proof can be found in the formalization. This proves the following proposition

Proposition 6.21 : (separated_modality)
$(\square_{n+1}, \mu)$ *defines a modality on* Type_{n+1}.

As this modality is just a step in the construction, we do not need to show that it is left exact, we will only do it for the sheafification modality.

From Separated Type to Sheaf

For any T in Type_{n+1}, $\bigcirc_{n+1} T$ is defined as the closure of $\square_{n+1} T$, seen as a subobject of $T \to \mathrm{Type}_n^{\bigcirc}$. $\bigcirc_{n+1} T$ can be given explicitly by

$$\bigcirc_{n+1} T \overset{def}{=} \sum_{u:T \to \mathrm{Type}_n^{\bigcirc}} \bigcirc_{-1} \left\| \sum_{a:T} (\lambda t,\ \bigcirc_n (a = t)) = u \right\|.$$

To prove that $\bigcirc_{n+1} T$ is a sheaf for any $T : \mathrm{Type}_{n+1}$, we use the following lemma.

Lemma 6.22 : (`closed_to_sheaf`)
Any closed (-1)-subobject of a sheaf is a sheaf.

Proof. Let U be a sheaf, and $\kappa : U \to \mathrm{HProp}$ be a closed (-1)-subobject. Let $E : \mathrm{Type}$ and $\chi : E \to \mathrm{HProp}$ dense in E. Let $\phi : \sum_{e:E} \chi\, e \to \sum_{u:U} \kappa\, u$. As $\pi_1 \circ \phi$ is a map $\sum_{e:E} \chi\, e \to U$ and U is a sheaf, it can be extended into $\psi : E \to U$. As κ is closed, it suffices now to prove $\prod_{e:E} \bigcirc_n (\kappa\, (\psi\, e))$ to obtain a map $E \to \sum_{u:U} \kappa\, u$.

Let $e : E$. As χ is dense, we have a term $w : \bigcirc_n (\chi\, e)$, and by $\bigcirc_n$-induction, a term $\widetilde{w} : \chi\, e$. Then, by retraction property, $\psi(e) = \phi(e, \widetilde{w})$, and by $\pi_2 \circ \phi$, we have hence our term of type $\kappa(\psi\, e)$.

□

As $T \to \mathrm{Type}_n^{\bigcirc}$ is a sheaf, and $\bigcirc_{n+1} T$ is closed in $T \to \mathrm{Type}_n^{\bigcirc}$, $\bigcirc_{n+1} T$ is a sheaf. We now prove that it forms a reflective subuniverse.

Proposition 6.23 : (`sheafification_subu`)
$(\bigcirc_{n+1}, \nu)$ *defines a reflective subuniverse.*

Proof. Let $T, Q : \mathrm{Type}_{n+1}$ such that Q is a sheaf. Let $f : T \to Q$. Because Q is a sheaf, it is in particular separated; thus we can extend f to $\square_{n+1} f : \square_{n+1} T \to Q$.

But as $\bigcirc_{n+1} T$ is the closure of $\square_{n+1} T$, $\square_{n+1} T$ is dense into $\bigcirc_{n+1} T$, so the sheaf property of Q allows to extend $\square_{n+1} f$ to $\bigcirc_{n+1} f : \bigcirc_{n+1} T \to Q$.

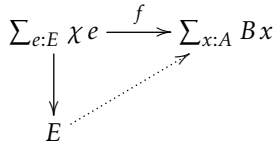
As all these steps are universal, the composition is. □

The next step is the closure under dependent sums, to state:

Proposition 6.24 : (`sheafification_modality`)
$(\bigcirc_{n+1}, \nu)$ *defines a modality.*

Proof. The proof uses the same ideas as in subsection 6.2.2. Let $A : \mathrm{Type}_{n+1}$ a sheaf and $B : A \to \mathrm{Type}_{n+1}$ a sheaf family. By proposition 6.21, we already

know that $\sum_{a:A} Ba$ is separated. Let E be a type, and $\chi : E \to \mathrm{HProp}$ a dense subobject. Let $f : \sum_{e:E} \chi\, e \to \sum_{x:A} B\, x$; we want to extend it into a map $E \to \sum_{x:A} B\, x$.



As A is a sheaf, and $\pi_1 \circ f : \sum_{e:E} \chi\, e \to A$, wa can recover an map $g_1 : E \to A$. We then want to show $\prod_{e:E} B(g_1\, e)$. Let $e : E$. As χ is dense, we have a term $w : \bigcirc_n(\chi\, e)$, and as $B(g_1\, e)$ is a sheaf, we can recover a term $\widetilde{w} : \chi\, e$. Then $g_1(e) = f(e, \widetilde{w})$, and $\pi_2 \circ f$ gives the result. □

It remains to show that $\bigcirc_{n+1}$ is left exact and is compatible with $\bigcirc_{-1}$. To do that, we need to extend the notion of compatibility and show that actually every modality $\bigcirc_{n+1}$ is compatible with $\bigcirc_n$ on lower homotopy types.

Proposition 6.25
If $T : \mathrm{Type}_n$, then $\bigcirc_{n+1}\widehat{T} = \bigcirc_n T$, where $\widehat{T}$ is T seen as a Type_{n+1}.

Proof. We prove it by induction on n:

- For $n = -1$: Let $T : \mathrm{HProp}$. Then

$$\begin{aligned}\bigcirc_0 \widehat{T} &\overset{def}{=} \sum_{u:T\to \mathrm{Type}_n^{\bigcirc}} \bigcirc_{-1} \left\| \sum_{a:T} (\lambda t,\ \bigcirc_{-1}(a = t)) = u \right\|_{-1} \\ &= \sum_{u:T\to \mathrm{Type}_n^{\bigcirc}} \bigcirc_{-1} \left(\sum_{a:T} (\lambda t,\ \bigcirc_{-1}(a = t)) = u \right)\end{aligned}$$

  because the type inside the truncation is already in HProp. Now, let define $\phi : \bigcirc_{-1} T \to \bigcirc_0 T$ by

$$\phi t = (\lambda t', \mathbf{1}; \kappa)$$

  where κ is defined by $\bigcirc_{-1}$-induction on t. Indeed, as T is an HProp, $(a = t) \simeq \mathbf{1}$. Let $\psi : \bigcirc_0 T \to \bigcirc_{-1} T$ by obtaining the witness $a : T$ (which is possible because we are trying to inhabit a modal proposition), and letting $\psi(u; x) = \eta_T a$. These two maps form an equivalence (the section and retraction are trivial because the equivalence is between mere propositions).

- Suppose now that $\bigcirc_{n+1}$ is compatible with all $\bigcirc_k$ on lower homotopy types. Let $\bigcirc_{n+2}$ be as above, and let $T : \mathrm{Type}_{n+1}$. Then, as $\bigcirc_{n+1}$ is

compatible with $\bigcirc_n$, and $(a = t)$ is in Type_n,

$$\bigcirc_{n+2}\widehat{T} = \sum_{u:T\to \mathrm{Type}_{n+1}^{\bigcirc}} \bigcirc_{-1} \left\| \sum_{a:T} (\lambda t,\ \bigcirc_n(a = t)) = u \right\|_{-1}.$$

It remains to prove that for every (u,x) inhabiting the Σ-type above, u is in $T \to \mathrm{Type}_n^{\bigcirc}$, *i.e.* that for every $t : T$, Is-n-type$(u\,t)$. But for any truncation index p, the type Is-p-type X : HProp is a sheaf as soon as X is, so we can get rid of $\bigcirc_{-1}$ and of the truncation, which tells us that for every $t : T$, $u\,t = \bigcirc_n(a = t) : \mathrm{Type}_n$. □

This proves in particular that $\bigcirc_{n+1}$ is compatible with $\bigcirc_{-1}$ in the sense of condition 6.14.

The last step is the left-exactness of $\bigcirc_{n+1}$. Let T be in Type_{n+1} such that $\bigcirc_{n+1}T$ is contractible. Thanks to the just shown compatibility between $\bigcirc_{n+1}$ and $\bigcirc_n$ for Type_n, left exactness means that for any $x,y : T$, $\bigcirc_n(x = y)$ is contractible.

Using a proof by univalence as we have done for proving $\bigcirc_n(a = b) \simeq (\mu_T(a) = \mu_T(b))$ in Proposition 6.18, we can show that:

Proposition 6.26 : (`good_sheafification_unit_paths_are_nj_paths`)
For all $a,b : T$, $\bigcirc_n(a = b) \simeq (\nu_T a = \nu_T b)$.

As $\bigcirc_{n+1}T$ is contractible, path spaces of $\bigcirc_{n+1}T$ are contractible, in particular $(\nu_T a = \nu_T b)$, which proves left exactness.

6.2.3 Summary

Starting from any left-exact modality $\bigcirc_{-1}$ on HProp, we have defined for any truncation level n, a new left-exact modality $\bigcirc_n$ on Type_n, which corresponds to $\bigcirc_{-1}$ when restricted to HProp.

When $\bigcirc_{-1}$ is consistent (in the sense of proposition 4.13), $\bigcirc_n\mathbf{0} = \bigcirc_{-1}\mathbf{0}$ is also not inhabited, hence $\bigcirc_n$ is consistent. In particular, the modality induced by the double negation modality on HProp is consistent.

In topos theory, the topos of Lawvere-Tierney sheaves for the double negation topology is a boolean topos. In homotopy type theory, this result can be expressed as:

Proposition 6.27
Taking $(\bigcirc_{\neg\neg})_n$, the modality obtained by sheafification of the double negation modality, the following holds

$$\prod_{P:\mathrm{HProp}} \bigcirc_{\neg\neg}(P + \neg P).$$

Proof. Let $P : \mathrm{HProp}$, and pose $Q \overset{def}{=} P + \neg P$. Then, as P and $\neg P$ are disjoint h-propositions, $P + \neg P$ is itself a h-proposition [HoTT/Coq, `ishprop_sum`]. Thus, $\bigcirc_{\neg\neg} Q \simeq \neg\neg Q$, and the latter is inhabited by the usual

$$\lambda\,(x : \neg Q),\, x(\mathrm{inr}(\lambda\, y : P,\, x(\mathrm{inl}\, y))).$$

□

6.2.4 Extension to Type

In the previous section, we have defined a (countably) infinite family of modalities $\mathrm{Type}_i \to \mathrm{Type}_i$. One can extend them to whole Type by composing with truncation:

Lemma 6.28
Let $\bigcirc_i : \mathrm{Type}_i \to \mathrm{Type}_i$ be a modality. Then $\bigcirc \overset{def}{=} \bigcirc_i \circ \|\cdot\|_i : \mathrm{Type} \to \mathrm{Type}$ is a modality in the sense of section 4.1

If $\bigcirc_{-1}$ is the double negation modality on HProp and $i = -1$, $\bigcirc$ is exactly the double negation modality on Type described in 4.2.3. Chosing $i \geqslant 0$ is a refinement of this double negation modality on Type: it will collapse every type to a Type_i, instead of an HProp.

Obviously, as truncation modalities are not left-exact [HoTT, Exercise 7.11], $\bigcirc$ isn't either. But in the following sense, when restricted to i-truncated types, it is:

Lemma 6.29
Let $A : \mathrm{Type}_i$. Then if $\bigcirc(A)$ is contractible, for any $x, y : A$, $\bigcirc(x = y)$ is contractible.

Proof. For i-truncated types, $\bigcirc = \bigcirc_i$, and $\bigcirc_i$ is left-exact. □

The compatibility between the modalities $\bigcirc_n$ and between the modalities $\|\cdot\|_n$ allow us to chose the truncation index as high as desired. Taking it as a non-fixed parameter allows to work in an universe where the new principle (*e.g.* mere excluded middle) is true for any explicit truncated type. Indeed, i can be chosen dynamically along a proof, and thus be increased as much as needed, without changing results for lower truncated types.

Furthermore, the univalence remains true in this new type theory in the following sense:

Proposition 6.30
Let n be a given truncation index, and $\bigcirc$ the modality associated to n as defined in lemma 6.28. Then, for any type $A, B : \mathrm{Type}_n^{\bigcirc}$, if φ is the canonical

arrow

$$A = B \to A \simeq B,$$

then IsEquiv(φ) *is modal.*

Proof. The first thing to notice is that, if X and Y are modal, and $f : X \to Y$, then the mere proposition IsEquiv f is also modal. Therefore, it suffices to show that both $A = B$ and $A \simeq B$ are modal. By proposition 4.3, $A = B$ is modal. Moreover, $(A \simeq B) \simeq \sum_{f:A\to B} \text{IsEquiv}\, f$. Therefore, as A and B are modal, $A \simeq B$ is too.

Hence, IsEquiv φ is modal. □

We can view sheafification in terms of model of type theory but because of the resulting modality on Type is not left exact, we need to restrict ourselves to a type theory with only one universe. Let $\mathfrak{M}$ be a model of homotopy type theory with one universe. Using the modality $\bigcirc_{\neg\neg}$ (for any level n) associated to the sheafification, there is a model $\bigcirc_{\neg\neg}\mathfrak{M}$ of type theory with one universe (using results in Section 4.5), where excluded middle is true, and which is univalent (as shown in Proposition 6.30).

6.3 Formalization

A Coq formalization of the sheafification process based on the Coq/HoTT library [HoTT/Coq] is available at https://github.com/KevinQuirin/sheafification.

After reviewing the content and some statistics about the formalization in Section 6.3.1, we present the limitations of our formalization in Section 6.3.2, in particular the issues relative to universe polymorphism.

6.3.1 Content of the formalization

We provide a more detailed insight of the structure of our formalization:

- Colimits and iterated kernel pairs are formalized in `Limit`, `T.v`, `OT.vv` `OT_Tf.v`, `T_telescope.v`, `Tf_Omono_sep.v`.
- Reflective subuniverses and modalities are formalized in `reflective_subuniverse.v`, `modalities.v`.
- The definition of the dense topology as a left exact modality on HProp is given in `sheaf_base_case.v`.
- Section 6.2.1 is formalized in `sheaf_def_and_thm.v`.
- Section 6.2.2 is formalized in `sheaf_induction.v`.

Overall, the project contains 8000 lines, and it could be reduced a bit by improving the way Coq tries to rewrite and apply lemmas automatically. The `coqwc` tool counts 1600 lines of specifications (definitions, lemmas, theorems, propositions) and 5500 lines of proof script. This constitutes a significant amount of work but the part dedicated to sheaves and sheafification is only 2200 lines of proof script, which seems quite reasonable and encouraging, because it suggests that homotopy type theory provides a convenient tool to formalize some part of the theory of higher topoï.

6.3.2 Limitations of the formalization

In the formalization, we had to use the `type-in-type` option, to handle the universe issues we faced. However, a lot of the code compiles without this flag, but need universe polymorphism.

Universes are used in type theory to ensure consistency by checking that definitions are well-stratified according to a certain hierarchy. Universe polymorphism [ST14] supports generic definitions over universes, reusable at different levels. Although the presence of universe polymorphism is mandatory for our formalization, its implementation is still too rigid to allow a complete formalization of our work for the following reasons.

If Coq handles cumulativity on Type natively, it is not the case for the Σ-type Type_n, which require propositional resizing. This issue could be solved by adding an axiom of cumulativity for Type_n with an explicit management of universes. But as it would not have any computational content, such a solution would really complicate the proofs as the axiom would appear everywhere cumulativity is needed and it would need explicit annotations for universe levels everywhere in the formalization.

One issue with universe polymorphism lies in the management of recursive definitions. Indeed, the following recursive definition of sheafification

$$\begin{aligned}
&\bigcirc : \forall\ (n : nat),\ \mathrm{Type}_n \to \mathrm{Type}_n \\
&\bigcirc_{-1}\ (T) \overset{def}{=} \neg\neg T \\
&\bigcirc_{n+1}(T) \overset{def}{=} \sum_{u : T \to \mathrm{Type}_n^{\bigcirc}} \bigcirc_{-1} \left\| \sum_{a:T} u = (\lambda t,\ \bigcirc_n (a = t)) \right\|
\end{aligned}$$

is not allowed. This is because Coq forces the universe of the first Type_n occurring in the definition to be the same for every n, whereas the universe of the first Type_{n+1} occurring in $\bigcirc_{n+1}$ should be at least one level higher as the one of Type_n occurring in $\bigcirc_n$ because of the use of Σ-type over $T \to \mathrm{Type}_n^{\bigcirc}$ and equality on the return type of $\bigcirc_n$. Thus, the induction step presented in this paper has been formalized, but the complete recursive sheafification can not be defined for the moment. Note that the same increasing in the universe levels occurs in the Rezk completion for categories [AKS15]. In the definition of the completion, they use the Yoneda embedding and representable functors, which is similar to our use of characteristic functions.

This restriction in our formalization may be solved by generalizing the management of universe polymorphism for recursive definition or by the use of general "resizing axiom" which is still under discussion in the community.

7 Conclusion and future works

Let us begin this conclusion by a summary of this thesis. Homotopy type theory is a new research domain, and consists of Martin-Löf type theory where we give a homotopical interpretation of identity types, together with the univalence axiom, linking equivalence of types with their equality, and higher inductive types, giving us a way to build non-trivial equalities. It seems that there exists a very strong link between higher topos theory [HTT] and this theory. More precisely, homotopy type theory is expected to be the internal language of $(\infty,1)$-toposes.

In homotopy type theory, types are classified by their *truncation level*, representing the complexity of its iterated loop spaces. In particular, if X is $(n+1)$-truncated, then each $x = y$ with $x, y : X$ is n-truncated. We will use this property to build an operator on all truncated types by induction on the truncation level.

The operator we want to build is a *modality*. Modalities are generalized version of localizations, which themselves are a way to characterize equivalently the notion of subtopos; this equivalence still holds in higher topos theory [HTT, Section 6.2.2]. In homotopy type theory, a modality is an operator $\bigcirc$ on Type, together with unit maps $\eta : \prod_{X:\mathrm{Type}} X \to \bigcirc X$ satisfying properties. They can be seen just as idempotent monads. Relying on the equivalence between type theory and $(\infty,1)$-toposes, one can conjecture that modalities – actually, accessible left-exact modalities – induces reflective sub-type theories. In this thesis, we want to describe a classical type theory (*i.e.* with the law of excluded middle) a sub-type theory of homotopy type theory, using a modality. We already know that it is possible in topos theory: take any topos $\mathcal{T}$, and the double-negation Lawvere-Tierney topology, and build the topos of sheaves $\mathrm{Sh}_{\neg\neg}(\mathcal{T})$. Then the latter is a boolean topos (*i.e.* satisfying the LEM).

The main idea of our work is to notice that Lawvere-Tierney topologies on a topos, which are operator on the subobject classifier, can be seen, in the setting of homotopy type theory, as modalities restricted to HProp, the second layer of the stratification of types. Moreover, the sheafification functor corresponds to extend this truncated modality to HSet, the next layer. Thus, we believed that it was possible to extend it again to the next level, *etc.* to finally give a modality on all truncated types. Actually, *modulo* some changes in several proofs – in particular the proof involving kernel pairs of arrows – and some sophistications in proofs, the method described in the topos theoretic

setting can be repeated infinitely to build a truncated modality on all level of our stratification.

Unfortunately, the actual management of universes by Coq does not allow us to formalize this result completely, but most of the proof is computer-checked. Some parts can only be checked using the (inconsistent) `type-in-type` option, allowing $\mathrm{Type}^i : \mathrm{Type}^i$, but the whole inductive definition cannot be checked at all. However, the key points of the definition of the functor are formalized.

Nevertheless, our work still need enhancements:

Extension to Type. At the moment, our sheafification functor only handles truncated type, and we have to compose it with truncations. It would be way more satisfying to be able to define it on whole Type left-exactly. The main issue is that some types, which are not n-truncated for any n, are not even the limit of their truncations [MV98]. Therefore, there seems to be no way to create a link between a non-truncated type and truncated types, to extend our inductive definition. It might be possible to have such a link using axioms such as Whitehead's principle [HoTT, Section 8.8] or Postnikov principle [HTT, Section 5.5.6], and use it to build a real modality on Type.

Lawvere-Tierney sheaves in higher topos theory. If we rely on the leitmotiv

> Homotopy type theory is the internal language of $(\infty, 1)$-topos,

we could transpose our work to higher topos theory. As there are more tools in topos theory (*e.g.* we can access the definitional equality), it could be a first step in solving the previous future work. This kind of "reverse engineered" proof has already been done for a proof of the Blakers-Massey theorem by Charles Rezk [Rez], inspired by the homotopy-type-theoretic proof by Peter LeFanu Lumsdaine, Eric Finster and Dan Licata.

Lawvere-Tierney subsumes Grothendieck? In topos theory, there are two different notions of sheaves: the Grothendieck sheaves and the Lawvere-Tierney sheaves. The former is a topological, geometrical concept, while the latter is rather a logical concept. Grothendieck sheaves are based on *Grothendieck topologies* [MM92, Chapter III], and one can show that Lawvere-Tierney topologies on a presheaf topos $\mathbf{Sets}^{\mathbf{C}^{\mathrm{op}}}$ correspond exactly to Grothendieck topologies on $\mathbf{C}$. Then, we have the following:

Theorem 7.1 : [MM92, Section V.4, theorem 2]
Let $\mathbf{C}$ *is a small category and* j *a Lawvere-Tierney topology on* $\mathbf{Sets}^{\mathbf{C}^{\mathrm{op}}}$*, while* J *is the corresponding Grothendieck topology on* $\mathbf{C}$*. Then a presheaf* P *is a sheaf for* j *iff* P *is a* J*-sheaf.*

The concept of Grothendieck sheaf and Grothendieck sheafification already exists in $(\infty,1)$-topos [HTT, Section 6.2.2]. It would be nice to check if theorem 7.1 still holds, either in the setting of homotopy type theory or in the setting of higher topos. The former requires to formalize Grothendieck topologies, sheaves and sheafification from higher topos theory to homotopy type theory, while the latter requires to work on the previous point.

What remains to be done

Nomenclature

$(x : A) \to (B\,a)$	Dependent product over B, page 21
$p \cdot q$	Concatenation of paths, page 24
$\mathrm{fib}_f(b)$	Fiber of f over b, page 22
idpath or 1	Constant path, page 23
idpath_x or 1_x	Constant path over$x$, page 23
p^{-1}	Inverse of path, page 24
$\mathbb{N}$	Type of naturals, page 22
$\mathbf{1}$	Unit type, page 20
$\prod_{a:A} B\,a$	Dependent product over B, page 21
ΣA	Suspension of a type, page 30
$\simeq$	Equivalence of types, page 26
$\sum_{a:A} B\,a$	Dependent sum over B, page 22
Type^i	i-th universe, page 20
Type_n	Type of n-truncated types, page 32
$\|\cdot\|$	n-truncation of types, page 34
$\|\cdot\|_n$	n-truncation of terms or arrows, page 34
$\mathbf{0}$	Empty type, page 20
$A + B$	Coproduct of types, page 21
$a : A$	Judgement "a is of type A", page 19
$a =_A b$	Type of paths from a to b in A, page 23
$a = b$	Type of paths from a to b, page 23
$A \times B$	Product of types, page 21
$A \to B$	Type of arrows from A to B, page 21

Bibliography

[AGS12] Steve Awodey, Nicola Gambino, and Kristina Sojakova. "Inductive types in homotopy type theory". In: *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science*. IEEE Computer Society. 2012, pp. 95–104 (cit. on p. 22).

[AKL15] Jeremy Avigad, Krzysztof Kapulkin, and Peter LeFanu Lumsdaine. "Homotopy limits in type theory". In: *Mathematical Structures in Computer Science* (Published online: 19 january 2015) (cit. on pp. 8, 53, 63).

[AKS15] Benedikt Ahrens, Krzysztof Kapulkin, and Michael Shulman. "Univalent Categories and the Rezk completion". In: *Mathematical Structures in Computer Science* (Published online: 19 january 2015) (cit. on p. 83).

[AW09] Steve Awodey and Michael A. Warren. "Homotopy theoretic models of identity types". In: *Math. Proc. Cambridge Philos. Soc.* 146.1 (2009), pp. 45–55. ISSN: 0305-0041. DOI: 10.1017/S0305004108001783. eprint: 0709.0248. URL: http://dx.doi.org/10.1017/S0305004108001783 (cit. on pp. 3, 15).

[BBC98] Stefano Berardi, Marc Bezem, and Thierry Coquand. "On the computational content of the axiom of choice". In: *Journal of Symbolic Logic* (1998), pp. 600–622 (cit. on p. 66).

[BL11] Andrej Bauer and Peter LeFanu Lumsdaine. *A Coq proof that Univalence Axioms implies Functional Extensionality*. 2011. URL: https://github.com/andrejbauer/Homotopy/raw/master/OberwolfachTutorial/univalence.pdf (cit. on p. 27).

[Bou16] Simon Boulier. *Colimits in HoTT*. Blog post. 2016. URL: http://homotopytypetheory.org/2016/01/08/colimits-in-hott/ (cit. on pp. 53, 60).

[BU02] Gilles Barthe and Tarmo Uustalu. "CPS translating inductive and coinductive types". In: *ACM SIGPLAN Notices* 37.3 (2002), pp. 131–142 (cit. on p. 66).

[Coh66] Paul Cohen. *Set theory and the continuum hypothesis*. WA Benjamin New York, 1966 (cit. on pp. 4, 16, 65, 68).

[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. 47).

[Göd38] Kurt Gödel. "The consistency of the axiom of choice and of the generalized continuum-hypothesis". In: *Proceedings of the National Academy of Sciences* 24.12 (1938), pp. 556–557 (cit. on p. 65).

[Göd40] Kurt Gödel. *The Consistency of the Axiom of Choice and of the Generalized Continuum Hypothesis with the Axioms of Set Theory*. Princeton University Press, 1940 (cit. on p. 68).

[Gol03] Robert Goldblatt. "Mathematical modal logic: a view of its evolution". In: *Journal of Applied Logic* 1.5 (2003), pp. 309–392 (cit. on p. 37).

[Gon+13] Georges Gonthier et al. "A machine-checked proof of the odd order theorem". In: *Interactive Theorem Proving*. Springer Berlin Heidelberg, 2013, pp. 163–179 (cit. on pp. 1, 13).

[Gon08] Georges Gonthier. "Formal proof–the four-color theorem". In: *Notices of the AMS* 55.11 (2008), pp. 1382–1393 (cit. on pp. 1, 13).

[GW] Herman Geuvers and Freek Wiedijk. "A logical framework with explicit conversions". In: (cit. on p. 47).

[Hal+15] Thomas Hales et al. "A formal proof of the Kepler conjecture". In: *arXiv preprint arXiv:1501.02155* (2015) (cit. on pp. 1, 13).

[Hal07] Thomas C Hales. "Jordan's proof of the Jordan curve theorem". In: *Studies in logic, grammar and rhetoric* 10.23 (2007), pp. 45–60 (cit. on pp. 1, 13).

[Her12] Hugo Herbelin. "A constructive proof of dependent choice, compatible with classical logic". In: *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science*. IEEE Computer Society. 2012, pp. 365–374 (cit. on p. 66).

[Hof95] Martin Hofmann. "Extensional concepts in intensional type theory". In: (1995) (cit. on pp. 2, 14).

[Hof97] Martin Hofmann. "Syntax and semantics of dependent types". In: *Extensional Constructs in Intensional Type Theory*. Springer, 1997, pp. 13–54 (cit. on pp. 5, 19).

[HoTT] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. IAS: http://homotopytypetheory.org/book, 2013 (cit. on pp. 5, 17, 19, 20, 23, 25, 30, 33, 37, 40, 48, 58, 68, 75, 81, 86).

[HoTT/Coq] The Univalent Foundations Program. *Coq HoTT library*. URL: https://github.com/HoTT/HoTT/ (cit. on pp. i, 17, 29, 37, 44, 50, 63, 81, 82).

[How80] William A. Howard. "The formulas-as-types notion of construction". In: *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*. Ed. by J. P. Seldin and J. R. Hindley. Reprint of 1969 article. Academic Press, 1980, pp. 479–490 (cit. on p. 20).

[HS96] Martin Hofmann and Thomas Streicher. "The Groupoid Interpretation of Type Theory". In: *In Venice Festschrift*. Oxford University Press, 1996, pp. 83–111 (cit. on pp. 2, 14, 26).

[HTS] Vladimir Voevodsky. *A simple type system with two identity types*. Started in 2013, in progress (cit. on p. 24).

[HTT] Jacob Lurie. *Higher topos theory*. Annals of mathematics studies. Princeton, N.J., Oxford: Princeton University Press, 2009 (cit. on pp. 4, 11, 16, 17, 50, 66, 85–87).

[Hut11] Michael Hutchings. *Introduction to higher homotopy groups and obstruction theory*. 2011. URL: https://math.berkeley.edu/~hutching/ (cit. on p. 31).

[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 pp. 4, 15, 44, 66).

[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. 4, 15, 44, 47, 66).

[Kra15] Nicolai Kraus. "Truncation Levels in Homotopy Type Theory". PhD thesis. University of Nottingham, June 2015 (cit. on p. 34).

[Kri03] Jean-Louis Krivine. "Dependent choice,'quote'and the clock". In: *Theoretical Computer Science* 308.1 (2003), pp. 259–276 (cit. on p. 66).

[Kun] Kenneth Kunen. *Set theory*. Ed. by North-Holland (cit. on pp. 11, 37).

[Lic] Dan Licata. *Another proof that univalence implies function extensionality*. Blog post. URL: http://homotopytypetheory.org/2014/02/17/another-proof-that-univalence-implies-function-extensionality/ (cit. on pp. 3, 15, 27).

[LS13] Daniel R. Licata and Michael Shulman. "Calculating the Fundamental Group of the Circle in Homotopy Type Theory". In: *LICS 2013: Proceedings of the Twenty-Eighth Annual ACM/IEEE Symposium on Logic in Computer Science*. 2013 (cit. on p. 28).

[m31] Andrej Bauer. *Andromeda implementation* (cit. on p. 24).

[Mar98] Per Martin-Löf. "An intuitionistic theory of types". In: *Twenty-five years of constructive type theory* 36 (1998), pp. 127–172 (cit. on p. 20).

[MM92] Saunders MacLane and Ieke Moerdijk. *Sheaves in Geometry and Logic*. Springer-Verlag, 1992 (cit. on pp. 4, 16, 59, 65–68, 71, 86).

[Mog91] Eugenio Moggi. "Notions of Computation and Monads". In: *Information and Computation* 93 (1991), pp. 55–92 (cit. on p. 37).

[Mou+15] Leonardo Mendonça de Moura et al. "The Lean Theorem Prover (System Description)". In: *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*. 2015, pp. 378–388 (cit. on pp. 1, 13).

[MV98] Fabien Morel and Vladimir Voevodsky. "A^1-homotopy theory of schemes". In: (1998) (cit. on pp. 68, 86).

[Nor07] Ulf Norell. "Towards a practical programming language based on dependent type theory". PhD thesis. Chalmers University of Technology, 2007 (cit. on pp. 1, 13).

[NPS01] B Nordström, K Petersson, and JM Smith. *Martin-Löf's type theory, Handbook of logic in computer science: Volume 5: Logic and algebraic methods*. Oxford University Press, Oxford, 2001 (cit. on p. 20).

[Par93] Michel Parigot. "Classical proofs as programs". In: *Computational logic and proof theory*. Springer, 1993, pp. 263–276 (cit. on p. 66).

[Rez] Charles Rezk. "Proof of the Bakers-Massey theorem". In: () (cit. on p. 86).

[RS13] Egbert Rijke and Bas Spitters. "Sets in homotopy type theory". In: *arXiv preprint arXiv:1305.3835* (2013) (cit. on p. 33).

[RSS] Egbert Rijke, Michael Shulman, and Bas Spitters. "On modalities in Homotopy Type Theory". In preparation (cit. on p. 37).

[Shu11] Mike Shulman. *Localization as an Inductive Definition*. Blog post. 2011. URL: https://homotopytypetheory.org/2011/12/06/inductive-localization/ (cit. on p. 50).

[Shu12] Mike Shulman. *All Modalities are HITs*. Blog post. 2012. URL: https://homotopytypetheory.org/2012/11/19/all-modalities-are-hits/ (cit. on p. 50).

[Shu15] Mike Shulman. *Module for Modalities*. Blog post. 2015. URL: http://homotopytypetheory.org/2015/07/05/modules-for-modalities/ (cit. on p. 44).

[ST14] Matthieu Sozeau and Nicolas Tabareau. "Universe Polymorphism in Coq". In: *Interactive Theorem Proving*. 2014 (cit. on p. 83).

[Str93] Thomas Streicher. "Investigations Into Intensional Type Theory". Habilitationsschrift. LMU München, 1993 (cit. on p. 26).

[The12] The Coq development team. *Coq 8.4 Reference Manual*. Inria. 2012. URL: http://coq.inria.fr/distrib/V8.4pl3/refman/ (cit. on pp. 1, 13).

[Tie72] Myles Tierney. *Sheaf theory and the continuum hypothesis*. Springer, 1972 (cit. on pp. 4, 16, 65).

[Van16] Floris Van Doorn. "Constructing the Proposition Truncation using Non-recursive HITs". In: *Certified Proofs and Programs* (2016) (cit. on p. 60).

[Voe10] Vladimir Voevodsky. "Univalent Foundations Project". In: *a modified version of an NSF grant application* (Oct. 2010), 1–12. URL: http://www.math.ias.edu/vladimir/files/univalent_foundations_project.pdf (cit. on pp. 3, 15).

[Voe11] Vladimir Voevodsky. *Resising Rules - their use and semantic justification*. 2011 (cit. on p. 44).

[Voe14] Vladimir Voevodsky. *Univalent Foundations*. 2014 (cit. on pp. 1, 13).

[WX10] Guozhen Wang and Zhouli Xu. "A Survey of Computations of Homotopy Groupes of Spheres and Cobordisms". In: (2010) (cit. on p. 31).