Travail d'Étude et de Recherche

Introduction à la théorie des types et aux assistants de preuves.

Dorian Guillet

Encadré par Vincent Beffara.

Mai 2025.

Institut Fourier – M1 Mathématiques Générales

Table des matières

Introduction

L'objectif du TER est de découvrir un assistant de preuve en formalisant un résultat non trivial, puis de comprendre les fondements théoriques sur lesquels reposent les assistants de preuves actuels comme Lean ou Rocq. Pour ce faire, ce rapport est décomposé en trois parties distinctes.

La première partie est une présentation de la théorie des types (dépendants), dans la version introduite par Per Martin-Löf [Mar21]. Cette théorie introduit la notion fondamentale de *type* qui est un analogue des ensembles, et est largement utilisée par de nombreux langages de programmation. Elle permet de formaliser les mathématiques et de construire un lien entre informatique et mathématiques en étendant la correspondance de Curry-Howard entre les preuves et les programmes. On décrira ici seulement les règles structurant les relations entre les types et décrivant le système de preuve qu'il fournit, et enfin les liens entre les preuves et les programmes et par extension les assistants de preuve comme Lean.

La deuxième partie décrit la formalisation d'un théorème à l'aide de l'assistant de preuve Lean. Pour ce faire, on présente les outils qui ont été utilisés dans le cadre de la formalisation et les obstacles rencontrés. Initialement, le théorème choisi pour être formalisé établissait l'existence de mesure dite de Gibbs. Cependant la preuve dûe à Rufus Bowen [Bow75] était trop longue pour être complètement portée en Lean. Le choix final a été de montrer rigoureusement un de ses lemmes essentiels. On présente également dans cette section les reformulations des énoncés qui ont été nécessaires pour formaliser ces résultats en utilisant les types présents dans Lean.

La dernière partie est une preuve du lemme finalement choisi. Ce lemme est utile dans le théorème de Bowen pour obtenir des estimations sur la mesure des ouverts d'un espace de Bernoulli, nécessaire pour obtenir l'unicité des mesures de Gibbs. On en donne ici une version plus générale, permettant d'obtenir une partition des ouverts bornés par des boules en utilisant des espaces ultramétriques et en prouvant également des résultats élémentaires sur ses espaces métriques particuliers.

I.

Théorie des Types

I.1 Introduction

La théorie des types se veut être une alternative à la théorie des ensembles de Zermelo-Fraenkel (ZF). Les fondations de cette dernière s'appuient sur le système déductif de la logique du premier ordre, la théorie des ensembles étant formulée dans ce système. Cette théorie est donc composée de deux couches : la logique du premier ordre, puis la théorie des ensembles, formée de ses axiomes. On a donc deux objets fondamentaux dans cette approche : les propositions (qui se basent sur la logique du premier ordre) et les ensembles de ZF.

Afin d'éviter cette construction en deux couches, la théorie des types possède son propre système déductif. Un système déductif est définit par des jugements, décrivant quelles affirmations peuvent être prouvées, et des règles qui donne la manière dont les jugements peuvent être dérivés les uns par rapport aux autres. Par conséquent, une fois les jugements et les règles du système déductif définies, il est prêt à être utilisé sans axiomes supplémentaires. Cette théorie ne possède qu'un objet fondamental : les **types**.

Pour pouvoir formuler des théorèmes, on a besoin de l'équivalent des propositions, qui sont ici des types, dont la construction suit des règles qui seront précisées dans les sections suivantes. Dans cette théorie, prouver un théorème est donc équivalent à construire un objet dont le type correspond au théorème (ici l'objet en question est une preuve).

On peut aussi voir les types d'un point de vue plus proche de la théorie des ensembles, en interprétant le fait qu'un élément a soit de type A comme l'affirmation $a \in A$. Cependant, l'affirmation $a \in A$ est une proposition alors que dire "a est de type A" (que l'on notera dorénavant $a : A$) est un jugement. En effet, dans la théorie des types un élément possède toujours un type déterminé.

Le système déductif de cette théorie est composé de trois jugements :

1. **Jugement de typage** $a : A$, qui affirme que a est de type A.
2. **Jugement d'égalité** $a \equiv b : A$, qui affirme que a et b sont égaux par définition dans le type A.
3. **Jugement de contexte** $\Gamma\,\mathrm{ctx}$, exprimant le fait que Γ est un contexte bien formé.

A noter que le symbole "$\equiv$" est différent de "$=$". En effet, si $a, b : A$, alors on a le type $a =_A b$ qui correspond à une égalité que l'on peut prouver, c'est l'égalité propositionnelle. Pour l'égalité "par définition" du système déductif, la prouver ou la supposer n'a pas réellement de sens étant donné qu'elle est vraie par définition (ou par construction).

Cette distinction entre proposition et jugement est fondamentale. Un jugement est l'affirmation qu'une proposition est vraie dans le système déductif de la théorie, alors qu'une proposition est un type, pouvant être non vide et se situe donc dans la théorie elle-même.

I.2 Règles structurelles

Pour prouver une proposition dans la théorie des types, on utilise des règles d'inférences, ou plus simplement règles, de la forme

$$\frac{\mathcal{I}_1 \quad \dots \quad \mathcal{I}_k}{\mathcal{I}}\,\textsc{Nom}$$

où $\mathcal{I}$ est la conclusion de la règle tandis que les $\mathcal{I}_1, \dots \mathcal{I}_k$ sont les prémisses ou hypothèses de la règle.

Avec de telles règles, on peut prouver une proposition en construisant un arbre de preuve dont la racine est le jugement à prouver et les liens entre les noeuds respectent les règles d'inférences de la théorie.

I.2.1 Contexte

Les règles concernant le jugement de contexte sont au nombre de deux et permettent de vérifier qu'un contexte Γ est bien formé, c'est-à-dire que toutes les variables et jugements apparaissant dans Γ sont dans un ordre cohérent. Pour ce faire on a ces règles :

$$\frac{}{\cdot\ \text{ctx}}\,(\emptyset\,\text{ctx}) \qquad \frac{a_1 : A_1, \dots, a_{n-1} : A_{n-1} \vdash A_n\ \text{type}}{(a_1 : A_1, \dots, a_n : A_n)\,\text{ctx}}\,(\text{ctx})$$

Le jugement "A_n type" signifie que A_n est un type et sera détaillé davantage dans la partie suivante. La première règle dit qu'un contexte vide est toujours bien formé, la seconde règle dit qu'un contexte est bien formé dès lors que la dernière variable est bien typé sous l'hypothèse que le contexte auquel on retire la dernière variable est lui-même bien formé.

I.2.2 Égalité

L'égalité par définition "$\equiv$" se comporte comme l'égalité classique, pour ce faire elle vérifie les mêmes propriétés élémentaires, à savoir la réflexivité, la symétrie et la transitivité que l'on exprime avec les règles suivantes.

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash a \equiv a : A}\,(\text{refl}) \qquad \frac{\Gamma \vdash b \equiv a : A}{\Gamma \vdash a \equiv b : A}\,(\text{sym}) \qquad \frac{\Gamma \vdash a \equiv b : A \qquad \Gamma \vdash b \equiv c : A}{\Gamma \vdash a \equiv c : A}\,(\text{trans})$$

On ajoute aussi deux règles supplémentaires qui permettent d'échanger le rôle de deux élément ou type sous l'hypothsèse qu'ils sont égaux.

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash a : B}\,(\equiv \text{type}) \qquad \frac{\Gamma \vdash a \equiv b : A \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash a \equiv b : B}\,(\equiv \text{eq})$$

Enfin, on ajoute une dernière règle permettant d'utiliser des jugements présent dans le contexte, qui s'exprime sous la forme suivante.

$$\frac{(a_1 : A_1, \ldots, a_n : A_n)\,\mathrm{ctx}}{a_1 : A_1, \ldots, a_n : A_n \vdash a_i : A_i}\,(\mathrm{var})$$

I.3 Construction de types

Pour construire un nouveau type à partir d'autres, on donne généralement 3 règles :

- **Formation du type**, qui permet de construire un type à partir d'autres types ou famille de types,
- **Introduction**, qui explique comment sont construits les éléments de ce type,
- **Élimination**, décrivant comment utiliser ces éléments.

Partant de ces 3 règles, on peut construire un type et éventuellement ajouter des règles supplémentaires concernant leur comportement par rapport à l'égalité par définition.

I.3.1 Univers

Plus tôt, on a utilisé un jugement de la forme "$\Gamma \vdash A$ type" pour signifier que A est un type. Cependant ce jugement n'est pas un jugement de la théorie, on se ramène alors à un autre jugement déjà présent : le jugement de typage. Pour ce faire, on doit introduire un type dont les éléments sont des types, un type univers $\mathcal{U}$. Pour la même raison, on voudrait pouvoir dire que $\mathcal{U}$ est lui-même un élément d'un type plus grand et ainsi de suite. Pour résoudre ce problème, on postule qu'il existe une hiérarchie de types

$$\mathcal{U}_0,\ \mathcal{U}_1,\ \mathcal{U}_2,\ \ldots,\ \mathcal{U}_i,\ \ldots$$

telle que $\mathcal{U}_i : \mathcal{U}_{i+1}$.

Pour ces types, on a seulement une règle d'introduction et une autre décrivant la hiérarchie entre ces types :

$$\frac{\Gamma\,\mathrm{ctx}}{\Gamma \vdash \mathcal{U}_i : \mathcal{U}_{i+1}}\,(\mathcal{U}\ \mathrm{i}) \qquad\qquad \frac{\Gamma \vdash A : \mathcal{U}_i}{\Gamma \vdash A : \mathcal{U}_{i+1}}\,(\mathcal{U}\ \mathrm{cumul})$$

Par conséquent, la règle sur les contextes devient la suivante :

$$\frac{(a_1 : A_1, \ldots, a_{n-1} : A_{n-1}) \vdash A_n : \mathcal{U}_k}{(a_1 : A_1, \ldots, a_n : A_n)\,\mathrm{ctx}}\,(\mathrm{ctx})$$

I.3.2 Fonctions

Étant donné deux types $A, B : \mathcal{U}_i$, on peut former le type des fonctions de A dans B noté $A \to B$ grâce à la règle de formation suivante :

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma \vdash B : \mathcal{U}_i}{\Gamma \vdash A \to B : \mathcal{U}_i}\,(\to \mathrm{f})$$

Une fonction est donc un élément f de type $A \to B$. Si on se donne $a : A$, on peut évaluer f en a ce qui donne un élément de type B que l'on note $f\ a$ ou bien $f(a)$ qui est appelé valeur

de f en a, c'est la règle d'élimination des types de fonctions :

$$\frac{\Gamma \vdash f : A \to B \qquad \Gamma \vdash a : A}{\Gamma \vdash f\ a : B} (\to \text{e})$$

Pour former une fonction f de A dans B *ie.* un élément de type $A \to B$, la manière canonique de faire est d'utiliser les λ-abstractions et la règle d'introduction suivante :

$$\frac{\Gamma, x : A \vdash \Phi : B}{\Gamma \vdash \lambda(x : A).\Phi : A \to B} (\to \text{i})$$

Dans cette règle, Φ correspond à une formule dans laquelle la variable x intervient éventuellement.

Enfin, lorsque l'on considère une fonction $f : A \to B$ définie par $f :\equiv \lambda(x : A).\Phi$ où Φ est une formule et un élément $a : A$, il est naturel que la valeur de f en a soit égale à la formule Φ où l'on remplace les occurences de x par des a, ce que l'on note $\Phi[a/x]$. Ce comportement est une règle d'égalité que l'on peut énoncer sous cette forme :

$$\frac{\Gamma, x : A \vdash \Phi : B \qquad \Gamma \vdash a : A}{\Gamma \vdash (\lambda(x : A).\Phi)\ a \equiv \Phi[a/x] : B} (\beta)$$

On ajoute aussi une dernière règle correspondant au principe d'unicité des fonctions, affirmant qu'une fonction $f : A \to B$ est égale à la fonction $\lambda(a : A).\ f\ a$. Sous forme d'une règle, on l'écrit :

$$\frac{\Gamma \vdash f : A \to B}{\Gamma \vdash f \equiv \lambda(a : A).\ f\ a : A \to B} (\to \text{u})$$

Exemple. Considérons un type $A : \mathcal{U}_i$ et la fonction "identité" de A définie par $I :\equiv \lambda(x : A).x$. Pour prouver que le type de I est bien $A \to A$, on peut utiliser les règles énoncées précédemment :

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{}{\cdot\ \text{ctx}}}{\vdash \mathcal{U}_i : \mathcal{U}_{i+1}}}{(A : \mathcal{U}_i)\ \text{ctx}}}{A : \mathcal{U}_i \vdash A : \mathcal{U}_i}}{(A : \mathcal{U}_i, x : A)\ \text{ctx}}}{A : \mathcal{U}_i, x : A \vdash x : A}}{A : \mathcal{U}_i \vdash \lambda(x : A).\ x : A \to A}$$

On a vu jusqu'ici comment construire des fonctions à une variable, on peut alors généraliser pour obtenir des fonctions à plusieurs variables. Pour ce faire, considérons trois types $A, B, C : \mathcal{U}_i$. Naïvement, on voudrait construire une fonction de type $A \times B \to C$, mais le type $A \times B$ n'est pas encore défini. Pour contourner ce problème, on peut voir une fonction f de $A \times B \to C$ comme

$$f(x,y) = (\lambda(a:A).f(a,y))x = ((\lambda(a:A).(\lambda(b:B).f(a,b)))\ x)\ y \quad (1)$$
$$= (\lambda(a:A).\lambda(b:B).f(a,b))\ x\ y. \quad (2)$$

On a ainsi obtenu une fonction de type $A \to B \to C$ qui prend les mêmes valeurs que f grâce à ce processus de *curryfication*.

Exemple. Considérons deux types $A, B : \mathcal{U}_i$ et la fonction K définie par

$$K :\equiv \lambda(x:A).(\lambda(y:B).x)$$

Grâce à un arbre similaire au précédent, on trouve que le type de K est $A \to B \to A$ (on omet les parenthèses en prenant comme convention que $A \to B \to C$ est égal à $A \to (B \to C)$).

I.3.3 Fonctions dépendantes (Π-type)

Une manière de généraliser le type des fonctions est de définir un type de fonctions dépendantes (ou $\prod$-type), qui est un type de fonction dont le codomaine dépend du point d'application de la fonction.

Pour former ce nouveau type, on considère $A : \mathcal{U}_i$ un type et $B : A \to \mathcal{U}_i$ une famille de type indexée sur A. On peut alors former le type $\prod_{a:A} B(a)$ grâce à la règle de formation suivante :

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma, a : A \vdash B(a) : \mathcal{U}_i}{\Gamma \vdash \prod_{a:A} B(a) : \mathcal{U}_i}\ (\Pi\ \text{f})$$

On construit des éléments de type $\prod_{a:A} B(a)$ de manière analogue aux fonctions en utilisant les λ-abstractions. En effet, $\lambda(a:A).\Phi$ est de type $\prod_{a:A} B(a)$ si lorsque $a : A$, alors $\Phi : B(a)$. Cette règle énoncée plus formellement est la règle d'introduction de ce nouveau type :

$$\frac{\Gamma, a : A \vdash \Phi : B(a)}{\Gamma \vdash \lambda(a:A).\Phi : \prod_{a:A} B(a)}\ (\Pi\ \text{i})$$

Ensuite, la règle d'élimination est similaire à celle des types de fonctions. Si on considère $f : \prod_{a:A} B(a)$ et $a : A$, alors $f\ a$ est de type $B(a)$, ce que l'on peut traduire avec la règle suivante :

$$\frac{\Gamma \vdash f : \prod_{a:A} B(a) \qquad \Gamma \vdash a : A}{\Gamma \vdash f\ a : B(a)}\ (\Pi\ \text{e})$$

Également, la règle β et le principe d'unicité des fonctions se généralisent au $\prod$-type :

$$\frac{\Gamma, x : A \vdash \Phi : B(x) \qquad \Gamma \vdash a : A}{\Gamma \vdash (\lambda(x:A).\Phi)\ a \equiv \Phi[a/x]}\ (\beta)$$

$$\frac{\Gamma \vdash f : \prod_{a:A} B(a)}{\Gamma \vdash f \equiv \lambda(a : A).\ f\ a : \prod_{a:A} B(a)} (\to \text{ u})$$

L'utilité des types de fonctions dépendantes est notamment de pouvoir définir des fonctions de façon polymorphes, comme le montre les exemples suivants.

Exemple. La fonction identité du paragraphe précédent était définie sur un type $A : \mathcal{U}_i$ quelconque. On peut alors généraliser sa définition de la manière suivante :

$$I \equiv \lambda(A : \mathcal{U}_i).\lambda(a : A).\ a : \prod_{A:\mathcal{U}_i} A \to A.$$

Ainsi définie, la fonction I ne dépend pas d'un type préexistant.

On peut également redéfinir la fonction constante K d'une manière analogue :

$$K \equiv \lambda(A : \mathcal{U}_i).\lambda(B : \mathcal{U}_i).\lambda(x : A).\lambda(y : B).\ x : \prod_{A:\mathcal{U}_i} \prod_{B:\mathcal{U}_i} A \to B \to A.$$

Pour ces types et définitions polymorphes, on peut également avoir la notation I_A à la place de $I\ A$ et $K_{A,B}$ pour $K\ A\ B$ pour plus de clarté.

I.3.4 Produits

Etant donné deux types $A, B : \mathcal{U}_i$, comme en théorie des ensembles on veut construire le type $A \times B$ des paires d'éléments de A et de B. Pour construire ce type on a la règle de formation suivante :

$$\frac{\Gamma \vdash A : \mathcal{U}_i \quad \Gamma \vdash B : \mathcal{U}_i}{\Gamma \vdash A \times B : \mathcal{U}_i} (\times \text{ f})$$

Les éléments de ce type sont des paires d'éléments et sont donc de la forme $(a, b) : A \times B$ où $a : A$ et $b : B$, on a donc la règle d'introduction pour décrire les paires :

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash (a, b) : A \times B} (\times \text{ i})$$

Ensuite pour les règles d'éliminations et de calculs, on introduit deux fonctions $\pi_1 : A \times B \to A$ et $\pi_2 : A \times B \to B$ qui projettent un élément $(a, b) : A \times B$ sur A et sur B.

$$\frac{\Gamma \vdash e : A \times B}{\Gamma \vdash \pi_1\ e : A} (\times_1 \text{ e}) \qquad \frac{\Gamma \vdash e : A \times B}{\Gamma \vdash \pi_2\ e : B} (\times_2 \text{ e})$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash \pi_1\ (a, b) \equiv a : A} (\times_1 \equiv) \quad \frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash \pi_2\ (a, b) \equiv b : B} (\times_2 \equiv)$$

Enfin, pour les types produits on a également une règle d'unicité reliant un élément à ses

projections :

$$\frac{\Gamma \vdash e : A \times B}{\Gamma \vdash e \equiv (\pi_1\ e, \pi_2\ e) : A \times B}\,(\times\ \mathrm{u})$$

On peut également voir les types produits comme un cas particulier des types de fonctions dépendantes dont le domaine est le type fini **2** à deux éléments, qui sera décrit dans I.3.8.

I.3.5 Paires dépendantes (Σ-types)

Si on considère un type $A : \mathcal{U}_i$ et une famille de type $B : A \to \mathcal{U}_i$ indexée sur A, on veut pouvoir construire l'union disjointe des $B(a)$ pour $a : A$. Pour ce faire, on introduit les types de paires dépendantes (ou Σ-type) qui se forment grâce à la règle :

$$\frac{\Gamma \vdash A : \mathcal{U}_i \quad \Gamma, a : A \vdash B(a) : \mathcal{U}_i}{\Gamma \vdash \Sigma_{a:A} B(a) : \mathcal{U}_i}\,(\Sigma\ \mathrm{f})$$

Un élément de ce type est une paire (a, b) où $a : A$ et $b : B(a)$ (d'où le nom de paire dépendante), ce que la règle suivante décrit :

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B(a)}{\Gamma \vdash (a, b) : \Sigma_{a:A} B(a)}\,(\Sigma\ \mathrm{i})$$

Ensuite, comme les paires dépendantes sont des paires, on a de la même manière que pour les types produits les projections $\pi_1 : \Sigma_{a:A} B(a) \to A$ et $\pi_2 : \prod_{e:\Sigma_{a:A} B(a)} B(\pi_1\ e)$ sur chacune des coordonnées d'un élément de $\Sigma_{a:A} B(a)$. Ces projections se comportent de manière analogue à celle des produits, ce qu'on remarque grâce aux règles d'éliminations et de calculs des Σ-types :

$$\frac{\Gamma \vdash e : \Sigma_{a:A} B(a)}{\Gamma \vdash \pi_1\ e : A}\,(\Sigma_1\ \mathrm{e}) \qquad \frac{\Gamma \vdash e : \Sigma_{a:A} B(a)}{\Gamma \vdash \pi_2\ e : B(\pi_1\ e)}\,(\Sigma_2\ \mathrm{e})$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B(a)}{\Gamma \vdash \pi_1\ (a, b) \equiv a : A}\,(\Sigma_1 \equiv) \qquad \frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B(a)}{\Gamma \vdash \pi_2\ (a, b) \equiv b : B(a)}\,(\Sigma_2 \equiv)$$

Enfin, on a également un équivalent de la règle d'unicité des types produits pour les Σ-types :

$$\frac{\Gamma \vdash e : \Sigma_{a:A} B(a)}{\Gamma \vdash e \equiv (\pi_1\ e, \pi_2\ e) : \Sigma_{a:A} B(a)}\,(\Sigma\ \mathrm{u})$$

Exemple (Cas où B est constant). Si on considère deux types $A, B : \mathcal{U}_i$, on peut poser $C :\equiv \Sigma_{a:A} B$ qui est en tout point similaire au produit $A \times B$. En effet, si $(a, b) : C$ alors $a : A$ et $b : B(a) \equiv B$ et donc $(a, b) : A \times B$. De plus les règles étant très similaires, lorsque la famille B est constante on retrouve les règles des types produits.

I.3.6 Type vide

On introduit le type vide **0**, qui ne contient aucun élément et donc n'a pas de règle d'introduction mais seulement une régle de formation et d'élimination :

$$\frac{\Gamma \text{ ctx}}{\Gamma \vdash \mathbf{0} : \mathcal{U}_i}\,(\mathbf{0}\text{ f}) \qquad\qquad \frac{\Gamma \vdash C : \mathcal{U}_i \qquad \Gamma \vdash a : \mathbf{0}}{\Gamma \vdash \mathrm{rec}_{\mathbf{0}}(C, a) : C}\,(\mathbf{0}\text{ e})$$

Ainsi, on a construit une fonction $\mathrm{rec}_{\mathbf{0}}$ de type $\prod_{C:\mathcal{U}_i} \mathbf{0} \to C$ qui à un type $C : \mathcal{U}_i$ associe une fonction qui ne prend aucune valeur. Par conséquent, on ne devrait jamais calculer cette fonction, c'est la raison pour laquelle il n'y a pas de règle de calcul pour ce type. La règle d'élimination agit comme le principe *ex falso quodlibet*, en effet si on construit un élément du type vide (*ie.* une preuve de $\bot$), alors on peut construire des éléments de n'importe quel type.

I.3.7 Type unité

Ensuite, dans l'objectif de construire tous les types finis, on commence par construire le type unité $\mathbf{1}$, contenant seulement un élément. Pour ce faire on a la règle de formation analogue à celle du type $\mathbf{0}$ avec une règle d'introduction simple car le type $\mathbf{1}$ ne possède qu'un seul élément :

$$\frac{\Gamma \text{ ctx}}{\Gamma \vdash \mathbf{1} : \mathcal{U}_i}\,(\mathbf{1}\text{ f}) \qquad\qquad \frac{\Gamma \text{ ctx}}{\Gamma \vdash \star : \mathbf{1}}\,(\mathbf{1}\text{ i})$$

La règle d'élimination de $\mathbf{1}$ permet de construire des fonctions de $\mathbf{1}$ vers un autre type C, qui sont donc des fonctions constantes et c'est ce qu'affirme la règle de calcul du type unité.

$$\frac{\Gamma, x : \mathbf{1} \vdash C(x) : \mathcal{U}_i \qquad \Gamma \vdash c : C(x) \qquad \Gamma \vdash x : \mathbf{1}}{\Gamma \vdash \mathrm{rec}_{\mathbf{1}}(C, c, x) : C(x)}\,(\mathbf{1}\text{ e})$$
$$\frac{\Gamma, x : \mathbf{1} \vdash C(x) : \mathcal{U}_i \qquad \Gamma \vdash c : C(\star)}{\Gamma \vdash \mathrm{rec}_{\mathbf{1}}(C, c, \star) \equiv c : C(\star)}\,(\mathbf{1} \equiv)$$

où la fonction $\mathrm{rec}_{\mathbf{1}}$ est de type $\prod_{C:\mathbf{1}\to\mathcal{U}_i} C(\star) \to \prod_{x:\mathbf{1}} C(x)$ De plus pour ce type, on a une règle d'unicité car le seul élément de $\mathbf{1}$ est $\star$:

$$\frac{\Gamma \vdash p : \mathbf{1}}{\Gamma \vdash p \equiv \star : \mathbf{1}}\,(\mathbf{1}\text{ u})$$

I.3.8 Coproduit

Le dernier élément manquant pour construire tout les types finis sont les coproduits, qui permettent de construire un équivalent de l'union disjointe de la théorie des ensembles à partir de deux types comme l'exprime la règle de formation suivante :

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma \vdash B : \mathcal{U}_i}{\Gamma \vdash A + B : \mathcal{U}_i}\,(+\text{ f})$$

Étant donné deux types $A, B : \mathcal{U}_i$, un élément e du type $A + B$ à deux manières d'être construit :

— soit e est issu d'un élément $a : A$ et est donc de la forme $\mathrm{inl}(a)$,

— soit e est issu d'un élément $b : B$ et est donc de la forme $\mathrm{inr}(b)$.

On a donc deux règles d'introduction pour les coproduits :

$$\frac{\Gamma \vdash A : \mathcal{U}_i \quad \Gamma \vdash B : \mathcal{U}_i \quad \Gamma \vdash a : A}{\Gamma \vdash \mathrm{inl}(a) : A + B}\,(+\,\mathrm{inl}) \qquad \frac{\Gamma \vdash A : \mathcal{U}_i \quad \Gamma \vdash B : \mathcal{U}_i \quad \Gamma \vdash b : B}{\Gamma \vdash \mathrm{inr}(b) : A + B}\,(+\,\mathrm{inr})$$

Pour prouver des propositions sur des coproduits, une manière naturelle de faire est de raisonner par cas : ou bien on a un élément de A ou bien il s'agit d'un élément de B. Pour décrire cette méthode de raisonnement, on a la règle d'élimination des coproduits pour deux types $A, B : \mathcal{U}_i$:

$$\frac{\begin{array}{c}\Gamma, a : A \vdash g_0\ a : C(\mathrm{inl}\ a) \\ \Gamma, x : A + B \vdash C(x) : \mathcal{U}_i \qquad \Gamma, b : B \vdash g_1\ b : C(\mathrm{inr}\ b) \qquad \Gamma \vdash e : A + B\end{array}}{\Gamma \vdash \mathrm{rec}_{A+B}(C, g_0, g_1, e) : C(e)}\,(+\,\mathrm{e})$$

Ainsi, la fonction rec_{A+B} est de type

$$\prod_{C : A+B \to \mathcal{U}_i} \left(\prod_{a:A} C(\mathrm{inl}\ a) \right) \to \left(\prod_{b:B} C(\mathrm{inr}\ b) \right) \to \left(\prod_{e:A+B} C(e) \right)$$

et permet de faire la disjonction de cas. On a de plus deux règles décrivant le comportement de la fonction rec_{A+B} en fonction de son dernier argument :

$$\frac{\begin{array}{c}\Gamma, a : A \vdash g_0\ a : C(\mathrm{inl}\ a) \\ \Gamma, x : A + B \vdash C(x) : \mathcal{U}_i \quad \Gamma, b : B \vdash g_1\ b : C(\mathrm{inr}\ b) \quad \Gamma \vdash x : A\end{array}}{\Gamma \vdash \mathrm{rec}_{A+B}(C, g_0, g_1, \mathrm{inl}(x)) \equiv g_0\ x : C(\mathrm{inl}\ x)}\,(+ \equiv_1)$$

$$\frac{\begin{array}{c}\Gamma, a : A \vdash g_0\ a : C(\mathrm{inl}\ a) \\ \Gamma, x : A + B \vdash C(x) : \mathcal{U}_i \quad \Gamma, b : B \vdash g_1\ b : C(\mathrm{inr}\ b) \quad \Gamma \vdash y : B\end{array}}{\Gamma \vdash \mathrm{rec}_{A+B}(C, g_0, g_1, \mathrm{inr}(y)) \equiv g_1\ y : C(\mathrm{inr}\ y)}\,(+ \equiv_2)$$

Exemple (Le type **2**). Avec le coproduit et le type **1** on peut désormais construire le type **2** à deux éléments. En effet, on définit **2** par $\mathbf{2} :\equiv \mathbf{1} + \mathbf{1}$, ce qui donne la règle de formation

$$\frac{\dfrac{\Gamma\ \mathrm{ctx}}{\Gamma \vdash \mathbf{1} : \mathcal{U}_i} \quad \dfrac{\Gamma\ \mathrm{ctx}}{\Gamma \vdash \mathbf{1} : \mathcal{U}_i}}{\Gamma \vdash \mathbf{1} + \mathbf{1} : \mathcal{U}_i}$$

que l'on peut résumer à :

$$\frac{\Gamma\ \mathrm{ctx}}{\Gamma \vdash \mathbf{2} : \mathcal{U}_i}\,(\mathbf{2}\ \mathrm{f})$$

Ensuite, comme pour le coproduit on a deux règles d'introduction :

$$\frac{\dfrac{\Gamma\ \mathrm{ctx}}{\Gamma \vdash \mathbf{1} : \mathcal{U}_i} \quad \dfrac{\Gamma\ \mathrm{ctx}}{\Gamma \vdash \mathbf{1} : \mathcal{U}_i} \quad \dfrac{\Gamma\ \mathrm{ctx}}{\Gamma \vdash \star : \mathbf{1}}}{\Gamma \vdash \mathrm{inl}(\star) : \mathbf{2}} \qquad \frac{\dfrac{\Gamma\ \mathrm{ctx}}{\Gamma \vdash \mathbf{1} : \mathcal{U}_i} \quad \dfrac{\Gamma\ \mathrm{ctx}}{\Gamma \vdash \mathbf{1} : \mathcal{U}_i} \quad \dfrac{\Gamma\ \mathrm{ctx}}{\Gamma \vdash \star : \mathbf{1}}}{\Gamma \vdash \mathrm{inr}(\star) : \mathbf{2}}$$

En posant $0_{\mathbf{2}} \equiv \bot \equiv \mathrm{inl}(\star)$ et $1_{\mathbf{2}} \equiv \top \equiv \mathrm{inr}(\star)$, on peut réécrire les deux règles d'introduction plus simplement :

$$\frac{\Gamma \,\mathrm{ctx}}{\bot : \mathbf{2}}\,(\bot\ \mathrm{i}) \qquad\qquad \frac{\Gamma \,\mathrm{ctx}}{\top : \mathbf{2}}\,(\top\ \mathrm{i})$$

On obtient alors la fonction $\mathrm{rec}_{\mathbf{2}}$ de type

$$\prod_{C:\mathbf{2}\to\mathcal{U}_i} \left(\prod_{x:\mathbf{1}} C(\mathrm{inl}\ x)\right) \to \left(\prod_{y:\mathbf{1}} C(\mathrm{inr}\ y)\right) \to \prod_{b:\mathbf{2}} C(b),$$

que l'on peut également voir comme le type $\prod_{C:\mathbf{2}\to\mathcal{U}} C(\bot) \to C(\top) \to \prod_{b:\mathbf{2}} C(b)$ (car les fonctions sur $\mathbf{1}$ sont constantes et égales à la valeurs qu'elles prennent sur $\star$). Cette fonction vérifie les règles d'élimination et de calcul suivantes :

$$\frac{\Gamma, b : \mathbf{2} \vdash C(b) : \mathcal{U}_i \quad \Gamma, p : \mathbf{1} \vdash g_0\ p : C(\mathrm{inl}\ p) \quad \Gamma, p : \mathbf{1} \vdash g_1\ p : C(\mathrm{inr}\ p) \quad \Gamma \vdash e : \mathbf{2}}{\Gamma \vdash \mathrm{rec}_{\mathbf{2}}(C, g_0, g_1, e) : C(e)}$$

$$\frac{\Gamma, b : \mathbf{2} \vdash C(b) : \mathcal{U}_i \quad \Gamma, p : \mathbf{1} \vdash g_0\ p : C(\mathrm{inl}\ p) \quad \Gamma, p : \mathbf{1} \vdash g_1\ p : C(\mathrm{inr}\ p)}{\Gamma \vdash \mathrm{rec}_{\mathbf{2}}(C, g_0, g_1, \bot) \equiv g_0\ \bot : C(\bot)}$$

$$\frac{\Gamma, b : \mathbf{2} \vdash C(b) : \mathcal{U}_i \quad \Gamma, p : \mathbf{1} \vdash g_0\ p : C(\mathrm{inl}\ p) \quad \Gamma, p : \mathbf{1} \vdash g_1\ p : C(\mathrm{inr}\ p)}{\Gamma \vdash \mathrm{rec}_{\mathbf{2}}(C, g_0, g_1, \top) \equiv g_0\ \top : C(\top)}$$

On peut les réécrire en modifiant les types de $\mathrm{rec}_{\mathbf{2}}$, puisque les fonctions dont le domaine est $\mathbf{1}$ sont des constantes :

$$\frac{\Gamma, b : \mathbf{2} \vdash C(b) : \mathcal{U}_i \quad \Gamma \vdash g_0 : C(\bot) \quad \Gamma \vdash g_1 : C(\top) \quad \Gamma \vdash e : \mathbf{2}}{\Gamma \vdash \mathrm{rec}_{\mathbf{2}}(C, g_0, g_1, e) : C(e)}\,(\mathbf{2}\ \mathrm{e})$$

$$\frac{\Gamma \vdash g_0 : C(\bot) \qquad \Gamma, b : \mathbf{2} \vdash C(b) : \mathcal{U}_i \quad \Gamma \vdash g_1 : C(\top)}{\Gamma \vdash \mathrm{rec}_{\mathbf{2}}(C, g_0, g_1, \bot) \equiv g_0 : C(\bot)}\,(\mathbf{2} \equiv_1) \qquad \frac{\Gamma \vdash g_0 : C(\bot) \qquad \Gamma, b : \mathbf{2} \vdash C(b) : \mathcal{U}_i \quad \Gamma \vdash g_1 : C(\top)}{\Gamma \vdash \mathrm{rec}_{\mathbf{2}}(C, g_0, g_1, \top) \equiv g_1 : C(\top)}\,(\mathbf{2} \equiv_2)$$

On remarque alors que la fonction $\mathrm{rec}_{\mathbf{2}}$ agit comme le "si .. alors .. sinon .." des booléens, ce qui renforce la comparaison entre le type $\mathbf{2}$ et les booléens.

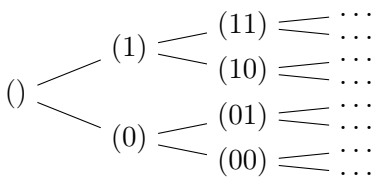
Plus généralement on peut construire tout les types finis $\mathbf{T}_n$ comme étant n fois le coproduit du type $\mathbf{1}$, ce qui construit des types à n éléments.

I.3.9 Types inductifs : W-types

Pour construire le type des entiers $\mathbf{N}$, les constructeurs précédents ne suffisent pas car ils ne permettent pas de décrire le caractère inductif des entiers. Habituellement, on définit le type $\mathbf{N}$ comme étant un élément $0 : \mathbf{N}$ et une fonction successeur $s : \mathbf{N} \to \mathbf{N}$ et un entier est de la forme 0 ou bien $s \circ s \circ \ldots \circ s\ 0$. Une autre manière de le voir est de décrire $\mathbf{N}$ comme un arbre dont 0 est une feuille et chacun des noeuds est d'arité un (*ie.* possède au plus un fils).

$$0 \text{ —— } 1 \text{ —— } 2 \text{ —— } \cdots$$

Un autre exemple de type défini de manière inductive est le type $L(A)$ des listes sur un type $A : \mathcal{U}_i$. La seule feuille est la liste vide notée () et la fonctions jouant le rôle de constructeur inductif est la fonction $c : A \to L(A) \to L(A)$ qui a un élément $a : A$ et une liste $l : L(A)$ renvoie la liste dont le premier élément est a et le reste est l. Ainsi, toutes les listes sont de la forme $() : L(A)$ ou bien $c\ a_1\ (c\ a_2\ (\ldots (c\ a_n\ ()))) : L(A)$.



Pour généraliser les types inductifs, on introduit alors les W-types (*well-founded trees*) de Martin-Löf. Pour ce faire, on utilise un type $A : \mathcal{U}_i$ qui correspond aux *étiquettes* et un type $B : A \to \mathcal{U}$ qui permet de coder l'arité des noeuds : un noeud dont l'étiquette est $a : A$ aura une arité de $B(a)$. Le type résultant de ces deux types est $W_{a:A}B(a)$ que l'on construit via la règle de formation suivante :

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma, a : A \vdash B(a) : \mathcal{U}_i}{\Gamma \vdash W_{a:A}B(a) : \mathcal{U}_i}\ (\text{W f})$$

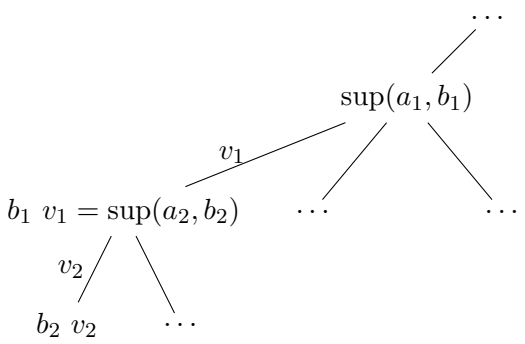
Les éléments d'un type $W_{a:A}B(a)$ sont donc des noeuds d'un arbre auxquels on associe la fonction donnant les noeuds précédents dans l'arbre. On construit ces éléments à l'aide de la règle d'introduction pour les W-types :

$$\frac{\Gamma \vdash a : A \qquad \Gamma, a : A \vdash b : B(a) \to W_{x:A}B(x)}{\Gamma \vdash \sup(a, b) : W_{x:A}B(x)}\ (\text{W i})$$

La fonction sup utilisée dans cette règle est donc de type

$$\sup : \prod_{a:A} (B(a) \to W_{x:A}B(x)) \to W_{a:A}B(a).$$

Chaque noeud $c :\equiv \sup(a, b) : W_{x:A}B(x)$ est donc relié aux autres par la fonction $b : B(a) \to W_{x:A}B(x)$, en effet si on considère un élément $v : B(a)$ alors $b\ v$ est un prédécesseur du noeud c dans l'arbre $W_{x:A}B(x)$.



Reprenons l'exemple du type $\mathbf{N}$ des entiers. On peut le décrire en terme de W-type en prenant

$\mathbf{N} :\equiv W_{x:\mathbf{2}}B(x)$ où $B(0_\mathbf{2}) :\equiv \mathbf{0}$ et $B(1_\mathbf{2}) :\equiv \mathbf{1}$. En effet, il y a deux manières de construire un entier : ou bien c'est 0 ou bien il est de la forme $s\ k$ avec $k : \mathbf{N}$. Pour 0, son étiquette dans l'arbre est donc $0_\mathbf{2}$ tandis que les éléments de la forme $s\ k$ ont l'étiquette $1_\mathbf{2}$. On peut donc écrire 0 avec la fonction sup :

$$0 :\equiv \sup(0_\mathbf{2}, \mathrm{rec}_{\mathbf{0},\mathbf{N}}).$$

De même la fonction successeur $s : \mathbf{N} \to \mathbf{N}$ peut être définie de la manière suivante :

$$s :\equiv \lambda n.\ \sup(1_\mathbf{2}, \lambda(x : \mathbf{1}).\ n) : \mathbf{N} \to \mathbf{N}.$$

Et donc on a $1 :\equiv s\ 0 \equiv \sup(1_\mathbf{2}, \lambda x.\ 0)$ etc.

La règle d'élimination décrit le principe d'induction sur les W-types. Pour prouver une proposition sur un type inductif, de manière analogue au principe de récurrence des entiers, on montre que la proposition est héréditaire, dans le cas présent cela revient à montrer que si la proposition est vraie sur tout les sous-arbres, alors elle est vraie sur le noeud reliant ces sous-arbres, c'est ce que dit la dernière prémisse de la règle d'élimination, qui s'énonce de la manière suivante :

$$\frac{\Gamma\ \vdash t : W_{x:A}B(x) \qquad \begin{array}{c}\Gamma, a : A, b : B(a) \to W_{x:A}B(x), \\ c : \prod_{v:B(a)} E(b\ v)\ \vdash e(a,b,c) : E(\sup(a,b))\end{array}}{\Gamma\ \vdash \mathrm{rec}_{W_{x:A}B(x)}(E,e,t) : E(t)}\ (\mathrm{W\ e})$$

La fonction e présente dans la règle est de type

$$e : \prod_{a:A} \prod_{b:(B(a)\to W_{x:A}B(x))} \left(\prod_{v:B(a)} E(b\ v) \right) \to E(\sup(a,b)).$$

et pour des éléments $a : A$, $b : B(a) \to W_{x:A}B(x)$ et une preuve c de $\prod_{v:B(a)} E(b\ v)$ renvoie une preuve de $E(\sup(a,b))$. La fonction $\mathrm{rec}_{W_{x:A}B(x)}$ présente dans la conclusion de la règle précédente est une fonction qui à un prédicat (ou famille de type) E sur le type $W_{x:A}B(x)$, une fonction e comme décrit juste avant et un élément $t : W_{x:A}B(x)$ retourne une preuve (ou un élément) de $E(t)$. Ainsi, la fonction e et la fonction $\mathrm{rec}_{W_{x:A}B(x)}$ renvoie des preuves de même nature, on peut donc imaginer une relation liant les deux : c'est la règle d'égalité (ou règle de calcul de $\mathrm{rec}_{W_{x:A}B(x)}$).

$$\frac{\begin{array}{c}\Gamma\ \vdash a : A \\ \Gamma\ \vdash b : B(a) \to W_{x:A}B(x) \\ \Gamma, x : A, y : B(x) \to W_{x:A}B(x), z : \prod_{v:B(x)} E(y\ v)\ \vdash e(x,y,z) : E(\sup(x,y))\end{array}}{\Gamma \vdash \mathrm{rec}_{W_{x:A}B(x)}(E,e,\sup(a,b)) \equiv e(a,b,\lambda v.\ \mathrm{rec}_{W_{x:A}B(x)}(E,e,b\ v)) : E(\sup(a,b))}\ (\mathrm{W} \equiv)$$

En effet, pour $a : A$ et $b : B(a) \to W_{x:A}B(x)$ la fonction $\lambda(v : B(a)).\ \mathrm{rec}_{W_{x:A}B(x)}(E,e,b\ v)$

retourne une preuve (un élément) de $\prod_{v:B(a)} E(b\ v)$, ce qui correspond au dernier paramètre de la fonction e et donc le membre de droite de l'égalité est bien une preuve (un élément) de $E(\sup(a,b))$ comme voulue. Les prémisses de cette règle sont de simple vérifications de typage de a, b et e.

I.3.10 Entiers naturels

Avec les W-types, nous pouvons désormais construire les entiers naturels. Pour ce faire, on note $\mathbf{N} \equiv W_{x:\mathbf{2}}B(x)$ où $B(0_\mathbf{2}) :\equiv \mathbf{0}$ et $B(1_\mathbf{2}) :\equiv \mathbf{1}$ (*ie.* $B \equiv \mathrm{rec}_\mathbf{2}(\mathcal{U}_i, \mathbf{0}, \mathbf{1}) : \mathbf{2} \to \mathcal{U}_i$). On a déjà vu dans le paragraphe précédent que tout élément de $\mathbf{N}$ s'exprime sous la forme 0 ou $s\ k$ avec $k : \mathbf{N}$ et de plus on avait les expressions de 0 et de $s : \mathbf{N} \to \mathbf{N}$ suivantes :

$$\begin{cases} 0 & :\equiv \sup(0_\mathbf{2}, \mathrm{rec}_{\mathbf{0},\mathbf{N}}) & : \mathbf{N} \\ s & :\equiv \lambda(n:\mathbf{N}).\ \sup(1_\mathbf{2}, \lambda(x:\mathbf{1}).\ n) & : \mathbf{N} \to \mathbf{N} \end{cases}$$

On peut également exprimer les règles de formation, d'introduction, d'élimination et d'égalité dans le cas de $\mathbf{N}$. Pour la règle de formation, on obtient :

$$\dfrac{\dfrac{\Gamma\,\mathrm{ctx}}{\Gamma\ \vdash \mathbf{2} : \mathcal{U}_i} \qquad \dfrac{\dfrac{\Gamma\,\mathrm{ctx}}{\Gamma, a:\mathbf{2}\ \vdash B : \mathbf{2} \to \mathcal{U}_i} \qquad \dfrac{\Gamma\,\mathrm{ctx}}{\Gamma, a:\mathbf{2}\ \vdash a : \mathbf{2}}}{\Gamma, a:\mathbf{2}\ \vdash B(a) : \mathcal{U}_i}}{\Gamma\ \vdash \mathbf{N} : \mathcal{U}_i}$$

On a donc la règle simplifiée :

$$\dfrac{\Gamma\,\mathrm{ctx}}{\Gamma\ \vdash \mathbf{N} : \mathcal{U}_i}\,\mathbf{N}\ \mathrm{i}$$

Ensuite, concernant la règle d'introduction on distingue deux cas puisqu'il y a deux manières de construire un élément $n : \mathbf{N}$, soit $n \equiv 0$, soit $n \equiv s\ k$ avec $k : \mathbf{N}$.

$$\dfrac{\dfrac{\dfrac{\Gamma\,\mathrm{ctx}}{\Gamma\ \vdash 0_\mathbf{2} : \mathbf{2}} \qquad \dfrac{}{\Gamma, 0_\mathbf{2} : \mathbf{2}\ \vdash \mathrm{rec}_{\mathbf{0},\mathbf{N}} : \mathbf{0} \to \mathbf{N}}}{\Gamma, 0_\mathbf{2} : \mathbf{2}\ \vdash \mathrm{rec}_{\mathbf{0},\mathbf{N}} : B(0_\mathbf{2}) \to \mathbf{N}}}{\Gamma\ \vdash \sup(0_\mathbf{2}, \mathrm{rec}_{\mathbf{0},\mathbf{N}}) : \mathbf{N}}}{\Gamma\ \vdash 0 : \mathbf{N}}$$

$$\dfrac{\dfrac{}{\Gamma\ \vdash s : \mathbf{N} \to \mathbf{N}} \qquad \Gamma\ \vdash k : \mathbf{N}}{\Gamma \vdash s\ k : \mathbf{N}}$$

Ce qui donne les deux règles d'introduction de $\mathbf{N}$:

$$\dfrac{\Gamma\,\mathrm{ctx}}{\Gamma\ \vdash 0 : \mathbf{N}}\,(0\ \mathrm{i}) \qquad\qquad \dfrac{\Gamma\ \vdash k : \mathbf{N}}{\Gamma\ \vdash s\ k : \mathbf{N}}\,(s\ \mathrm{i})$$

Pour la règle d'élimination sur le type $\mathbf{N}$, on s'attend à retrouver le principe de récurrence

usuel sur les entiers, à savoir que si une prédicat $E : \mathbf{N} \to \mathcal{U}_i$ est vrai sur 0 et que si $E(k)$ est vrai alors $E(s\ k)$ est vrai, alors $E(k)$ est vrai sur tout $k : \mathbf{N}$.

$$\dfrac{\Gamma\ \vdash k : \mathbf{N} \qquad \dfrac{\text{Lemme 1} \qquad \text{Lemme 2}}{\Gamma, a : \mathbf{2}, b : B(a) \to \mathbf{N}, c : \prod_{v:B(a)} E(b\ v)\ \vdash e(a,b,c) : E(\sup(a,b))}}{\Gamma\ \vdash \mathrm{rec}_{\mathbf{N}}(E,e,k) : E(k)}$$

Avec

$$\dfrac{\Gamma, a : \mathbf{1}, b : \mathbf{0} \to \mathbf{N}, c : \prod_{v:\mathbf{0}} E(b\ v) \vdash e(a,b,c) : E(\sup(0_{\mathbf{2}}, b))}{\Gamma, a : \mathbf{1}, b : B(\mathrm{inl}(a)) \to \mathbf{N}, c : \prod_{v:B(\mathrm{inl}(a))} E(b\ v) \vdash e(a,b,c) : E(\sup(a,b))}\ \text{Lemme 1}$$

et

$$\dfrac{\Gamma, a : \mathbf{1}, b : \mathbf{1} \to \mathbf{N}, c : \prod_{v:\mathbf{1}} E(b\ v) \vdash e(a,b,c) : E(\sup(1_{\mathbf{2}}, b))}{\Gamma, a : \mathbf{1}, b : B(\mathrm{inr}(a)) \to \mathbf{N}, c : \prod_{v:B(\mathrm{inr}(a))} E(b\ v) \vdash e(a,b,c) : E(\sup(\mathrm{inr}(a), b))}\ \text{Lemme 2}$$

On peut simplifier les prémisses du Lemme 1 en remarquant que ses hypothèses sont triviales et que $\sup(0_{\mathbf{2}}, b) \equiv 0$ pour toute fonction $b : \mathbf{0} \to \mathbf{N}$, ce qui donne après simplification :

$$\Gamma \vdash e_0 : E(0).$$

De même, pour le Lemme 2 en remarquant qu'une fonction $b : \mathbf{1} \to \mathbf{N}$ est équivalente à choisir un $k : \mathbf{N}$ et que $c : \prod_{v:\mathbf{1}} E(b\ v)$ est similaire à $c : E(k)$, on peut réécrire la prémisse en

$$\Gamma, k : \mathbf{N}, p : E(k) \vdash e_s(k,p) : E(s\ k).$$

Finalement, on peut décomposer la fonction e du récurseur : $e_0 : E(0)$ et $e_s : \prod_{k:\mathbf{N}} E(k) \to E(s\ k)$, ce qui permet de réécrire la règle d'élimination sous la forme plus simple :

$$\dfrac{\Gamma \vdash k : \mathbf{N} \quad \Gamma \vdash e_0 : E(0) \quad \Gamma, k : \mathbf{N}, p : E(k) \vdash e_s(k,p) : E(s\ k)}{\Gamma \vdash \mathrm{rec}_{\mathbf{N}}(E, e_0, e_s, k) : E(k)}\ \mathbf{N}\text{ e}$$

On retrouve bien le principe de récurrence usuel, en effet pour prouver un prédicat sur les entiers $\mathbf{N}$ il suffit de le prouver sur 0 puis qu'il est héréditaire *ie.* s'il est vérifié sur un entier $k : \mathbf{N}$ alors il est vérifié sur $s\ k$ l'entier suivant.

I.3.11 Listes

Un autre exemple de type inductif est le type des listes sur un type $A : \mathcal{U}_i$, que l'on note $L(A)$. Pour la construire on utilise alors le W-type $W_{x:\mathbf{1}+A} B(x)$ où $B(\mathrm{inl}(\star)) \equiv \mathbf{0}$ et $B(\mathrm{inr}(a)) \equiv \mathbf{1}$,

autrement dit on pose

$$L(A) \equiv W_{x:\mathbf{1}+A}\, \mathrm{rec}_{1+A}(\mathcal{U}_i, \lambda(x:\mathbf{1}).\ \mathbf{0}, \lambda(a:A).\ \mathbf{1}, x).$$

De là, on déduit la règle de formation de $L(A)$:

$$\dfrac{\dfrac{\Gamma \vdash A:\mathcal{U}_i \quad \dfrac{\Gamma\ \mathrm{ctx}}{\Gamma \vdash \mathbf{1}:\mathcal{U}_i}}{\Gamma \vdash \mathbf{1}+A:\mathcal{U}_i} \quad \dfrac{\Gamma, x:\mathbf{1}+A \vdash \mathrm{rec}_{\mathbf{1}+A}(\mathcal{U}_i, \lambda(y:\mathbf{1}).\ \mathbf{0}, \lambda(y:A).\ \mathbf{1}, x):\mathcal{U}_i}{\Gamma, x:\mathbf{1}+A \vdash B(x):\mathcal{U}_i}}{\Gamma \vdash L(A):\mathcal{U}_i}$$

Plus simplement,

$$\dfrac{\Gamma \vdash A:\mathcal{U}_i}{\Gamma \vdash L(A):\mathcal{U}_i}\, L\ \mathrm{f}$$

Ensuite pour construire un élément on a deux constructeurs :

— La liste vide, notée () que l'on définit comme $() :\equiv \sup(\mathrm{inl}(\star), \mathrm{rec}_{\mathbf{0}}(L(A))) : L(A)$,

— L'ajout d'un élément en tête de liste via la fonction $c : A \to L(A) \to L(A)$ qui est définit de la manière suivante :

$$c :\equiv \lambda(a:A).\ \lambda(l:L(A)).\ \sup(a, \lambda(x:\mathbf{1}).\ l)\,.$$

que l'on peut également noter :: de manière infixe.

Ainsi, une liste $l \equiv (a_1, \ldots, a_k) : L(A)$ est définie par

$$l :\equiv c\ a_1\ (a_2, \ldots, a_k) \equiv a_1 :: (a_2, \ldots, a_k) \equiv \sup(a_1, \lambda(x:\mathbf{1}).\ (a_2, \ldots, a_k)).$$

On remarque alors une structure d'arbre dans le type des listes, dont la racine est () et la fonction donnant les successeurs d'un noeud l est la fonction $\lambda(a:A).\ c\ a\ l : A \to L(A)$.

On a donc deux règles d'introduction correspondant à ces deux cas.

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash \mathbf{1}:\mathcal{U}_i \quad \Gamma \vdash A:\mathcal{U}_i \quad \Gamma \vdash \star:\mathbf{1}}{\Gamma \vdash \mathrm{inl}(\star):\mathbf{1}+A} \quad \dfrac{\Gamma \vdash \mathrm{rec}_{\mathbf{0}}(L(A)):\mathbf{0}\to L(A)}{\Gamma \vdash \mathrm{rec}_{\mathbf{0}}(L(A)):B(\mathrm{inl}(\star))\to L(A)}}{\Gamma \vdash \sup(\mathrm{inl}(\star), \mathrm{rec}_{\mathbf{0}}(L(A))):L(A)}}{\Gamma \vdash ():L(A)}$$

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash \mathbf{1}:\mathcal{U}_i \quad \Gamma \vdash A:\mathcal{U}_i \quad \Gamma \vdash a:A}{\Gamma \vdash \mathrm{inr}(a):\mathbf{1}+A} \quad \dfrac{\dfrac{\dfrac{\Gamma \vdash l:L(A)}{\Gamma, x:\mathbf{1} \vdash l:L(A)}}{\Gamma \vdash \lambda(x:\mathbf{1}).\ l:\mathbf{1}\to L(A)}}{\Gamma \vdash \lambda(x:\mathbf{1}).\ l:\mathrm{rec}_{\mathbf{1}+A}(\mathcal{U}_i, \mathbf{0}, \mathbf{1}, \mathrm{inr}(a))\to L(A)}}{\Gamma \vdash \lambda(x:\mathbf{1}).\ l:B(\mathrm{inr}(a))\to L(A)}}{\Gamma \vdash \sup(\mathrm{inr}(a), \lambda(x:\mathbf{1}).\ l):L(A)}}{\Gamma \vdash a :: l : L(A)}$$

Plus simplement,

$$\frac{\Gamma \vdash A : \mathcal{U}_i}{\Gamma \vdash () : L(A)}\,() \text{ i} \qquad \frac{\Gamma \vdash A : \mathcal{U}_i \quad \Gamma \vdash a : A \quad \Gamma \vdash l : L(A)}{\Gamma \vdash a :: l : L(A)}\,c \text{ i}$$

A partir de la règle d'élimination des W-types, on peut déduire la règle d'élimination du type $L(A)$:

$$\frac{\Gamma \vdash l : L(A) \qquad \dfrac{\text{Lemme 1} \quad \text{Lemme 2}}{\Gamma, a : \mathbf{1} + A, b : B(a) \to L(A), c : \prod_{v:B(a)} E(b\ v) \vdash e(a,b,c) : E(\sup(a,b))}}{\Gamma \vdash \mathrm{rec}_{L(A)}(E,e,l) : E(l)}$$

$$\frac{\Gamma, a : \mathbf{1}, b : \mathbf{0} \to L(A), c : \prod_{v:\mathbf{0}} E(b\ v) \vdash e(a,b,c) : E(())}{\Gamma, a : \mathbf{1}, b : B(\mathrm{inl}(a)) \to L(A), c : \prod_{v:B(\mathrm{inl}(a))} E(b\ v) \vdash e(a,b,c) : E(\sup(\mathrm{inl}(a), b))}\ \text{Lemme 1}$$

$$\frac{\Gamma, a : A, b : \mathbf{1} \to L(A), c : \prod_{v:\mathbf{1}} E(b\ v) \vdash e(a,b,c) : E(a :: b)}{\Gamma, a : A, b : B(\mathrm{inr}(a)) \to L(A), c : \prod_{v:B(\mathrm{inr}(a))} E(b\ v) \vdash e(a,b,c) : E(\sup(\mathrm{inr}(a), b))}\ \text{Lemme 2}$$

Le lemme 1 correspond à une preuve de E sur la liste vide $()$ dont les hypothèses sont triviales. On peut donc réécrire sa prémisse sous une forme plus simple

$$\Gamma \vdash e_0 : E(()).$$

De même, on peut simplifier le lemme 2 en remarquant que $b : \mathbf{1} \to L(A)$ est équivalent à une liste $l \equiv b\ \star$, et que $c : \prod_{v:\mathbf{1}} E(b\ v)$ est une preuve de $E(l)$:

$$\Gamma, a : A, l : L(A), p : E(l) \vdash e_s(a,l,p) : E(a :: l).$$

On peut aussi réécrire e en le décomposant en $e_0 : E(())$ et $e_s : \prod_{a:A} \prod_{l:L(A)} E(l)$ pour correspondre à ces deux cas, ce qui donne finalement la règle

$$\frac{\Gamma \vdash l : L(A) \quad \Gamma \vdash e_0 : E(()) \quad \Gamma, a : A, l : L(A), p : E(l) \vdash e_s(a,l,p) : E(a :: l)}{\Gamma \vdash \mathrm{rec}_{L(A)}(E, e_0, e_s, l) : E(l)}\,L \text{ e}$$

Ce qui correspond à la preuve par induction sur les listes, à savoir que si un prédicat E sur $L(A)$ est vraie sur la liste vide $()$ et qu'il est héréditaire *ie.* si E est vraie sur une liste l et si $a : A$ alors E est vérifié sur $a :: l$, alors E est vérifié sur toutes les listes.

I.4 Correspondance preuves – programmes

Dans la théorie des types, les propositions sont des types et les preuves de ces propositions sont des éléments du type correspondant. Ainsi, prouver un théorème correspond à construire un élément (ou un terme) du type correspondant à l'énoncé du théorème. On a décrit comment construire des types de manière générale et ces constructions peuvent être utilisés pour construire des propositions, comme décrit dans le tableau suivant.

Logique	Type
$A \implies B$	$A \to B$
$A \wedge B$	$A \times B$
$A \vee B$	$A + B$
$\bot$	$\mathbf{0}$
$\top$	$\mathbf{1}$
$\neg A$	$A \to \mathbf{0}$
$\forall x : A, P(x)$	$\prod_{x:A} P(x)$
$\exists x : A, P(x)$	$\sum_{x:A} P(x)$

On peut donc voir les règles d'inférences comme des règles de construction des preuves dans la théorie des types. Par exemple, pour prouver une implication $A \implies B$, en théorie des types il faut construire une fonction $f : A \to B$. Or pour construire une fonction f du type $A \to B$, on doit donner une manière de construire une preuve de B à partir d'une preuve de A, c'est-à-dire que si $a : A$ est un preuve de A, $f\ a$ est une preuve de B et ce pour toute preuve a de A. On peut alors voir la fonction f comme un algorithme qui donne une preuve de B à partir de n'importe quelle preuve de A.

Ainsi, les règles d'introduction des types donnent une manière de construire des éléments du type en question, et donc si ce type représente une proposition alors on peut voir la règle d'introduction comme une règle de construction de preuve. De la même manière les règles d'élimination donnent la manière d'utiliser une hypothèse.

Enfin, les deux derniers types du tableau sont les quantificateurs universels et existentiels et leurs équivalents : les types dépendants. Le caractère dépendant de ces types se situe dans le fait qu'ils dépendent d'un paramètre : pour les fonctions dépendantes le codomaine dépend du point d'application, pour les paires dépendantes c'est le type de la deuxième coordonnées qui dépend de la première coordonnée. Le caractère dépendant de ces types permet notamment la construction de polymorphismes, comme détaillé dans la section sur les fonctions dépendantes.

Prouver une proposition en théorie des types est équivalent à construire un élément d'un certain type. Pour ce faire, les assistants de preuves permettent de simplifier la création de terme (ou d'éléments). Par exemple, la proposition

$$(\neg(p \wedge q) \implies r) \implies (\neg p \implies r) \wedge (\neg q \implies r).$$

correspond au type

$$\begin{aligned}&(\neg(p \times q) \to r) \to (\neg p \to r) \times (\neg q \to r)\\ \equiv\ &(((p \times q) \to \mathbf{0}) \to r) \to ((p \to \mathbf{0}) \to r) \times ((q \to \mathbf{0}) \to r).\end{aligned}$$

dont un terme est

$$\lambda h.\ (\lambda x.\ h\ (\lambda a.\ x\ (\pi_1\ a)), \lambda y.\ h\ (\lambda a.\ y\ (\pi_2\ a)))$$

En effet si on considère $h : ((p \times q) \to \mathbf{0}) \to r$, pour construire un terme de type $(p \to \mathbf{0}) \to r$ (*resp.* $(q \to \mathbf{0}) \to r$), on se donne $x : p \to \mathbf{0}$ (*resp.* $y : q \to \mathbf{0}$), puis on cherche à construire un terme de type r dans les deux cas. Pour ce faire, on applique la fonction h à un terme du type $(p \times q) \to \mathbf{0}$ que l'on doit construire. Or, $\pi_1\ a$ (*resp.* $\pi_2\ a$) est de type p (*resp.* q) si a est de type $p \times q$, donc $x\ (\pi_1\ a)$ (*resp.* $y\ (\pi_2\ a)$) est de type $\mathbf{0}$. Les fonctions $\lambda a.\ x\ (\pi_1\ a)$ et $\lambda a.\ y\ (\pi_2\ a)$ sont de type $p \times q \to \mathbf{0}$, on peut alors appliquer h à chacune de ses deux fonctions pour obtenir un terme de type r comme voulu.

La construction d'une telle fonction est parfois compliquée, c'est pourquoi les assistants de preuves possèdent un système de construction de termes à partir de commande plus simple ayant un sens qui se rapproche des règles habituelles de raisonnement. Reprenons l'exemple précédent. En Lean, le code suivant qui permet de construire un terme de ce type :

```
example : (¬(p ∧ q) → r) → (¬p → r) ∧ (¬q → r) := by
  intro h              -- Introduit un terme de type ¬(p ∧ q) → r)
  constructor          -- Sépare l'objectif en deux sous-objectifs
  <;> intro hn         -- Introduit un terme de type ¬p (resp. ¬q)
                       -- dans le premier (resp. second) sous-objectif
  <;> apply h          -- Applique h aux deux sous-objectifs
  <;> rintro ⟨hp, hq⟩ -- Introduit des termes de type p et q
  . exact hn hp
  . exact hn hq        -- Construit des termes de type 0
```

Le terme créé par Lean à partir de ce code est :

```
fun h =>
  ⟨fun hn => h fun a => And.casesOn a fun hp hq => hn hp,
   fun hn => h fun a => And.casesOn a fun hp hq => hn hq⟩
```

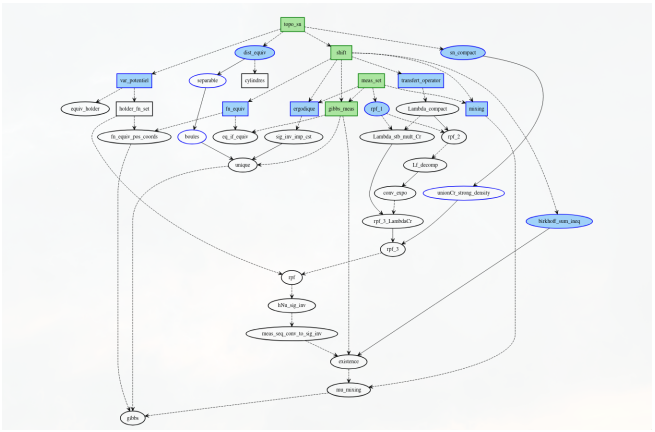
II.

Formalisation du théorème de Bowen en Lean

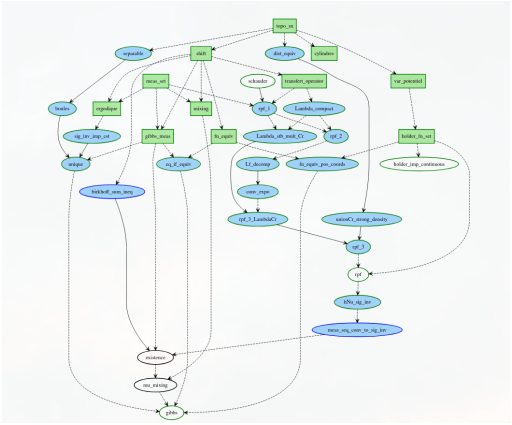
Cette section est un retour d'expérience sur l'utilisation de Lean en tant qu'outil de formalisation pour les mathématiques.

Durant ce TER, j'ai utilisé Lean pour formaliser le théorème de Bowen, que j'ai eu l'occasion d'étudier durant un stage et dont une preuve figure dans mon rapport de stage en annexe (preuve initialement dûe à R. Bowen). Ce théorème établit l'existence et l'unicité d'une certaine mesure de probabilité dite de Gibbs sur un espace métrique particulier, et donc ce théorème fait intervenir notamment de la topologie et de la théorie de la mesure.

Pour formaliser ce théorème, j'ai eu recours à des outils extérieurs à Lean comme leanblueprint [Mas] qui permet à partir d'un texte mathématiques d'obtenir un graphe décrivant les liens entre les différents lemmes, théorèmes et définitions intermédiaires et d'en donner l'état d'avancement dans le code Lean associé.



(a) État initial du graphe



(b) État final du graphe

Les cases rectangulaires sont les définitions, les ellipses sont les lemmes, théorèmes, corollaires et propositions. Les couleurs en fond indiquent l'état d'avancement des preuves : le vert foncé pour pour les lemmes prouvés et dont les parents sont également prouvés, le vert clair pour les lemmes prouvés et le bleu pour les lemmes prêt à être prouvés (*ie.* toutes les définitions nécessaires sont formalisés). Enfin les couleurs des bordures indiquent l'état d'avancement des énoncés des lemmes et des définitions : le vert pour les cases dont l'énoncé est formalisé et le bleu pour les cases dont l'énoncé n'est pas formalisé mais prêt à l'être.

II.1 Formalisation de l'énoncé du théorème

Dans un premier temps, j'ai donc converti chaque définition et résultat nécessaire pour le théorème de Bowen en code Lean afin de pouvoir écrire l'énoncé du théorème. Pour ce faire on a besoin notamment de définir la notion de mesure de Gibbs puis on peut alors énoncé le théorème avec le code suivant :

```
class IsGibbs (ϕ : Bernoulli Z n → R) (μ : Measure (Bernoulli Z n))
extends InvariantProb μ where
  gibbs_prop : Exists P : R, Exists c c' : NNReal, c > 0 ∧ c' > 0 ∧
  Forall x : Bernoulli Z n, Forall m : N,
μ (cylinder x m) / nnexp (- P * m + Σ k ∈ Ico 0 m, ϕ (shift^[k] x))
  ∈ Icc (c : ENNReal) (c': ENNReal)
```

```
theorem gibbs (ϕ : Bernoulli Z n → R) [HolderLike ϕ] :
  ExistsUnique μ : Measure (Bernoulli Z n), IsGibbs ϕ μ := by sorry
```

Cependant, la preuve de ce résultat nécessite un certain nombre de lemmes intermédiaires qu'il faut également prouver et donc la preuve du théorème était trop longue par rapport au temps imparti.

On a donc décidé de se concentrer sur la formalisation des énoncés et sur la preuve d'un lemme important de topologie des espaces ultramétriques, pour expérimenter tout les aspects d'un assistant de preuves.

Pour la formalisation des énoncés il a donc fallu dans un premier temps adapter le rapport de stage au format demandé par l'outil lean-blueprint, en ajoutant des détails et en décomposant certains énoncés pour correspondre davantage à la structure du code Lean. L'étape suivante était de traduire chaque définition et lemme (sans les preuves) en un code le plus clair possible pour faciliter le travail d'écriture des preuves suivant l'ordre préconisé par le graphe de dépendance. Enfin, j'ai pu commencer l'écriture des morceaux de preuves pour les lemmes les plus simples. Finalement, au vu du nombre de preuve à convertir en code, il a fallu choisir un résultat dont la preuve était moins conséquente pour s'y consacrer.

II.2 Preuve d'un résultat topologique

Le choix qui a été fait était de prouver un théorème de topologie permettant de décomposer les ouverts d'un espace ultramétrique compact en une union disjointe de boules. Ce résultat était nécessaire pour réécrire la mesure des ouverts en somme de mesure de boules pour lesquelles on possède une estimation dans le cas des mesures de Gibbs.

Afin de prouver ce résultat, la première étape était de réécrire de manière la plus détaillée possible sa preuve et d'en donner les étapes intermédiaires pour ensuite les convertir en lemme dans Lean pour pouvoir écrire les preuves de ces lemmes. L'étape finale à donc été d'écrire la preuve du théorème en utilisant tout les lemmes précédents.

Cependant, la traduction des preuves en Lean ne se fait pas littéralement, il faut souvent adapter les énoncés pour les faire correspondre aux lemmes déjà existants et prouvés dans Mathlib (la bibliothèque de Lean contenant une large variété de théorèmes). Ainsi, nous avons donc transformés l'énoncé mathématique standard en un énoncé utilisant à la fois la théorie des types et des propriétés déjà présentes dans Mathlib.

Théorème. *Soit $\mathcal{O}$ un ouvert borné de E, alors il existe une partie $R \subseteq \mathcal{O}$ et une fonction $r : R \to \mathbf{R}_+^*$ telle que*

$$\mathcal{O} = \bigsqcup_{x \in R} B(x, r(x)).$$

Cet énoncé transformé nécessite l'introduction d'un type $S_{\mathcal{O}}$ construit comme l'ensemble quotient de l'ensemble $B_{\mathcal{O}}$ des boules incluses dans $\mathcal{O}$ par une relation d'équivalence, qui sera décrite dans la section suivante, et d'une fonction $\Phi : S_{\mathcal{O}} \to B$ qui à chaque classe d'équivalence associe l'union de ses éléments.

Théorème. *Soit $\mathcal{O}$ un ouvert borné de E. Alors la fonction Φ vérifie les propriétés suivantes :*

1. $\bigcup_{s:S_{\mathcal{O}}} \Phi(s) = \mathcal{O}$
2. *Si $s_1, s_2 : S_{\mathcal{O}}$ sont distincts alors $\Phi(s_1)$ et $\Phi(s_2)$ sont disjoints.*

On peut alors remarquer que les deux énoncés sont en fait équivalents, il s'agit simplement d'une reformulation plus maniable en Lean d'un même théorème, car elle fait intervenir des types et des fonctions largement utilisées dans Mathlib et qui sont donc plus faciles d'utilisation. Une fois converti en Lean, il peut s'écrire avec le code suivant :

```
theorem partition (hO : IsOpen O)
  (O_bdd : Exists (x : X), Exists r > 0, O ⊆ ball x r) :
  let Φ := iBall (Obdd O_bdd)
  ⋃ s, Φ s = O ∧ Pairwise (onFun Disjoint (fun s => (Φ s).1)) := by sorry
```

Une preuve de ce théorème se trouve dans la section suivante et une formalisation de ce théorème est disponible sur Github.

III.

Topologie des espaces ultramétriques

Pour montrer l'unicité dans le théorème de Bowen, on a besoin d'un résultat de topologie de certains espaces, appelés espaces ultramétriques, qui sont des espaces métriques où l'inégalité triangulaire habituelle est remplacée par une inégalité dite ultramétrique :

$$\forall x, y, z \in E, d(x, z) \leq \max\left(d(x, y), d(y, z)\right).$$

Cette inégalité donne de nombreux résultats topologiques très différents des espaces métriques, notamment sur les triangles et les boules.

Le théorème permettant de prouver l'unicité du théorème de Bowen est le suivant, et nous allons en donner une preuve dans cette section, puis une formalisation possible en Lean.

Théorème III.1. *Soit (E, d) un espace ultramétrique et $\mathcal{O}$ un ouvert borné de E, alors il existe une partie $R \subseteq \mathcal{O}$ et une application $r \colon R \longrightarrow \mathbf{R}_+^*$ tels que*

$$\mathcal{O} = \bigsqcup_{x \in R} B(x, r(x)).$$

Dans la suite on fixe (E, d) un espace ultramétrique Commençons par quelques lemmes décrivant les relations entres les boules dans ces espaces.

Lemme III.2. *Soit $x, y \in E$ et un réel $r > 0$, si $y \in B(x, r)$, alors $B(x, r) = B(y, r)$*

Démonstration. Supposons que $y \in B(x, r)$. Soit $z \in B(x, r)$, alors

$$d(y, z) \leq \max\left(d(y, x), d(x, z)\right) < \max\left(r, r\right) = r,$$

car $y, z \in B(x, r)$. Donc on a une inclusion : $B(x, r) \subseteq B(y, r)$.

Réciproquement comme $y \in B(x, r)$ on a $x \in B(y, r)$ par symétrie de la distance et donc avec le même raisonnement on a l'autre inclusion. Finalement, $B(x, r) = B(y, r)$. □

Lemme III.3. *Soit $x, y \in E$ et deux réels $r, s > 0$, alors il y a trois possibilités :*

1. $B(x, r) \subseteq B(y, s)$,
2. $B(y, s) \subseteq B(x, r)$,
3. $B(x, r) \cap B(y, s) = \emptyset$.

Démonstration. Supposons que $B(x, r) \cap B(y, s) \neq \emptyset$, alors soit $z \in B(x, r) \cap B(y, s)$. On peut alors écrire $B(x, r) = B(z, r)$ et $B(y, s) = B(z, s)$ grâce au lemme III.2, et ainsi on a bien le résultat voulu en fonction de l'ordre de r et s. □

Lemme III.4. *Soit $(x_i)_{i\in I}$ une famille de points de E et $(r_i)_{i\in I}$ une famille de réels strictement positifs. De plus, supposons qu'il existe $x \in E$ tel que $x \in B(x_i, r_i)$ pour tout $i \in I$ et que $R = \sup_{i\in I} r_i < \infty$. Alors*

$$\bigcup_{i\in I} B(x_i, r_i) = B(x, R).$$

Démonstration. Commençons par réécrire chaque boule en changeant son centre grâce au lemme III.2 et notons $R = \sup_{i\in I} r_i < \infty$, on a alors l'inclusion

$$\bigcup_{i\in I} B(x_i, r_i) = \bigcup_{i\in I} B(z, r_i) \subseteq B(z, R).$$

Réciproquement si $y \in B(z, R)$, alors il existe $i_0 \in I$ tel que $d(y, z) < r_{i_0} \leq R$ et donc $y \in B(z, r_{i_0}) \subseteq \bigcup_{i\in I} B(z, r_i)$. Finalement, l'union de ces boules est encore une boule. □

On notera dans la suite $\mathcal{O} \subseteq E$ un ouvert borné, et donc on a pour chaque $x \in \mathcal{O}$ un réel $r_x > 0$ tel que $B(x, r_x) \subseteq \mathcal{O}$. On a alors

$$\mathcal{O} = \bigcup_{x\in\mathcal{O}} B(x, r_x).$$

Remarque. On peut voir les $(r_x)_{x\in\mathcal{O}}$ comme une une application $r\colon \mathcal{O} \longrightarrow \mathbf{R}_+^*$.

Definition III.1. On note $B(\mathcal{O})$ l'ensemble des boules incluses dans $\mathcal{O}$, en particulier :

$$B(\mathcal{O}) := \{B(x, r) \mid x \in \mathcal{O} \wedge r > 0 \wedge B(x, r) \subseteq \mathcal{O}\} \subseteq \mathcal{P}(E).$$

Soit $\mathcal{U} \subseteq B(\mathcal{O})$, on définit une relation $\sim_{\mathcal{U}}$ (ou plus simplement $\sim$ s'il n'y a pas d'ambuiguité) sur $\mathcal{U}$ par :

$$\forall u, v \in \mathcal{U}, \quad u \sim_{\mathcal{U}} v \iff \exists w \in \mathcal{U}, u \cup v \subseteq w.$$

Proposition III.5. *Pour tout $\mathcal{U} \subseteq B(\mathcal{O})$, la relation $\sim_{\mathcal{U}}$ est une relation d'équivalence.*

Démonstration. Soit $u_1, u_2, u_3 \in \mathcal{U}$ trois boules.

Pour la reflexivité, on considère $w = u_1 \in \mathcal{U}$ et on a bien $u_1 \sim u_1$.

Pour la symétrie, la définition de la relation est clairement symétrique.

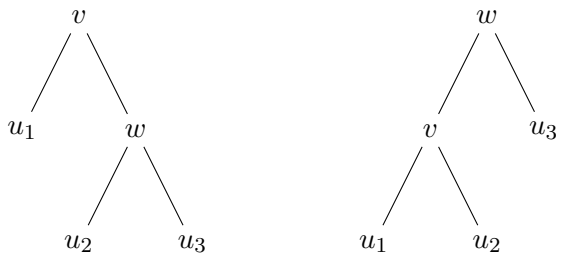
Enfin pour la transitivité, on suppose que $u_1 \sim u_2$ et $u_2 \sim u_3$. On a alors $v, w \in \mathcal{U}$ tels que

$$u_1 \subseteq v, u_2 \subseteq v, u_2 \subseteq w \text{ et } u_3 \subseteq w.$$

Ainsi, $\emptyset \neq u_2 \subseteq v \cap w$, or v et w sont des boules d'un espace ultramétrique. On a alors deux cas :

— $v \subseteq w$ dans ce cas on a $u_1 \overset{v}{\sim} u_3$

— $w \subseteq v$ et alors on a $u_1 \overset{w}{\sim} u_3$



Donc $\sim_{\mathcal{U}}$ est bien une relation d'équivalence sur $\mathcal{U}$. □

Pour la suite de la section on fixe $\mathcal{U} \subseteq B(\mathcal{O})$ un sous-ensemble de l'ensemble des boules incluses dans $\mathcal{O}$.

Definition III.2. Soit $u \in \mathcal{U}$, la classe d'équivalence de u pour la relation $\sim$ est noté $\bar{u}$. On introduit également $\mathcal{B}_u \in B(\mathcal{O})$ l'union des éléments de la classe d'équivalence de u :

$$\mathcal{B}_u := \bigcup_{u \sim u'} u' \subseteq \mathcal{O}.$$

Remarquons que $\mathcal{B}_u$ est une boule dès lors que cette union est bornée (d'après le lemme III.4), or cette $\mathcal{B}_u \subseteq \mathcal{O}$ et $\mathcal{O}$ est borné donc $\mathcal{B}_u$ est toujours une boule.

Definition III.3. Soit R un système complet de représentant pour la relation $\sim$. On pose alors

$$\mathcal{U}' := \mathcal{U} \cup \{\mathcal{B}_u \mid u \in R\} \subseteq B(\mathcal{O}).$$

car $\mathcal{B}_u \in B(\mathcal{O})$ pour chaque $u \in R$.

On considère désormais $\sim'=\sim_{\mathcal{U}'}$ qui est encore une relation d'équivalence, et soit R' un système complet de représentant pour la relation $\sim'$.

Théorème III.6. *Soit $\mathcal{O}$ un ouvert borné de E. Avec les notations introduites précédemment on a*

$$\mathcal{O} = \bigsqcup_{u \in R'} \mathcal{B}_u.$$

Démonstration. On commence par montrer que cette union est bien égale à $\mathcal{O}$. En considérant $\mathcal{U} = \{B(x, r_x) \mid x \in \mathcal{O}\}$,

$$\mathcal{O} = \bigcup_{u \in \mathcal{U}} u = \bigcup_{u \in \mathcal{U}'} u = \bigcup_{u \in R'} \left(\bigcup_{v \sim' u} v \right) = \bigcup_{u \in R'} \mathcal{B}_u$$

Il reste encore à montrer que cette union est disjointe. Supposons que $\mathcal{B}_u \cap \mathcal{B}_v \neq \emptyset$. Alors on a $\mathcal{B}_u \subseteq \mathcal{B}_v$ ou $\mathcal{B}_v \subseteq \mathcal{B}_u$ par le lemme III.3. Dans les deux cas on a $u \sim' v$ avec $w = \mathcal{B}_v \in \mathcal{U}'$ dans le premier cas et $w = \mathcal{B}_u \in \mathcal{U}'$ dans le second. Or R est un système complet de représentant pour $\sim'$ et donc nécessairement $u = v$. Finalement les $(\mathcal{B}_u)_{u \in R'}$ sont deux à deux disjoints et leur union est égale à $\mathcal{O}$, ce qui conclut la preuve du théorème III.1. □

Bibliographie

[24] *Martin-Löf dependent type theory in nLab*. 2024. URL : https://ncatlab.org/nlab/show/Martin-L%C3%B6f+dependent+type+theory.

[Bow75] Rufus Bowen. « Equilibrium states and the ergodic theory of Anosov diffeomorphisms ». In : *Lecture notes in mathematics* 470 (1975), p. 11-25.

[Car19] Mario Carneiro. « The Type Theory of Lean ». In : (2019). URL : https://github.com/digama0/lean-type-theory/releases.

[Mar21] Per Martin-Löf. « Intuitionistic Type Theory : Notes by Giovanni Sambin of a series of lectures given in Padova, June 1980 ». In : (2021).

[Mas] Patrick Massot. *Lean blueprints*. URL : https://github.com/PatrickMassot/leanblueprint.

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory : Univalent Foundations of Mathematics*. Institute for Advanced Study : https://homotopytypetheory.org/book, 2013.