Introduction au domaine de recherche : Théorie des types homotopiques et fondations univalentes des mathématiques

Guillaume Brunerie, sous la direction de Carlos Simpson

Septembre 2012

Introduction

La théorie des types homotopiques est une théorie à mi-chemin entre la théorie du typage en informatique théorique, la théorie de l'homotopie en topologie algébrique et les fondements des mathématiques en philosophie des sciences. C'est une théorie très jeune et en pleine expansion. L'un des premiers articles ayant pressenti le lien entre théorie du typage et théorie de l'homotopie est un article de Hofmann et Streicher sur l'interprétation groupoïdale de la théorie des types, qui a été publié en 1995 ([HS95]). Quelques autres articles sur ce sujet ont été publiés vers la fin des années 2000 par Steve Awodey, Michael Warren, Peter Lumsdaine, Richard Garner, Benno van den Berg, et d'autres, mais le domaine n'a vraiment commencé à prendre de l'ampleur qu'à partir de 2009, après l'énoncé par Vladimir Voevodsky de son axiome d'univalence et l'annonce de son programme des fondations univalentes des mathématiques, un vaste programme de recherche visant à refonder les mathématiques sur la théorie des types dépendants augmentée de l'axiome d'univalence.

Un groupe de travail a été organisé en février/mars 2011 à l'institut allemand d'Oberwolfach, réunissant divers informaticiens et mathématiciens intéressés par les rapports entre théorie de l'homotopie et théorie des types et par l'axiome d'univalence de Voevodsky. C'est à cette occasion qu'a été dégagée, entre autres, la notion de *type inductif supérieur*, pierre angulaire de la formalisation de la théorie de l'homotopie en théorie des types homotopiques.

En 2012/2013 une année spéciale de recherche sur les fondations univalentes des mathématiques est organisée à l'Institute for Advanced Study (IAS) à Princeton par Vladimir Voevodsky, Steve Awodey et Thierry Coquand et réunira une cinquantaine de chercheurs qui travailleront ensemble sur des sujets proches des fondations univalentes, ce qui apportera très certainement de nombreuses avancées dans le domaine.

Je vais présenter ici les grandes lignes de ce domaine de recherche et je finirai par une description des problèmes ouverts importants et des avancées attendues lors de cette année de recherche à l'IAS.

1 Théorie du typage en informatique

La notion de typage est une notion qui est surtout utilisée en informatique. Cela consiste à classifier les données traitées par un programme selon leur *type* afin de rejeter les erreurs manifestes de programmation. Considérons par exemple une déclaration de type d'une fonction en C++ :

```
int f(int n)
```

Ce que signifie cette ligne est que `f` est une fonction qui prend en entrée un entier `n` et renvoie un entier. En particulier, si on essaye plus loin dans le programme d'écrire la ligne

```
x = f("a");
```

le code va être rejeté par le compilateur parce qu'une chaîne de caractères n'est pas un entier. De même, on ne pourra pas utiliser le résultat de `f(n)` à un endroit où l'on devait utiliser une fonction ou une chaîne de caractères.

Parmi les nombreux langages de programmation existants, il existe beaucoup de systèmes de types très différents. On va s'intéresser ici plus particulièrement à des langages ayant un système de types très expressif. Par exemple un langage de programmation comme OCaml est *fonctionnel*, c'est-à-dire que les fonctions sont des valeurs comme les autres, elles peuvent être données en argument à d'autres fonctions et une fonction peut renvoyer une autre fonction en sortie.

Enfin, des langages de programmation tels qu'Agda possèdent une notion de *types dépendants*, c'est-à-dire qu'il existe des types qui dépendent de termes (i.e. de programmes). Par exemple le programme ci-dessous définit le type des entiers naturels ainsi que le type `intlist n` des listes d'entiers de longueur `n` (où `n` est n'importe quel programme calculant un entier), ainsi qu'une fonction qui renvoie la tête d'une liste dont la longueur est de la forme `S n` (`S` est la fonction successeur). Le fait que la longueur de la liste est effectivement de la forme `S n` est vérifié statiquement par le compilateur, ce qui signifie que si le programme compile, le compilateur a réussi à démontrer que la fonction `head` est toujours appelée avec un argument de la forme `S n`.

L'absence de types dépendants en OCaml fait qu'on ne peut pas vérifier à la compilation si une liste donnée en argument à `head` est de longueur nulle ou non et on est obligé d'avoir recours à des exceptions à la place.

```
data ℕ : Set where
  0 : ℕ
  S : ℕ → ℕ

data intlist : ℕ → Set where
  nil : intlist 0
  cons : {n : ℕ} → ℕ → intlist n → intlist (S n)

head : ∀ {n : ℕ} (intlist (S n)) → ℕ
head (cons h t) = h
```

2 Correspondance de Curry–Howard

Considérons un langage de programmation fonctionnel particulier (qui s'appelle le *λ-calcul simplement typé*) comprenant un certain nombre de types de base $A, B, \ldots$, et tel que si X et Y sont deux types, on a un type $X \to Y$ correspondant au type des fonctions de X vers Y. Par exemple on a le type $A \to ((A \to B) \to B)$ correspondant aux fonctions prenant un élément de type A en argument et renvoyant un élément de type $(A \to B) \to B$, ce dernier étant une fonction prenant une fonction de type $A \to B$ en argument et renvoyant un élément de type B.

Ce langage possède deux constructions : l'abstraction et l'application. La règle d'abstraction dit que le terme $(x \mapsto u)$ est de type $X \to Y$ si u est une expression (un terme) faisant intervenir x et de type Y lorsque x est considéré comme étant de type X, et la règle d'application dit si f est de type $X \to Y$ et x de type X, alors $f(x)$ est de type Y.

Il est alors facile de construire un terme de type $A \to ((A \to B) \to B)$:

$$(x \mapsto (f \mapsto f(x)))$$

Par contre, sans informations supplémentaires sur A et B, il n'est pas possible de trouver de terme de type $A \to B$.

Considérons maintenant que A et B sont des propositions logiques et que la flèche représente l'implication logique. On remarque alors que la proposition $A \to ((A \to B) \to B)$ est démontrable alors que $A \to B$ ne l'est pas en général. Plus généralement, étant donné un type X, on peut montrer qu'il existe un terme de type X si et seulement si la proposition associée à X est démontrable en logique implicative minimale.

En effet, la logique implicative minimale est une logique ayant l'implication pour seul connecteur logique et ses règles de déduction sont les suivantes :

- Si B est vrai sous l'hypothèse que A est vrai, alors $A \to B$ est vrai
- Si $A \to B$ et A sont vrais, alors B est vrai

Et on remarque que ces règles de déduction correspondent exactement aux règles de typage du λ-calcul simplement typé, c'est-à-dire qu'un terme de type X n'est rien d'autre qu'une démonstration de X annotée, dans le sens où l'on nomme les hypothèses et qu'on garde une trace de l'endroit où est utilisée chaque hypothèse.

Un autre exemple est le suivant : considérons la proposition $A \to (A \to A)$. Alors cette proposition admet *deux* démonstrations distinctes, représentées respectivement par les fonctions $(x \mapsto (y \mapsto x))$ et $(x \mapsto (y \mapsto y))$.

Cette correspondance entre preuves et programmes est connue sous le nom de *correspondance de Curry–Howard* et peut être résumée dans le tableau suivant :

type	proposition
programme	démonstration
fonction	implication

Une autre façon de le voir est de dire que démontrer une implication $A \to B$ revient à construire une fonction associant une démonstration de B à toute démonstration de A.

Cette correspondance ne s'arrête pas à la logique implicative minimale. En effet, à tout connecteur logique correspondra un type. Par exemple à la conjonction va correspondre le type produit (une démonstration de $A \wedge B$ est la même chose qu'un couple formé d'une démonstration de A et d'une démonstration de B) et à la disjonction va correspondre le type somme (une démonstration de $A \vee B$ est soit une démonstration de A soit une démonstration de B).

Encore une fois, les règles de typage correspondantes vont correspondre exactement aux règles de déduction de la logique en question, donc on en déduit que démontrer une proposition X revient essentiellement à écrire un programme ayant le type X.

3 Types dépendants

Les types dépendants évoqués plus haut ont également une interprétation selon la correspondance de Curry–Howard. Ils permettent de gérer les propositions dépendants d'objets mathématiques. Par exemple la proposition $n = m$ est une proposition dépendant de deux entiers n et m, il va lui correspondre un type dépendant de deux termes n et m.

Soit A un type et $B(x)$ un type dépendant d'un terme x de type A.

On peut alors considérer un type noté $\Pi x^A.B(x)$ (produit dépendant) qui correspond au type des fonctions prenant un élément x de type A en argument et renvoyant un élément de type $B(x)$, et un type $\Sigma x^A.B(x)$ (somme dépendante) qui correspond au type des paires (u, v) où u est de type A et v est de type $B(u)$.

Ces deux types correspondent respectivement aux quantificateurs universel et existentiel. En effet, sous l'interprétation des preuves comme programmes, démontrer $\forall x \in A, B(x)$ revient à trouver une fonction qui à tout élément de A associe une démonstration de $B(x)$, et démontrer $\exists x \in A, B(x)$ revient à trouver une paire (u, v) où u est de type A et v est une preuve que $B(u)$ est vrai (c'est-à-dire que v est de type $B(u)$).

Remarquons qu'on ne fait pas la distinction entre les termes représentant des objets mathématiques et les termes représentant une démonstration d'une proposition. En effet, dans cette théorie, les démonstrations sont considérées comme des objets mathématiques à part entière au même titre qu'un entier naturel ou qu'une variété différentielle est un objet mathématique.

4 Règles d'introduction, élimination, réduction

En théorie des types, définir un nouveau type consiste à postuler son existence ainsi que celle des constructeurs de termes associés, ainsi qu'à imposer certaines règles de typage. Les règles de typage sont principalement de trois sortes :

Les règles d'introduction pour un type A spécifient les manières canoniques de produire un élément de type A.

Par exemple la règle d'introduction pour $\Pi x^A.B(x)$ dit que la manière canonique de produire une fonction de type $\Pi x^A.B(x)$ est de considérer la fonction $(x \mapsto u)$ où u est une expression (un terme) de type $B(x)$ faisant intervenir x. Pour $\Sigma x^A.B(x)$, la manière canonique de produire un élément de ce type est la formation d'une paire (u, v) où u est de type A et v de type $B(u)$.

Les règles d'élimination pour un type A spécifient les manières canoniques d'utiliser un élément de type A.

Par exemple pour $\Pi x^A.B(x)$, la règle d'élimination dit que si f est de type $\Pi x^A.B(x)$ et u est de type A, alors $f(u)$ est de type $B(u)$. Pour $\Sigma x^A.B(x)$ on a deux règles d'élimination, la première dit que si u est de type $\Sigma x^A.B(x)$, alors $\pi_1(u)$ est de type A, et la deuxième dit que si u est de type $\Sigma x^A.B(x)$, alors $\pi_2(u)$ est de type $B(\pi_1(u))$.

Enfin, les règles d'introduction et d'élimination sont reliées par des règles de β-réduction qui spécifient ce qu'il se passe quand on utilise une règle d'élimination immédiatement après une règle d'introduction.

Pour $\Pi x^A.B(x)$, la règle de réduction dit que $(x \mapsto u)(v)$ se réduit en $u[x := v]$ et pour $\Sigma x^A.B(x)$ les deux règles de réduction disent que $\pi_1(u, v)$ se réduit en u et que $\pi_2(u, v)$ se réduit en v.

5 Types inductifs

De nombreux autres types peuvent être définis en utilisant la notion de *type inductif*. Un type inductif consiste à définir un type en spécifiant uniquement ses règles d'introduction et en respectant certaines contraintes syntaxiques. Ses règles d'élimination et de réduction s'en déduiront alors automatiquement.

Par exemple on peut définir le type des entiers naturels en tant que type inductif de la manière suivante :

$$\begin{aligned}\mathbb{N} ::= &| \ \mathsf{0} : \mathbb{N} \\ &| \ \mathsf{S}(n : \mathbb{N}) : \mathbb{N}\end{aligned}$$

Cela signifie qu'un entier naturel est soit égal à $\mathsf{0}$, soit égal à $\mathsf{S}(n)$ pour n un autre entier naturel. La règle d'élimination déduite correspond alors à la règle de construction par récurrence (et la règle de raisonnement par récurrence en est un cas particulier lorsque ce que l'on construit est une démonstration).

On peut de même définir le type des entiers relatifs ou la somme de deux types de la façon suivante :

$$\begin{aligned}\mathbb{Z} ::= &| \ \mathsf{0} : \mathbb{Z} \\ &| \ \mathsf{pos}(n : \mathbb{N}) : \mathbb{Z} \\ &| \ \mathsf{neg}(n : \mathbb{N}) : \mathbb{Z}\end{aligned}$$

$$\begin{aligned}A + B ::= &| \ \mathsf{inl}(a : A) : A + B \\ &| \ \mathsf{inr}(b : B) : A + B\end{aligned}$$

6 Univers

Il peut être utile de pouvoir manipuler les types au même titre que des termes, par exemple afin de pouvoir définir un type par récurrence.

On utilise pour cela la notion d'*univers*. Un univers est un type U tel que tout terme A de type U peut être vu lui-même comme un type. En d'autres termes, on voit U comme le type des petits types.

Ainsi, si on veut définir un (petit) type dépendant d'un entier $n : \mathbb{N}$, il suffit de définir une fonction $\mathbb{N} \to \mathsf{U}$ correspondant à ce type dépendant, et cette fonction peut être définie par récurrence si besoin.

7 Égalité propositionelle

Une notion très importante en mathématiques et dont on n'a pas encore parlé est la notion d'égalité. Étant donné deux éléments x et y d'un ensemble A, on peut considérer la proposition $x = y$ qui est vraie si et seulement si x et y sont égaux. D'après la correspondance de Curry–Howard, il devrait y correspondre un type.

Plus précisément, il est possible de définir une famille de types (appelés types identité) qu'on notera $x \equiv_A y$, pour A un type et x et y deux termes de type A, de telle sorte qu'un terme de type $x \equiv_A y$ corresponde (intuitivement) à une démonstration de l'égalité $x = y$.

On définit $x \equiv_A y$ en tant que famille inductive de types engendrée par le constructeur $\mathsf{refl}_A(u)$ de type $u \equiv_A u$. Autrement dit, la seule façon canonique de construire un terme de type $x \equiv_A y$ (c'est-à-dire de démontrer que x est égal à y) est d'utiliser la règle de réflexivité lorsque x est (syntaxiquement) égal à y.

De même que pour les types inductifs, on a alors une règle d'élimination qui dit essentiellement que toute démonstration de $x \equiv_A y$ peut se ramener à une démonstration de la forme $\mathsf{refl}_A(u)$ où x et y sont syntaxiquement égaux (à u).

Par exemple pour démontrer la proposition $x = y \Rightarrow f(x) = f(y)$ où $x, y : A$ et $f : A \to B$, on veut trouver un terme de type $x \equiv_A y \to f(x) \equiv_B f(y)$. On suppose donc qu'on a un terme de type $x \equiv_A y$ et on veut fabriquer un terme de type $f(x) \equiv_B f(y)$. D'après la règle d'élimination, le terme de type $x \equiv_A y$ peut se ramener à un terme de la forme $\mathsf{refl}_A(u)$ où x et y sont syntaxiquement égaux à u, et il reste à montrer $f(u) \equiv_B f(u)$ ce qui est vrai par réflexivité car $f(u)$ est syntaxiquement égal à $f(u)$.

8 Interprétation homotopique

Étant donné que le seul constructeur du type $x \equiv_A y$ est $\mathsf{refl}_A(x)$ pour x et y syntaxiquement égaux, on pourrait s'attendre à ce que si p et q sont deux termes de type $x \equiv_A y$, alors il existe nécessairement un terme de type $p \equiv_{x \equiv_A y} q$, témoignant du fait que p et q sont égaux (on parle d'*unicité des preuves d'égalité*). Remarquons en passant que les types identité pouvant s'appliquer à n'importe quel type A, il est tout à fait autorisé de l'appliquer à un autre type identité, ce qui revient à comparer deux témoins d'égalité de deux points.

Or, en 1995 Hofmann et Streicher ont construit un modèle de la théorie des types dans lequel les types identité n'ont pas cette propriété. Plus précisément, leur modèle interprète un type comme un groupoïde (un groupoïde est une catégorie dont toutes les flèches sont des isomorphismes) et le type $x \equiv_A y$ (où A est un groupoïde et x et y deux objets de A) est interprété par l'ensembles des isomorphismes de x vers y. Or il est facile de trouver un groupoïde A et deux objets x et y tels qu'il existe plusieurs isomorphismes différents entre x et y.

Une dizaine d'années plus tard, Steve Awodey, Michael Warren et d'autres ont remarqué qu'il était également possible d'interpréter les types identité dans le cadre beaucoup plus général des *systèmes de factorisation faibles* utilisés en théorie de l'homotopie abstraite, et en particulier dans la catégorie des espaces topologiques où les types sont interprétés par des espaces topologiques et où le type $x \equiv_A y$ est interprété par l'espace des chemins continus entre x et y (cf [AW09] et [GB10]). Ce que dit ce résultat est qu'il est cohérent de considérer deux points d'un espace topologique comme étant « égaux » s'ils sont reliés par un chemin, le chemin étant vu comme le témoin de cette égalité.

9 L'axiome d'univalence

En théorie des ensembles, la notion d'égalité entre deux ensembles quelconques est une notion beaucoup trop stricte. On a souvent besoin d'identifier deux ensembles qui ne sont pas égaux mais simplement en bijection. Cependant, toujours en théorie des ensembles, on ne peut pas pour autant identifier deux ensembles en bijection dans tous les cas, car certaines propriétés ne sont pas préservées par isomorphisme. Par exemple les ensembles $A = \{0, 1\}$ et $B = \{0, 2\}$ sont en bijection mais $1 \in A$ et $1 \notin B$.

En théorie des types, on a également une notion similaire d'équivalence. On dit qu'une flèche $f : A \to B$ est une *équivalence* si on a une flèche $g : B \to A$ et deux termes $p : \Pi x^A.g(f(x)) \equiv_A x$ et $q : \Pi y^B.f(g(y)) \equiv_B y$ qui témoignent du fait que g est un inverse de f. Sous l'interprétation homotopique des types identité, une flèche est une équivalence si et seulement si c'est une équivalence d'homotopie.

D'autre part, si A et B sont deux types d'un univers U on peut considérer le type $A \equiv_{\mathsf{U}} B$ des témoins d'égalité entre A et B. De même qu'en théorie des ensembles, cette relation d'égalité est a priori trop stricte, deux types équivalents n'ont pas de raison d'être égaux.

Cependant, contrairement à ce qu'il se passe en théorie des ensembles, la notion d'équivalence se comporte bien : il n'est pas possible d'énoncer une propriété qui ne soit pas préservée par

équivalence car les seules propriétés que l'on peut énoncer sont celles qui sont bien typées, et une propriété bien typée est toujours invariante par équivalence.

Vladimir Voevodsky a ainsi proposé en 2009 un axiome à rajouter à la théorie des types, l'*axiome d'univalence*, qui identifie le type $A \equiv_{\mathsf{U}} B$ au type de toutes les équivalences entre A et B. Plus précisément, si U est un univers et A et B sont deux types dans U, il y a une application canonique du type $A \equiv_{\mathsf{U}} B$ dans le type $\mathrm{Eq}(A, B)$ des équivalences entre A et B. On dit alors qu'un univers U satisfait l'axiome d'univalence s'il satisfait la propriété suivante :

Axiome 1 *(Axiome d'univalence)*
Pour tous types A et B dans U, l'application canonique de $A \equiv_U B$ vers $\mathrm{Eq}(A, B)$ est une équivalence.

En particulier, l'axiome d'univalence dit que si deux structures sont isomorphes, alors elles sont en un certain sens *égales*.

10 Fondations univalentes et formalisation des mathématiques

L'axiome d'univalence a été proposé dans le cadre des fondations univalentes des mathématiques, un programme de recherche dont le but est de refonder l'intégralité des mathématiques en utilisant la théorie des types et l'axiome d'univalence comme base. Il y a plusieurs avantages à ces fondements par rapport à la théorie des ensembles.

Tout d'abord il s'agit de fondements bien plus proches de la pratique mathématique courante et prévus pour être utilisés en tant que tels, contrairement à la théorie des ensembles qui avait pour principal but de montrer qu'il est possible en principe de faire reposer l'intégralité des mathématiques sur des bases purement formelles. Il y a certes beaucoup plus de règles de typage en théorie des types que d'axiomes en théorie des ensembles, mais on obtient ainsi une théorie bien plus structurée et dans laquelle il est syntaxiquement impossible d'écrire quelque chose d'absurde. En théorie des ensembles tout est un ensemble, donc rien ne nous empêche a priori de nous demander si $\exp \in \sqrt{2}$ est vrai. En théorie des types, exp est de type $\mathbb{R} \to \mathbb{R}$ et $\sqrt{2}$ est de type $\mathbb{R}$, donc $\exp \in \sqrt{2}$ n'est pas bien typé et on n'a donc même pas le droit de se demander si cette proposition est vraie ou non.

Un autre grand avantage d'avoir des fondements basés sur la théorie des types est la connection avec l'informatique. En effet, étant donné que vérifier une démonstration d'une proposition revient à vérifier qu'un certain programme (correspondant à la démonstration) est bien typé, cette vérification peut être entièrement automatisée et effectuée par ordinateur. On peut de plus aller plus loin et assister l'utilisateur dans la construction de la démonstration en permettant la construction pas à pas de la démonstration, en automatisant les démonstrations faciles, etc.

Un tel logiciel s'appelle un *assistant de preuves* et plusieurs assistants de preuves existent déjà et sont utilisés par un nombre croissant de personnes. Les deux assistants de preuves utilisés plus particulièrement en théorie des types homotopiques sont Coq, développé en France à l'INRIA, et Agda, développé en Suède à l'université de Chalmers. Ils sont tous deux basés sur des théories assez similaires et il est facile d'y rajouter l'axiome d'univalence.

Vladimir Voevodsky a déjà commencé à utiliser ces fondations pour formaliser dans Coq de nombreux résultats sur les ensembles finis, les entiers, les structures algébriques de base comme les groupes et les anneaux, etc.

11 Quotients

La notion de quotient d'un ensemble par une relation d'équivalence est quelque chose de traditionellement difficile à définir en théorie des types. Jusqu'à présent, cette difficulté était contournée en utilisant des *setoïdes* (un setoïde étant simplement un type muni d'une relation d'équivalence) et en ne considérant que des applications entre setoïdes qui respectent les relations d'équivalence.

En théorie des types homotopiques, il devient possible d'obtenir de vrais quotients, et il y en a même au moins trois définitions différentes. La première définition reproduit la définition usuelle en terme de classes d'équivalences et nécessite une forme faible de l'axiome d'univalence ainsi que des règles de redimensionnement (dont on parlera un peu plus loin) pour avoir de bonnes propriétés. La deuxième revient à définir directement le quotient par sa propriété universelle et nécessite également l'utilisation de l'axiome d'univalence et de plusieurs règles de redimensionnement. Enfin la troisième définition consiste à utiliser un type inductif supérieur (voir plus loin) pour rajouter artificiellement des égalités entre les éléments équivalents, et aucune règle de redimensionnement n'est nécessaire.

12 Constructivité

Une particularité notoire de la théorie des types utilisée ici est que l'on obtient une logique constructive par défaut. Cela signifie que par exemple si on considère un terme u de type $\Sigma x^A.B(x)$, c'est-à-dire une démonstration du fait qu'il existe un x tel que $B(x)$ soit vérifié, on peut voir u comme un programme et l'exécuter (en appliquant répétitivement les règles de réduction). On peut alors montrer que l'exécution s'arrête toujours au bout d'un nombre fini d'étapes et que le résultat final obtenu est de la forme (t, v) où t est de type A et v de type $B(t)$. En particulier on a la propriété du témoin : si on a démontré qu'il existe un x vérifiant $B(x)$, on peut extraire explicitement un tel x de la démonstration. Cette propriété est complètement fausse en logique classique à cause de la possibilité d'utilisation du raisonnement par l'absurde.

Les théorèmes que l'on pourra démontrer devront ainsi être valables en logique intuitionniste et les théorèmes nécessitant le tiers exclu ou l'axiome du choix devront être reformulés pour prendre en compte cette contrainte. Cela peut sembler être un inconvénient, mais c'est également une occasion pour comprendre plus en profondeur les théorèmes mathématiques et savoir quelles sont les parties purement explicites et constructives et quelles sont les parties nécessitant l'axiome du choix ou le raisonnement par l'absurde.

13 Types inductifs supérieurs

On a vu plus haut que l'interprétation homotopique de la théorie des types consiste à imaginer que tout type est un espace topologique et que tout terme de type $x \equiv_A y$ est un chemin entre x et y. Mais ceci ne nous explique pas comment construire des espaces topologiques usuels comme les sphères, les espaces projectifs, etc. On va utiliser pour cela la notion de *type inductif supérieur*.

La notion de type inductif supérieur est similaire à celle de type inductif présentée plus haut, à la différence près que les constructeurs peuvent désormais construire des chemins, c'est-à-dire des éléments de type $x \equiv_A y$ où A est le type inductif supérieur que l'on est en train de définir. On peut par exemple définir le cercle de la façon suivante :

$$\begin{aligned}\mathbb{S}^1 ::= &| \ \mathsf{e} : \mathbb{S}^1 \\ &| \ \mathsf{p} : \mathsf{e} \equiv_{\mathbb{S}^1} \mathsf{e}\end{aligned}$$

Autrement dit, le cercle est le type (l'espace) librement engendré par un point et un lacet en ce point.

De même que pour les types inductifs simples, les règles d'élimination et de réduction peuvent s'en déduire automatiquement.

On peut définir de nombreux autres espaces de cette manière, par exemples les sphères, les bouquets de sphères, les espaces projectifs, et bien d'autres. Essentiellement tous les espaces ayant une décomposition cellulaire « finiment descriptible » sont représentables par un type inductif supérieur

Cette notion d'être finiment descriptible mériterait d'être précisée, mais aucune description précise n'est connue actuellement. En particulier, un espace peut tout à fait avoir un nombre infini de cellules mais être tout de même finiment descriptible, par exemple $\mathbb{N}$, la sphère de dimension infinie ou les troncations sont descriptibles par des types inductifs supérieurs.

14 Formalisation de la théorie de l'homotopie

L'utilisation des types inductifs supérieurs ouvre la voie à l'utilisation d'assistants de preuves pour démontrer formellement des résultats de théorie de l'homotopie. Par exemple, Mike Shulman a démontré en avril 2011 que le type $\mathsf{e} \equiv_{\mathbb{S}^1} \mathsf{e}$ (correspondant à l'espace des lacets du cercle) est équivalent au type $\mathbb{Z}$ des entiers relatifs, démontrant ainsi que tous les groupes d'homotopie du cercle sont ceux auxquels on s'attend. Cette démonstration est une démonstration entièrement formelle écrite avec l'assistant de preuves Coq. Elle a depuis été réécrite en Agda par plusieurs personnes, en particulier par Daniel Licata et moi-même (indépendamment).

L'élément clé de la démonstration est l'utilisation de l'axiome d'univalence pour construire le revêtement universel du cercle : il est facile de définir l'application successeur de $\mathbb{Z}$ vers $\mathbb{Z}$ et de montrer que c'est une équivalence, donc d'après l'axiome d'univalence il lui correspond un lacet non trivial $p_{\mathsf{succ}} : \mathbb{Z} \equiv_{\mathsf{U}} \mathbb{Z}$ dans l'univers U (en supposant que $\mathbb{Z}$ est dans un univers U). On peut alors en déduire une application $F : \mathbb{S}^1 \to \mathsf{U}$ définie par $F(\mathsf{e}) = \mathbb{Z}$ et $F(\mathsf{p}) = p_{\mathsf{succ}}$. Cette application peut être vue comme un type dépendant $F(x)$ au dessus de $\mathbb{S}^1$ et le type $\Sigma x^{\mathbb{S}^1}.F(x)$ correspond alors intuitivement au revêtement universel de $\mathbb{S}^1$. Il suffit ensuite de démontrer que l'espace que l'on vient de construire est contractile (et qu'il s'agit donc bien du revêtement universel du cercle) et on peut ensuite en déduire le résultat en le comparant à la fibration des chemins du cercle.

15 Questions ouvertes, pistes de recherche

L'année de recherche à l'Institute for Advanced Study en 2012/2013 est organisée selon quatre axes majeurs :

Conception et implémentation d'un assistant de preuves basé sur un système de types ayant du polymorphisme d'univers et les règles de redimensionnement Les règles de redimensionnement sont des règles de typage permettant de diminuer le niveau d'univers d'un type. En général, on suppose qu'on a une infinité d'univers $\mathsf{U}_0, \mathsf{U}_1, \ldots, \mathsf{U}_n, \ldots$ et il est possible de voir un type de l'univers U_n comme étant dans l'univers U_{n+k}, mais pas réciproquement.

Les règles de redimensionnement spécifient que sous certaines conditions sur un type $A : \mathsf{U}_n$, il est possible de considérer A comme étant dans un univers inférieur (voire dans U_0). Ces règles sont utiles pour la formalisation des mathématiques et ont été utilisées à de nombreuses reprises par Vladimir Voevodsky, mais elles n'ont encore quasiment pas été étudiées théoriquement et aucun assistant de preuves ne permet de les utiliser d'une façon propre.

Le polymorphisme d'univers permet quant à lui de définir un type d'une façon paramétrique en le niveau d'univers. Par exemple le type $\mathrm{Eq}(A, B)$ des équivalences entre A et B n'est a priori défini que lorsque A et B appartiennent à un univers fixé U_ℓ, et si on veut l'utiliser pour $A, B : \mathsf{U}_{\ell+1}$ il faut le redéfinir pour l'univers $\mathsf{U}_{\ell+1}$. Le polymorphisme d'univers permet de définir $\mathrm{Eq}(A, B)$ pour tous les univers simultanément. Coq ne possède pas de polymorphisme d'univers et Agda a une forme de polymorphisme d'univers très puissante (mais peut-être trop puissante, personne n'a (à ma connaissance) étudié théoriquement le polymorphisme d'univers d'Agda en conjonction avec l'axiome d'univalence, et il se peut que les deux ne soient pas compatibles).

L'objectif principal ici est donc la conception d'un nouveau système de types ayant ces propriétés, ainsi que la conception d'un nouvel assistant de preuves basé sur ce système de types.

Interprétation calculatoire de l'axiome d'univalence On a dit plus haut que la théorie des types utilisée ici est constructive. Or, l'ajout de l'axiome d'univalence détruit complètement cette propriété. Il est facile de trouver par exemple un terme de type $\mathbb{N}$ qu'il n'est pas possible de réduire en un vrai entier. La plupart des gens s'attendent cependant à ce que l'axiome d'univalence ait un contenu calculatoire, plus précisément on a la conjecture suivante :

Conjecture 1 *Il existe un algorithme qui, étant donné un terme $u : \mathbb{N}$ utilisant l'axiome d'univalence lui associe un terme $\widetilde{u} : \mathbb{N}$ n'utilisant pas l'axiome d'univalence ainsi qu'un terme de type $u \equiv_{\mathbb{N}} \widetilde{u}$ (c'est-à-dire une preuve que u et $\widetilde{u}$ sont égaux).*

L'objectif est de tenter de résoudre cette conjecture, en particulier en recherchant des algorithmes permettant de supprimer l'utilisation de l'axiome d'univalence.

Formalisation de structures de théorie de l'homotopie Les types inductifs supérieurs évoqués plus haut sont très puissants dans la construction de structures homotopiques, mais ils sont encore assez mal compris et peu de choses ont été formalisées avec.

Comme pour l'axiome d'univalence, un des principaux problèmes avec les types inductifs supérieurs est leur interprétation calculatoire. En utilisant des types inductifs supérieurs, il est facile de construire un terme de type $\mathbb{N}$ qui ne peut pas se réduire en un vrai entier. L'idée est de rajouter les règles de réduction manquantes, mais l'explicitation de ces règles de réduction s'avère très compliquée et seuls des résultats très partiels et nécessitant de grands changements dans la théorie sous-jacente sont disponibles actuellement (cf [LH11]).

D'autre part, beaucoup de définitions et de résultats de la théorie de l'homotopie semblent formalisables en utilisant des types inductifs supérieurs, comme le calcul de certains groupes d'homotopie des sphères, la définition de la cohomologie des espaces, la définition des catégories supérieures, etc. Cependant, le faible nombre de personnes travaillant avec des types inductifs supérieurs et le fait que l'écriture de preuves formelles prenne beaucoup plus de temps que l'écriture de preuves mathématiques informelles font qu'il reste encore beaucoup de choses à explorer.

Relations entre fondations univalentes et théorie des ensembles La plupart des mathématiciens travaillent actuellement en théorie des ensembles, donc il est important de mettre

en relation la théorie des ensembles avec les fondations univalentes pour que les résultats puissent se transférer d'un côté à l'autre.

Ceci se fait au moyen de modèles de la théorie des types : un modèle de la théorie des types est une catégorie munie de structures imitant la structure syntaxique présente dans la théorie des types. Par exemple si la théorie des types considérée contient des types produits, il faudra que la catégorie possède également des produits (au sens catégorique du terme). Une question importante qui se pose alors est de savoir quelles sont les structures nécessaires sur une catégorie pour pouvoir interpréter la théorie des types homotopiques (avec l'axiome d'univalence).

Jusqu'en mars 2012, le seul modèle connu de l'axiome d'univalence était la catégorie des ensembles simpliciaux (une sorte de variante algébrique des espaces topologiques). En mars 2012, Mike Shulman a construit toute une famille de nouveaux modèles ([Sh12]) et en août 2012, David Gepner et Joachim Kock ont généralisé ce résultat en montrant qu'on peut modéliser l'axiome d'univalence dans tout ∞-*topos* de Lurie ([GK12]). La notion d'∞-topos de Lurie est une notion très technique mais l'idée générale est qu'un ∞-topos est une catégorie qui « ressemble » à la catégorie des espaces topologiques à homotopie près, et ce que dit le résultat de Gepner et Kock est que cette ressemblance est suffisante pour conserver l'axiome d'univalence et toutes les conséquences qui en découlent.

Cependant la réciproque n'est pas vraie, il existe des modèles de la théorie des types et de l'axiome d'univalence qui ne sont pas des ∞-topos de Lurie, mais les conditions précises n'ont pas encore été découvertes. Une conjecture est qu'il existe une notion d'∞-*topos élémentaire* qui étend la notion d'∞-topos de Lurie et telle qu'une catégorie est un modèle de la théorie des types homotopiques si et seulement si cette catégorie est un ∞-topos élémentaire.

16 Intérêts personnels

Pour conclure, mes intérêts personnels se situent principalement dans la formalisation pratique de résultats de théorie de l'homotopie. Durant cette année j'ai formalisé un certain nombre de résultats dans l'assistant de preuves Agda et le code source de ce que j'ai fait est disponible sur Internet à l'adresse suivante :

`https://github.com/guillaumebrunerie/HoTT`

Parmi ceux-ci on peut citer

- La réimplémentation de la librairie standard de la théorie des types homotopiques, c'est-à-dire tous les résultats de base concernant les types identité, les équivalences, l'axiome d'univalence, etc. et qui sont ensuite utilisés constamment dans toutes les autres démonstrations
- Une démonstration du fait que l'espace des lacets du cercle est équivalent au type des entiers. Ma démonstration est essentiellement la même que celle de Mike Shulman, mis à part un passage technique que je gère d'une façon moins ad-hoc et plus généralisable
- Une définition des troncations. Pour tout type A et tout entier n je définis un type $\tau_{\leq n}A$ correspondant à A où l'on a tué tous les groupes d'homotopie au dessus de n, et j'ai démontré sa propriété universelle
- Une définition du groupe libre sur un ensemble quelconque. Cette définition utilise un quotient implémenté grâce à la troncation
- Une démonstration du fait que l'espace des lacets d'un bouquet d'un ensemble A de cercles est équivalent au groupe libre sur A. Une restriction importante à imposer est que l'égalité sur A soit *décidable*, c'est-à-dire qu'il soit vrai que pour tous $x, y : A$, soit x et y sont égaux soit il sont différents. Cette propriété qui est toujours vraie en logique classique peut ne

pas l'être en logique intuitionniste et ma démonstration ne fonctionne pas si elle n'est pas vérifiée. Je ne sais pas si le résultat est vrai dans le cas où l'égalité n'est pas décidable. Le calcul de l'espace des lacets d'un bouquet de cercle a également été fait indépendamment par Kuen-Bang Hou (favonia) et Chris Kapulkin dans le cas d'un nombre fini de cercles

J'ai également démontré que la sphère de dimension infinie est contractile il y a plus longtemps (en Coq) et j'ai plusieurs autres démonstrations en cours, dont en particulier une démonstration du fait que $\pi_i(\mathbb{S}^k)$ est trivial pour tout $i < k$ (il me reste uniquement un lemme un peu technique à démontrer) et une démonstration du fait que $\pi_2(\mathbb{S}^2)$ est isomorphe à $\mathbb{Z}$ (il me manque encore plusieurs résultats, mais j'ai une démonstration avec les mains qui me semble correcte et formalisable).

Pour l'avenir proche, j'aimerais pouvoir finir ces deux démonstrations et travailler sur d'autres problèmes :

- Démontrer que $\pi_n(\mathbb{S}^n)$ est isomorphe à $\mathbb{Z}$ pour tout n. Ceci devrait permettre de construire les espaces d'Eilenberg Mac-Lane $K(\mathbb{Z}, n)$ et d'en déduire la définition de la cohomologie des espaces et on pourrait alors essayer de démontrer les propriétés usuelles de la cohomologie
- D'autre part, je pense avoir une définition de 1-catégorie intrinsèquement homotopique, c'est-à-dire pour laquelle deux 1-catégories équivalentes sont indiscernables. En particulier, toute construction catégorique effectuée dans ce cadre serait alors automatiquement invariante par équivalence. Je n'ai pas encore eu le temps de formaliser cette notion
- Enfin, d'un côté plus pratique, j'aimerais également aider à la conception d'un assistant de preuves permettant de formaliser efficacement les résulats souhaités. La pratique de la formalisation de preuves en Agda m'a permis de voir plusieurs points sur lesquels une certaine automatisation serait possible et simplifierait grandement les démonstrations

Références

[AW09] S. Awodey and M. Warren, *Homotopy theoretic models of identity types*, Mathematical Proceedings of the Cambridge Philosophical Society, 2009
`http://www.andrew.cmu.edu/user/awodey/preprints/homotopy.pdf`

[GB10] R. Garner and B. van den Berg, *Topological and simplicial models of identity types*, ACM Transactions on Computational Logic Volume 13 Issue 1, January 2012
`http://tocl.acm.org/accepted/467garner.pdf`

[HS95] M. Hofmann and T. Streicher, *The groupoid interpretation of type theory*, Oxford Logic Guides, vol. 36, Oxford Univ. Press, New York, 1998, pp. 83–111.

[LH11] D. Licata and R. Harper, *Canonicity for 2-Dimensional Type Theory*
`http://www.cs.cmu.edu/~drl/pubs/lh112tt/lh112tt.pdf`

[Sh12] M. Shulman, *The univalence axiom for inverse diagrams*
`http://arxiv.org/pdf/1203.3253v1.pdf`

[GK12] D. Gepner and J. Kock, *Univalence in locally cartesian closed ∞-categories*
`http://arxiv.org/pdf/1208.1749v1.pdf`

A Exemple d'interaction avec l'assistant de preuves Agda

L'assistant de preuves Agda s'utilise dans l'éditeur de texte Emacs. Il est fourni avec un mode majeur pour Emacs (un plugin) qui permet l'édition interactive du code. Ce mode permet également d'écrire des caractères unicode directement en utilisant des suites de touches similaires à des commandes LaTeX. Par exemple taper la suite de touches `\to` insère le caractère →, et la suite de touches `\bn` affiche le caractère ℕ.

On va montrer comment définir les entiers relatifs, les fonctions successeur et prédécesseur, et montrer qu'elles sont inverses l'une de l'autre.

On commence par définir les entiers naturels et relatifs. Le mot-clé `data` permet de définir des types inductifs et `Set` représente le type de tous les types (un univers).

```
data ℕ : Set where
  0 : ℕ
  S : ℕ → ℕ

data ℤ : Set where
  0 : ℤ
  pos : ℕ → ℤ
  neg : ℕ → ℤ
```

On va ensuite définir la fonction successeur interactivement. On commence par écrire les lignes suivantes. La première ligne donne son type et le point d'interrogation sur la deuxième ligne indique qu'on n'a pas encore défini le terme qui doit remplacer le point d'interrogation.

```
succ : ℤ → ℤ
succ n = ?
```

La définition de la fonction successeur se fait par disjonction de cas sur la variable `n`. On place le curseur sur le point d'interrogation et on utilise le raccourci clavier `C-c C-c` qui permet de faire une disjonction de cas sur la variable indiquée par l'utilisateur (ici `n`). Le code est alors réécrit en le suivant :

```
succ : ℤ → ℤ
succ 0 = ?
succ (pos n) = ?
succ (neg n) = ?
```

Le premier point d'interrogation doit contenir le successeur de `0` et on sait qu'il s'agit de `pos 0`. On utilise le raccourci clavier `C-c C-SPC` qui permet de fournir un terme pour remplacer le point d'interrogation, et Agda accepte `pos 0` après avoir vérifié qu'il est bien typé. Pour `succ (pos n)` on entre de même le terme `pos (S n)`, et pour `succ (neg n)` on fait une nouvelle disjonction de cas sur `n` (qui est cette fois de type ℕ) et on rentre les deux termes qui conviennent.

```
succ : ℤ → ℤ
succ 0 = pos 0
succ (pos n) = pos (S n)
succ (neg 0) = 0
succ (neg (S n)) = neg n
```

On définit de même la fonction prédécesseur.

```
pred : ℤ → ℤ
pred 0 = neg 0
pred (pos 0) = 0
pred (pos (S n)) = pos n
pred (neg n) = neg (S n)
```

On définit ensuite le prédicat d'égalité par une famille inductive de types. Les tirets bas servent à définir des symboles infixes, cela signifie qu'on peut écrire `n ≡ m` au lieu de `_≡_ n m` pour le type des témoins d'égalité entre `n` et `m`. Le terme `refl n` est de type `n ≡ n` et correspond au témoin de la réflexivité de l'égalité.

```
data _≡_ : ℤ → ℤ → Set where
  refl : (n : ℤ) → (n ≡ n)
```

On souhaite ensuite prouver que la fonction successeur et la fonction prédécesseur sont inverses l'une de l'autre, c'est à dire qu'on veut construire un terme ayant le type suivant :

```
pred-succ : (n : ℤ) → pred (succ n) ≡ n
pred-succ n = ?
```

La fonction successeur étant définie par disjonction de cas sur `n`, on procède de la même manière (afin de pouvoir utiliser la définition de la fonction successeur) :

```
pred-succ : (n : ℤ) → pred (succ n) ≡ n
pred-succ 0 = ?
pred-succ (pos n) = ?
pred-succ (neg 0) = ?
pred-succ (neg (S n)) = ?
```

Lorsque le curseur est sur un point d'interrogation, le raccourci clavier `C-u C-c C-t` nous indique le type du terme qui doit remplir ce point d'interrogation. Pour le premier cas, on obtient `pred (succ 0) ≡ 0`. Or par définition `succ 0` est égal à `pos 0` et `pred (pos 0)` est égal à `0`. On peut aussi utiliser le raccourci clavier `C-c C-t` qui effectue automatiquement toutes les réductions possibles dans le type et nous indique que le type du point d'interrogation est en fait `0 ≡ 0`. Le résultat découle donc de la réflexivité de l'égalité, et on peut remplir le point d'interrogation par le terme `refl 0` qui est bien de type `0 ≡ 0`.

On peut également utiliser le raccourci clavier `C-c C-a` qui trouve automatiquement un terme quand il est « facile » de le déduire de son type. Dans le cas présent, utiliser ce raccourci clavier dans chacun des points d'interrogation insère automatiquement le terme `refl x` pour le `x` qui convient.

```
pred-succ : (n : ℤ) → pred (succ n) ≡ n
pred-succ 0 = refl 0
pred-succ (pos n) = refl (pos n)
pred-succ (neg 0) = refl (neg 0)
pred-succ (neg (S n)) = refl (neg (S n))
```

On peut de la même façon construire un terme `succ-pred` de type

```
succ-pred : (n : ℤ) → succ (pred n) ≡ n
```

ce qui démontre que les fonctions prédécesseur et successeurs sont inverses l'une de l'autre.