Repenser la bibliothèque réelle de Coq : vers une formalisation de l'analyse classique mieux adaptée

Catherine Lelay

UNIVERSITÉ PARIS-SUD

ÉCOLE DOCTORALE INFORMATIQUE DE PARIS-SUD (ED 427)
LRI - Laboratoire de Recherche en Informatique
INRIA Saclay - Île-de-France - TOCCATA

***DISCIPLINE** INFORMATIQUE*

THÈSE DE DOCTORAT

soutenue le 15/06/2015

par

Catherine LELAY

Repenser la bibliothèque réelle de Coq : vers une formalisation de l'analyse classique mieux adaptée

Directeur de thèse :	Sylvie BOLDO	Chargée de Recherche, Inria Saclay – Île-de-France
Co-directeur de thèse :	Guillaume MELQUIOND	Chargé de Recherche, Inria Saclay – Île-de-France
Composition du jury :		
Rapporteurs :	Yves BERTOT	Directeur de Recherche, Inria Sophia Antipolis
	Herman GEUVERS	Professeur, Radboud University Nijmegen
Examinateurs :	Nicolas Brisebarre	Chargé de recherche, École Normale Supérieure de Lyon
	César Muñoz	Chercheur, NASA Langley Research Center
	Nicolas Thiéry	Professeur, Université Paris-Sud
Membre invité :	Micaela Mayero	maître de conférences HDR, IUT de Villetaneuse

Table des matières

Chapitre 1

Introduction

1.1 Preuve formelle et analyse réelle

Il est systématique en mathématiques de démontrer ses résultats pour s'assurer de leur exactitude. Malheureusement, ces démonstrations peuvent être longues et compliquées, ce qui peut entraîner des erreurs de raisonnement difficiles à repérer [Lec35]. La preuve formelle permet de réduire très fortement ce risque. Il s'agit de traduire les démonstrations mathématiques dans un langage suffisamment rigoureux pour pouvoir être lues par un ordinateur. L'ordinateur va alors vérifier que chacune des étapes de la démonstration est permise par les règles logiques du système. Les logiciels utilisés pour réaliser de telles démonstrations s'appellent des assistants de preuve. Ce désir de faire de telles démonstrations vérifiées par ordinateur est arrivé très tôt dans l'histoire de l'informatique. En effet, l'un des premiers assistants de preuve était Automath en 1967.

Depuis, de nombreux systèmes de preuve formelle ont été développés. Coq est l'un de ces systèmes. Il est basé sur la logique intuitionniste d'ordre supérieur. La définition des différents objets mathématiques utilise un lambda-calcul avec des types dépendants et du pattern-matching. Une preuve est l'application successive de lemmes déjà démontrés. À chaque étape, Coq vérifie que le lemme est applicable puis demande à l'utilisateur de démontrer chacune de ses hypothèses. Une preuve est terminée lorsque toutes les hypothèses ont été démontrées. Pour que l'utilisateur n'ait pas besoin de redémontrer des théorèmes usuels, Coq est fourni avec une bibliothèque couvrant de nombreux champs des mathématiques. L'un de ces champs est l'analyse réelle.

La bibliothèque d'analyse réelle de Coq s'appuie sur une axiomatisation classique des nombres réels. Les notions couvertes correspondent à peu de choses près au programme d'analyse de Terminale. Cette bibliothèque comprend la définition des limites de suites et de fonctions, des dérivées et des intégrales de Riemann ainsi que de nombreux théorèmes pour les manipuler. Elle contient également des notions plus avancées comme les séries et séries entières, ainsi que les suites et séries de fonctions. Certains domaines ont cependant été laissés de côté comme les équations différentielles, les séries de Fourier ou les nombres complexes.

Les mathématiques pures ne sont cependant pas le seul usage de la bibliothèque d'analyse réelle de Coq. Elle peut également être utilisée pour démontrer que des programmes sont corrects, c'est-à-dire que le résultat calculé correspond au résultat attendu. En effet, les propriétés de base des opérations arithmétiques ne sont pas forcément suffisantes pour démontrer la correction d'un programme ; des théorèmes d'analyse peuvent être nécessaires. Ainsi, cette thèse provient, à l'origine, d'un besoin dans la preuve d'un programme servant à calculer une solution de l'équation des ondes en dimension 1.

1.2 Les défauts de la bibliothèque standard de Coq

La bibliothèque d'analyse réelle a été développée par M. Mayero en 2001 [May01]. Cette bibliothèque n'a pas beaucoup évolué depuis. Elle souffre de nombreux défauts que nous allons lister ici. Un certain nombre de ces défauts apparaissent lors de l'étude de la fonction γ définie par

$$\forall x, t \in \mathbb{R}, \quad \gamma(x,t) = \frac{1}{2c} \int_0^t \int_{x-c(t-\tau)}^{x+c(t-\tau)} f(\xi,\tau)\, d\xi\, d\tau.$$

Cette fonction est solution de l'équation des ondes en dimension 1. La preuve du programme mentionné ci-dessus nécessite de la formaliser en Coq et de calculer ses dérivées.

L'un des premiers problèmes qu'un utilisateur peut rencontrer est l'obligation d'utiliser des types dépendants pour écrire les intégrales et les dérivées. Ainsi, avant de pouvoir utiliser l'intégrale d'une fonction, il doit démontrer que cette fonction est intégrable. Cette démonstration est en effet indispensable pour écrire l'intégrale. Voici le type Coq de la fonction renvoyant la valeur de l'intégrale :

```
RiemannInt : forall (f : R -> R) (a b : R) (pr : Riemann_integrable f a b), R.
```

Elle prend en argument une fonction réelle `f` à une variable, les bornes de l'intégrale `a` et `b` et une preuve `pr` que la fonction est intégrable entre ces bornes. Une écriture mathématique de cette fonction pourrait être $\int_a^b(f, \mathtt{pr})$.

Cette intégrale est lourde à manipuler à cause de ce terme de preuve. Par exemple, concernant l'addition de deux fonctions f et g intégrables, l'utilisateur doit dans un premier temps démontrer un lemme `pr` disant que la somme $f+g$ de ces deux fonctions est intégrable. Ce lemme est ensuite nécessaire pour énoncer que l'intégrale de la somme est égale à la somme des intégrales $\int(f+g, \mathtt{pr}) = \int(f, \mathtt{pr\_f}) + \int(g, \mathtt{pr\_g})$. En ce qui concerne la fonction γ, la situation est pire car elle fait intervenir une intégrale double. En effet, cela oblige à démontrer plusieurs lemmes intermédiaires juste pour définir γ.

```
Lemma RInt1_f :
  forall a b tau, Riemann_integrable (fun xi => f xi tau) a b.
Definition gamma0 x t tau :=
  RiemannInt (RInt1_f (x - c * (t - tau)) (x + c * (t - tau)) tau).
Lemma RInt_gamma0 :
  forall x tau t, Riemann_integrable (fun tau => gamma0 x t tau) 0 tau.
Definition gamma x t :=
  1 / (2 * c) * @RiemannInt (fun xi => f xi tau) a b (RInt_gamma0 x t t).
```

L'utilisateur peut également rencontrer des difficultés pour trouver les lemmes existants afin de les utiliser dans ses démonstrations. L'une des principales raisons est le nommage arbitraire de certains théorèmes. Par exemple, le théorème disant que $\int_a^a(f, \mathtt{pr}) = 0$ porte le nom peu explicite `RiemannInt_P9`. Ce problème se pose aussi pour les tactiques. Ainsi, il existe dans la bibliothèque d'analyse réelle la tactique `reg` permettant de démontrer automatiquement la dérivabilité et la continuité d'une fonction. Cette tactique n'est malheureusement ni documentée ni décrite dans un article.

Une autre difficulté importante est le manque d'homogénéité dans la formulation des théorèmes. Par exemple, la convergence des séries entières est exprimée tantôt en utilisant la convergence des sommes partielles comme dans la définition de arc-tangente, tantôt en utilisant une définition dédiée comme dans le lemme `GP_infinite`. Le manque d'homogénéité se retrouve également dans les définitions. Ainsi, pour désigner des paramètres réels strictement positifs, on peut trouver soit le type `posreal` comme dans la définition de l'intégrale, soit une hypothèse comme dans la définition de la convergence d'une suite, soit les deux selon le paramètre comme dans la définition de la dérivabilité d'une fonction.

Par ailleurs, certains domaines de l'analyse réelle ne sont pas couverts, ou très partiellement, par la bibliothèque standard. Par exemple, le cas des limites des suites et fonctions vers $\pm\infty$ n'est formalisé que dans un seul cas : les suites réelles tendant vers $+\infty$, et seuls trois lemmes utilisent cette notion. Un autre exemple est que la bibliothèque standard se limite à l'étude des fonctions à une variable. Ainsi, certaines notions plus avancées comme les intégrales à paramètres dont nous avons besoin pour définir et dériver la fonction γ ne sont pas abordées. Cette étude a nécessité plusieurs centaines de lignes de preuve. Qui plus est, la tactique `reg` n'a été d'aucune utilité pour automatiser ces preuves car elle ne gère pas les intégrales.

Un dernier défaut de la bibliothèque standard est son manque de généralité. Par exemple, de nombreuses démonstrations concernant les limites de suites et les limites de fonctions, comme l'opposé ou la somme, auraient pu être factorisées. De plus, cette bibliothèque ne permet pas d'aborder facilement des problèmes concernant les nombres complexes et les matrices. En effet, alors que de nombreuses démonstrations concernant les fonctions réelles et complexes sont semblables sur le papier, la formalisation actuelle ne permet pas de généraliser les théorèmes existants.

1.3 Coquelicot

Pour pallier les défauts de la bibliothèque standard décrits ci-dessus, j'ai développé au cours de ma thèse la bibliothèque Coquelicot. C'est une bibliothèque Coq d'analyse classique et toutes les définitions introduites ont été prouvées équivalentes à leurs versions de la bibliothèque standard. Cela permet d'utiliser la puissance de la bibliothèque Coquelicot sur des énoncés de la bibliothèque standard. Elle n'ajoute pas d'axiome et se base seulement sur ceux définissant les nombres réels de Coq. De plus, plusieurs cas

concrets ont été étudiés afin de vérifier l'utilisabilité de la bibliothèque (voir chapitre 5). Par exemple, l'exhaustivité et la facilité d'utilisation des théorèmes de base ont été testées sur une épreuve de Baccalauréat de Mathématiques, passée en même temps que les élèves.

L'une des réponses à l'usage des types dépendants a été la conception de fonctions totales permettant d'écrire facilement les limites, les dérivées et les intégrales (voir section 3.5). Ces fonctions permettent d'écrire des expressions contenant une dérivée ou une intégrale avant de justifier la dérivabilité ou l'intégrabilité. Elles renvoient une valeur arbitraire dans le cas où la dérivée ou l'intégrale n'est pas définie. Pour revenir sur l'exemple de la fonction γ, celle-ci est définie par

```
Definition gamma x t :=
  1/(2*c) * RInt (fun tau => RInt (fun xi => f xi tau)
                                  (x - c * (t - tau))
                                  (x + c * (t - tau))) 0 t.
```

Dans cette formule, `RInt : (R -> R) -> R -> R -> R` désigne la fonction totale permettant d'écrire l'intégrale de Riemann. Elle prend en argument la fonction à intégrer et les bornes de l'intégrale.

Tout au long de la conception de la bibliothèque Coquelicot, un effort a été fait pour définir toutes les fonctions partielles suivant le même patron (voir section 3.1). En effet, les limites, dérivées, intégrales et opérations arithmétiques dans $\overline{\mathbb{R}} = \mathbb{R} \cup \{\pm\infty\}$ ont été définies de la façon suivante : un prédicat caractérisant sa valeur, un prédicat d'existence et une fonction totale renvoyant la valeur. Cette approche entraîne la normalisation des noms des théorèmes et facilite ainsi la recherche de théorèmes par l'utilisateur.

La notion de limite est devenue l'un des concepts centraux de la bibliothèque Coquelicot. Par exemple, l'intégrale de Riemann est définie comme une limite (section 3.3). La définition de limite a été repensée pour traiter aussi bien le cas des limites finies ou infinies en un point fini ou infini (section 3.2). Pour un traitement homogène des définitions reposant sur la convergence, la notion de filtre a été introduite. Les filtres sont des collections d'ensemble permettant de représenter des voisinages (section 4.1). La continuité a été redéfinie en utilisant cette notion de limite.

Une hiérarchie algébrique a été conçue dans le but de pouvoir utiliser la bibliothèque Coquelicot sur des ensembles possédant des propriétés communes avec les nombres réels, comme par exemple les nombres complexes (section 4.2). J'ai ainsi formalisé des structures générales comme les groupes abéliens et les anneaux non commutatifs afin d'obtenir des énoncés de théorème suffisamment génériques. Sans être exhaustive, cette hiérarchie permet de couvrir aussi bien le cas des nombres complexes que des matrices.

La notion de différentiabilité a été formalisée dans la hiérarchie algébrique évoquée ci-dessus (section 4.3). Les différentielles étant par définition des fonctions linéaires continues, j'ai ajouté à la bibliothèque Coquelicot ce concept. Afin d'exprimer l'idée de meilleure approximation par une fonction linéaire, j'ai introduit la relation d'équivalence. Cette relation exprime le fait que la différence entre deux fonctions au voisinage d'un point donné est négligeable. Cette définition est basée sur la notion de prépondérance.

Dans le but de démontrer la différentiabilité d'intégrales à paramètre de la forme $(b, x) \mapsto \int_a^b f(t, x)\, dt$, j'ai été amenée à étudier leurs dérivées partielles et la continuité de ces dernières. Ces résultats ont été obtenus dans le cadre restreint de $\mathbb{R}^2$ (section 3.4). Afin de démontrer la dérivabilité par rapport à la variable x, j'ai formalisé les notions de compacité sur un pavé et de continuité uniforme.

Les preuves de dérivabilité des fonctions réelles étant simples mais fastidieuses, j'ai implémenté une tactique permettant de les automatiser (section 3.6). Cette tactique permet même de traiter des fonctions construites à partir d'intégrales à paramètre comme la fonction γ.

1.4 Plan du document

Le chapitre 2 présente différentes formalisations des nombres et de l'analyse réelle. Ce chapitre commence avec la présentation de Coq, des nombres réels et de la bibliothèque d'analyse réelle disponible dans la bibliothèque standard. Les sections suivantes sont consacrées à la présentation d'autres systèmes et formalisations. J'ai ainsi présenté la bibliothèque de Mizar, la bibliothèque de PVS développée par la NASA, les bibliothèques de différents systèmes de la famille HOL (HOL Light, HOL4, Isabelle/HOL et ProofPower-HOL), l'extension ACL2(r) du prouveur ACL2 et les bibliothèques d'analyse constructive de Coq : C-CoRN et MathClasses.

Les chapitres 3 et 4 détaillent la bibliothèque Coquelicot présentée dans la section précédente. Le chapitre 3 est centré sur les outils développés pour l'analyse réelle et le chapitre 4 présente la généralisation de la bibliothèque.

Le chapitre 5 présente différentes applications ayant guidé la conception de la bibliothèque Coquelicot. Les développements sur les intégrales à paramètre et la tactique de dérivation automatique ont ainsi été

testés lors de la démonstration du fait que la formule de d'Alembert est une solution de l'équation des ondes en dimension 1. Des propriétés sur les fonctions de Bessel ont été démontrées pour tester les séries entières. Enfin, la bibliothèque a été testée en temps réel sur les exercices d'analyse du Baccalauréat de Mathématiques 2013.

Pour finir, le chapitre 6 récapitule les contributions de cette thèse et présente quelques perspectives.

Chapitre 2

État de l'art

Les assistants de preuves sont arrivés assez tôt dans l'histoire de l'informatique : le premier système remarquable était Automath en 1967, mais il y a eu des travaux antérieurs [GOBS69]. Puis vinrent Boyer-Moore (1971), LCF (1972), Mizar (1973), et de nombreux autres systèmes. La première formalisation des nombres réels a suivi rapidement [Jut77]. Pour cet état de l'art, je me limite à des systèmes largement diffusés et disposant d'une bibliothèque standard d'analyse réelle : Mizar, PVS, HOL4, HOL Light, Isabelle/HOL, ProofPower-HOL et Coq, ou des bibliothèques ayant la même légitimité comme ACL2(r) pour ACL2, C-CoRN/MathClasses pour Coq et la bibliothèque de la NASA pour PVS. D'autre part, la bibliothèque Coquelicot ne traitant que d'analyse, je ne présente que les bibliothèques couvrant des mêmes domaines et laisse de côté les développements sur les probabilités et la théorie de la mesure.

La section 2.1 est consacrée à l'analyse réelle dans la bibliothèque standard de Coq. Je présente dans les sections suivantes les autres prouveurs (section 2.2), leurs définitions respectives des nombres réels (section 2.3), et pour finir les différents choix faits pour formaliser les notions usuelles d'analyse réelle (section 2.4). Je termine ce chapitre avec un tableau récapitulatif (page 34) présentant les principales caractéristiques de toutes les formalisations présentées.

2.1 L'analyse réelle en Coq

Le système formel Coq [1] est basé sur le calcul des constructions inductives (CIC) qui combine à la fois une logique d'ordre supérieur et un langage de programmation fortement typé [BC04]. Des programmes peuvent être extraits de preuves Coq vers des langages externes comme OCaml, Haskell, ou Scheme. En tant que système de preuve, Coq fournit des méthodes de preuves interactives, des algorithmes de décision et semi-décision, ainsi qu'un langage de tactique permettant à l'utilisateur de définir de nouvelles méthodes de preuve.

La bibliothèque support de Coq est structurée en deux parties : la bibliothèque initiale, qui contient des notions élémentaires de logique et des types de données, et la bibliothèque standard qui contient de nombreux développements et axiomatisations à propos des ensembles, des listes, des tris, de l'arithmétique, des nombres réels, etc.

Je présente dans la section 2.1.1 les types permettant d'exprimer les énoncés mathématiques en Coq. Puis je présente plus en détail les nombres réels définis dans la bibliothèque standard dans la section 2.1.2 et les différentes notions d'analyse réelle disponibles dans cette bibliothèque dans la section 2.1.3. Il faut noter qu'il existe une seconde bibliothèque majeure d'analyse réelle en Coq qui est présentée dans les sections 2.2.8 et 2.3.2.

2.1.1 Le type des énoncés et des preuves en Coq : `Prop` et `Type`

En Coq, tous les objets ont un type. Pour donner quelques exemples usuels : `nat` est le type des entiers unaires, `nat -> R` le type des suites à valeurs dans les réels et `3 <= 6` le type des démonstrations de $3 \leq 6$. De même, les énoncés mathématiques ont un type, ils sont classés en deux catégories : `Prop` et `Set`/`Type`.

Usuellement, les objets de types `Prop` sont les énoncés mathématiques et les propriétés. Par exemple :

Exemple 2.1. `3 <= 6 : Prop`

Exemple 2.2. `(forall A B : Prop, A \/ B) : Prop`

1. `http://coq.inria.fr/`

Logique	Prop	Type
$\top$	`True`	
$\bot$	`False`	
$\neg$`A`	`~ A`	`A -> False`
`A` $\Rightarrow$ `B`	`A -> B`	
`A` $\wedge$ `B`	`A /\ B`	`A * B`

`A` $\vee$ `B`	`A \/ B`	`{A} + {B}`
		`A + {B}`
		`A + B`

$\forall$`x` $\in$ `T, A`	**forall** `x [: T], A`	
$\exists$`x` $\in$ `T, A`	**exists** `x [: T], A`	`{ x : T | A }` `{ x : T & A }`

TABLEAU 2.1 – Opérateurs logiques dans `Prop` et `Type`

Exemple 2.3.
```
(forall P : nat -> Prop,
  (exists n : nat, P n) \/ (forall n : nat, ~ P n)) : Prop
```

Soit il est possible de démontrer ces énoncés (comme dans l'exemple 2.1), soit leur négation est vraie (comme dans l'exemple 2.2), soit il n'est pas possible de les démontrer en Coq sans hypothèse supplémentaire (comme dans l'exemple 2.3 qui sera développé dans la section 3.5). Après avoir été démontrés, ils peuvent être utilisés pour démontrer d'autres théorèmes.

Les objets de type `Type` (ou `Set`) sont utilisés pour définir les types des objets, comme le type des entiers naturels ou le type des fonctions dérivables :

Exemple 2.4 ($\mathbb{N}$)**.**
```
Inductive nat : Set :=
  | O : nat
  | S : nat -> nat.
```

Exemple 2.5 (type des fonctions dérivables en un point)**.**
```
Definition derivable_pt (f : R -> R) (x : R) : Set :=
  { l : R | derivable_pt_lim f x l }.
```

Exemple 2.6 (espaces métriques)**.**
```
Record Metric_Space : Type :=
  { Base : Type;
    dist : Base -> Base -> R;
    dist_pos : forall x y : Base, dist x y >= 0;
    dist_sym : forall x y : Base, dist x y = dist y x;
    dist_refl : forall x y : Base, dist x y = 0 <-> x = y;
    dist_tri : forall x y z : Base, dist x y <= dist x z + dist z y
  }.
```

L'exemple 2.5 se lit "la fonction `f` est dérivable au point `x`" et est issu de la bibliothèque standard d'analyse réelle. Il pourrait également être exprimé dans `Prop` par **exists** `l, derivable_pt_lim f x l`. La différence entre ces deux énoncés tient aux propriétés informatives de `Prop` et `Type`. En effet, les objets de type `Prop` sont purement logiques, alors que les objets de type `Type` sont informatifs : dans l'exemple 2.5, à l'issue d'une démonstration de `derivable_pt f x`, la valeur du nombre dérivé `l` sera conservée, alors qu'elle sera oubliée dans le cas de l'énoncé dans `Prop`. Ce mécanisme est utilisé dans le cadre de l'extraction de programme à partir de preuves Coq : à condition d'avoir instancié tous les axiomes et paramètres ayant permis de démontrer un théorème dans `Type`, il est possible d'obtenir un programme exécutable en OCaml, en Haskell ou en Scheme. Les parties logiques de la démonstration (dans `Prop`) sont "oubliées" pour ne conserver que la partie calculatoire. Les axiomes des réels de Coq utilisés dans mes travaux n'étant pas réalisables, je ne m'attarderai pas plus sur l'extraction de programmes. Plus de détails sont disponibles dans le Coq'Art [BC04] et dans la thèse de P. Letouzey [Let04].

Les différentes structures logiques permettant d'exprimer les propriétés mathématiques dans `Prop` et `Type` sont présentées dans le tableau 2.1. Pour les quantificateurs universels et existentiels, `T` est le type de la variable et j'ai utilisé la notation `[: T]` pour indiquer les cas où il est facultatif de le préciser, Coq étant capable le plus souvent de le déduire automatiquement. Les différentes notations pour la disjonction et les quantificateurs existentiels dans `Type` sont liées au type de `A` et `B` (`Prop` ou `Type`). Dans la bibliothèque

d'analyse réelle, les structures de `Type` les plus souvent utilisées sont `{A} + {B}` pour la disjonction et `{ x : T | A }` pour le quantificateur existentiel. Voici quelques exemples de leur utilisation :

Exemple 2.7 (nombre dérivé). *En reprenant l'exemple 2.5, si* `pr` *est une démonstration de* `derivable_pt f x`*, alors le nombre dérivé de* `f` *en* `x` *est* `derive_pt f x pr := proj1_sig pr` *avec* `proj1_sig` *la projection qui, étant donné un élément du type* `{ x : A | P x }` *renvoie un élément* `x` *tel que* `P x`*.*

Exemple 2.8 (fonction en escalier). *En utilisant le lemme* `Rlt_dec : forall r1 r2 : R, {r1 < r2} + {~r1 < r2}` *de type* `Set`*, on peut définir la fonction en escalier qui vaut 1 sur les réels strictement positifs et 0 ailleurs comme*

```
Definition pos_fct (x : R) : R := match Rlt_dec 0 x with
                                  | left _ => 1
                                  | right _ => 0
                                  end.
```

Il est important de noter qu'une telle définition n'est pas possible en analyse constructive en raison de la non-décidabilité de l'ordre sur les nombres réels.

2.1.2 Les réels de la bibliothèque standard

Voici un rapide historique de la bibliothèque standard d'analyse réelle de Coq. Elle a été développée dans un premier temps par Micaela Mayero [May01] dans le but de démontrer en Coq le théorème des trois intervalles. Elle contenait initialement une axiomatisation classique des nombres réels, des limites de suites et de fonctions, une définition de la dérivabilité, ainsi que des séries et séries entières permettant de définir quelques fonctions transcendantes. Cette bibliothèque a ensuite été complétée par Olivier Desmettre [Des02] avec une formalisation de l'intégrale de Riemann.

Les nombres réels sont définis comme un ensemble `R` dans `Set` contenant deux éléments distincts `R0` (représentant 0) et `R1` (représentant 1) et disposant d'une addition `Rplus`, d'un opposé `Ropp`, d'une multiplication `Rmult` et d'une relation d'ordre stricte `Rlt` (à valeur dans `Prop`). Des notations permettent de revenir à une écriture plus naturelle de ces objets. Cet ensemble est muni d'une structure d'anneau totalement ordonné donnée par 12 axiomes, dont :

```
Axiom Rplus_comm : forall r1 r2 : R, r1 + r2 = r2 + r1.
Axiom Rplus_opp_r : forall r : R, r + - r = 0.
Axiom Rplus_0_l : forall r:R, 0 + r = r.
Axiom Rmult_assoc : forall r1 r2 r3 : R, r1 * r2 * r3 = r1 * (r2 * r3).
Axiom Rmult_plus_distr_l : forall r1 r2 r3 : R, r1 * (r2 + r3) = r1 * r2 + r1 * r3.
Axiom Rmult_lt_compat_l : forall r r1 r2 : R, 0 < r -> r1 < r2 -> r * r1 < r * r2.
```

Parmi ces axiomes, il faut remarquer celui donnant la propriété « totalement ordonné » :

```
Axiom total_order_T : forall r1 r2 : R, {r1 < r2} + {r1 = r2} + {r1 > r2}.
```

Contrairement aux axiomes cités précédemment, celui-ci est à valeur dans `Set` au lieu de `Prop`. C'est également le premier à être classique. En effet, en analyse constructive, il n'est pas possible de démontrer que deux nombres réels sont soit égaux, soit différents, or cette propriété est impliquée par l'axiome `total_order_T`.

La structure de corps est obtenue en axiomatisant une fonction inverse `Rinv` :

```
Parameter Rinv : R -> R.
Axiom Rinv_l : forall r:R, r <> 0 -> Rinv r * r = 1.
```

On remarque que l'inverse multiplicatif est une fonction totale, définie même pour zéro, mais qu'aucun des axiomes n'implique de valeur pour ce point. Par exemple, l'axiome ci-dessus établit l'égalité $x^{-1}x = 1$ sous l'hypothèse $x \neq 0$. En fait, on peut démontrer en Coq que $0^{-1}0 = 0$, puisque 0^{-1} est un réel et que 0 est absorbant pour la multiplication. Cependant, $0^{-1} = 0$ n'est pas prouvable.

La complétude des nombres réels de Coq est donnée par l'existence d'une borne supérieure :

```
Definition is_upper_bound (E : R->Prop) (m : R) :=
  forall x : R, E x -> x <= m.
Definition is_lub (E : R -> Prop) (m : R) :=
  is_upper_bound E m /\ (forall b : R, is_upper_bound E b -> m <= b).
Axiom completeness : forall E : R -> Prop,
  (exists m : R, is_upper_bound E m) ->
  (exists x : R, E x) -> { m : R | is_lub E m }.
```

La formulation de `is_lub` est naturelle pour décrire la borne supérieure : c'est le plus petit des majorants. L'axiome `completeness` postule, que pour tout ensemble non vide et majoré, il existe (dans `Set`) une borne supérieure.

Pour finir la liste des axiomes de `R`, la propriété d'Archimède est donnée par une fonction semblable à la partie entière :

```
Parameter up : R -> Z.
Axiom archimed : forall r : R, IZR (up r) > r /\ IZR (up r) - r <= 1.
```

Cette formulation était bien adaptée pour démontrer le théorème des trois intervalles qui est la première application de la bibliothèque d'analyse réelle. Elle ne correspond cependant pas à la définition mathématique de la partie entière. En effet, `up` x est l'entier n tel que $n-1 \leq x < n$, alors que la partie entière supérieure de x est l'entier n tel que $n-1 < x \leq n$.

2.1.3 L'analyse réelle dans la bibliothèque standard

Comme signalé au début de la section 2.1.2, les nombres réels de la bibliothèque standard ont été formalisés dans le but de démontrer des théorèmes d'analyse réelle. Ainsi, les notions de limite de suites et de fonctions ont été formalisées, suivies par la notion d'intégrale de Riemann.

Limites de suites

La convergence d'une suite (u_n) vers une limite $\ell \in \mathbb{R}$ est définie de façon usuelle par

$$\texttt{Un\_cv}\ u\ \ell\ :=\ \forall \varepsilon \in \mathbb{R}_+^*,\ \exists N \in \mathbb{N},\ \forall n \in \mathbb{N},\ n \geq N \Rightarrow |u_n - \ell| < \varepsilon \tag{2.1}$$

où $\mathbb{R}_+^*$ est l'ensemble des réels strictement positifs. Les théorèmes usuels concernant l'ordre et les opérations arithmétiques ont été démontrés. Concernant l'existence d'une limite, elle a été démontrée dans le cas d'une suite croissante majorée. Cela a permis de démontrer l'équivalence entre la convergence d'une suite et le critère de Cauchy.

La notion de convergence vers $+\infty$ a également été définie, mais seul le fait que l'inverse d'une suite convergeant vers $+\infty$ converge vers 0 a été démontré.

Deux versions du théorème de Césaro ont été démontrées à partir de ces définitions. Dans sa version simple, ce théorème dit que, pour toute suite convergente la suite des moyennes arithmétiques $(1/n\ \sum_{k=0}^{n-1} u_k)_n$ converge vers la même limite. La version plus complexe de ce théorème prend en compte le cas de moyennes pondérées : si la somme des poids $b_n > 0$ tend vers $+\infty$, alors la moyenne $(\sum_{k=0}^{n} b_k \cdot u_k\ /\ \sum_{k=0}^{n} b_k\ )_n$ tendra vers la même limite que la suite u_n.

Limites de fonctions

La convergence est la seule notion d'analyse dans toute la bibliothèque `Reals` à ne pas être limitée aux nombres réels. Elle a été définie dans un espace métrique arbitraire (défini de façon usuelle), puis spécialisée dans l'espace métrique des réels muni de la distance $(x, y) \mapsto |y - x|$. Pour une fonction $f : X \to Y$, la convergence est définie par

$$\texttt{limit\_in}\ X\ Y\ f\ D\ x\ \ell\ :=\ \forall \varepsilon \in \mathbb{R}^+,\ \exists \delta \in \mathbb{R}^+,\ \forall y \in D,\ d_X(x, y) < \delta \Rightarrow d_Y(\ell, f(y)) < \varepsilon. \tag{2.2}$$

où d_X et d_Y sont les distances respectives des espaces métriques X et Y.

Cette définition ne prend pas en compte le fait que x devrait être dans l'adhérence du domaine D. Cela permet d'avoir une définition plus simple, mais l'unicité de la limite ℓ nécessite l'hypothèse supplémentaire que x appartient à l'adhérence du domaine D. Ce théorème d'unicité a été démontré dans l'espace métrique des nombres réels.

La limite ainsi définie est utilisée pour définir la continuité `continuity_pt` d'une fonction, ainsi qu'une définition de la fonction dérivée sur un domaine `D_in`. Cette dernière n'étant pas utilisée pour formaliser des théorèmes compliqués, je ne la détaillerai pas plus. Les théorèmes usuels concernant la continuité et la dérivabilité de la somme, du produit, de l'opposé et de l'inverse de fonctions ont été démontrés. La continuité de la racine carrée et la dérivabilité des fonctions puissance [2] ont également été démontrées.

2. $x \mapsto x^y$ avec $y \in \mathbb{N}$ et $y \in \mathbb{R}$

Différentiabilité

En plus de la définition de la dérivabilité évoquée dans le paragraphe précédent, une seconde définition basée sur le taux d'accroissement a été introduite :

$$\texttt{derivable\_pt\_lim}(f, x, \ell) :=$$
$$\forall \varepsilon \in \mathbb{R}^+,\ \exists \delta \in \mathbb{R}^+,\ \forall y \in \mathbb{R}\backslash\{x\},\ |y - x| < \delta \Rightarrow \left| \frac{f(y) - f(x)}{y - x} - \ell \right| < \varepsilon. \tag{2.3}$$

L'équivalence entre les deux définitions, `D_in` et `derivable_pt_lim`, a été démontrée. Afin d'exprimer la dérivabilité, le prédicat `derivable_pt`, déjà évoqué dans l'exemple 2.5, a été défini dans `Set` par `{ l : R | derivable_pt_lim f x l }` et est utilisé pour écrire le nombre dérivé :

```
Definition derive_pt f (x:R) (pr:derivable_pt f x) := proj1_sig pr.
```

Cette fonction permet d'écrire des énoncés lisibles à propos des nombres dérivés, mais elle est difficile à utiliser en pratique en raison du terme de preuve `pr`.

Comme pour la continuité, les théorèmes usuels concernant la dérivabilité de la somme, du produit, de l'opposé et de l'inverse de fonctions ont été démontrés. Les dérivées de toutes les fonctions de base définies dans la bibliothèque standard, comme la fonction exponentielle, ont été démontrées.

Cette définition est également utilisée pour formaliser l'intégrale de Newton :

```
Definition antiderivative (f g : R -> R) (a b : R) : Prop :=
  (forall x : R, a <= x <= b ->
     exists pr : derivable_pt g x, f x = derive_pt g x pr) /\
  a <= b.
```

Comme pour la dérivée `D_in`, je ne poursuivrai pas la description de cette intégrale car elle n'est pas utilisée dans le reste de la bibliothèque.

Séries et séries entières

Les séries et les séries entières ont été formalisées afin de définir les fonctions exponentielle, sinus et cosinus. Ces séries sont basées sur les sommes partielles de 0 à n et sont utilisées pour construire les séries en passant à la limite :

$$\sum_{n\in\mathbb{N}} u_n \text{ converge vers } \ell \text{ si } n \mapsto \sum_{k=0}^{n} u_k \text{ converge vers } \ell. \tag{2.4}$$

Les sommes partielles de 0 à n sont définies par récurrence par

$$\texttt{sum\_f\_R0}\ u\ 0 = u_0 \quad \text{et} \quad \texttt{sum\_f\_R0}\ u\ (n+1) = u_{n+1} + \texttt{sum\_f\_R0}\ u\ n$$

Les sommes partielles $\sum_{k=n}^{m} u_k$ sont définies à partir de `sum_f_R0` par `sum_f_R0` u_{k+n} $(m-n)$. On remarque que lorsque $m < n$, cette somme ne prend pas la valeur usuelle 0. En effet, la différence $m - n$ sur `nat` est alors égale à 0, et donc la somme entre n et m ainsi définie est égale à u_n.

La convergence des séries est définie avec le prédicat ad-hoc `infinite_sum` plutôt que comme la limite des sommes partielles à partir de 0. Ce choix empêche d'utiliser les théorèmes démontrés sur les suites sans passer par un lemme de compatibilité à définir par l'utilisateur. La majorité des résultats sur les séries de cette bibliothèque n'utilisent d'ailleurs pas ce prédicat et préfèrent utiliser la formule (2.1) `Un_cv (sum_f_R0 u) l` pour parler de série convergent vers un réel `l`. Cela entraîne des difficultés pour retrouver les théorèmes déjà démontrés.

Parmi les critères de convergence disponibles pour les séries, on peut citer le critère de Cauchy, le critère de d'Alembert ainsi que le critère des séries alternées. Les théorèmes concernant les opérations arithmétiques doivent être déduits par l'utilisateur des théorèmes correspondants sur les suites ainsi que des règles de réécriture sur les sommes partielles. On peut noter au passage que le produit de Cauchy sur les séries $(\sum a_n)(\sum b_n) = \sum(\sum_{k=0}^{n} a_k b_{n-k})$, même s'il intervient dans la démonstration de $e^x e^y = e^{x+y}$, doit également être redémontré par l'utilisateur à partir du lemme technique `cauchy_finite`.

Pour les séries entières, la bibliothèque standard définit également un prédicat ad-hoc nommé `Pser` qui est une spécialisation de `infinite_sum` aux séries de terme général $a_n x^n$. Le seul critère spécifique à la convergence des séries entières démontré est le critère de d'Alembert sous ses deux versions : convergence de $|a_{n+1}/a_n|$ vers 0 et vers $k > 0$. Comme pour les séries, la majorité des théorèmes sur les séries entières

doit être déduit par l'utilisateur des théorèmes sur les suites. Pour faire le lien entre les limites de suite et les séries entières, le lemme `tech12` montre que si la suite $\left(\sum_{k=0}^{n} a_k x^k\right)_n$ converge vers un réel ℓ alors la série entière $\sum a_n x^n$ converge vers le même réel. La réciproque n'est cependant pas disponible dans la bibliothèque standard.

Pour finir, aucun théorème concernant la dérivabilité et la continuité des séries entières n'a été explicitement démontré, mais ces propriétés peuvent l'être à partir des théorèmes sur les suites et les séries de fonctions. Encore une fois, c'est à l'utilisateur de faire les lemmes de compatibilité entre ces différentes parties de la bibliothèque.

L'intégrale de Riemann

L'intégrabilité au sens de Riemann a été définie en utilisant les intégrales sur les fonctions en escalier suivantes :

```
Record StepFun (a b:R) : Type := mkStepFun
  {fe :> R -> R; pre : IsStepFun fe a b}.
```

avec `IsStepFun fe a b` une preuve qu'il existe une subdivision `l` de l'intervalle $[a;b]$ et une liste de valeurs `lf` telle que sur chacun des sous-intervalles $]\mathtt{l}_i;\mathtt{l}_{i+1}[$, $\mathtt{fe}(x) = \mathtt{lf}_i$. L'intégrale d'une telle fonction en escalier est définie naturellement par $\sum_i (\mathtt{l}_{i+1} - \mathtt{l}_i)\,\mathtt{lf}_i$.

L'intégrabilité pour les fonctions quelconques est alors définie de façon traditionnelle :

$$\forall \varepsilon > 0,\ \exists \varphi_\varepsilon, \psi_\varepsilon : \texttt{StepFun a b},\ (\forall t \in [a;b],\ |f(t) - \varphi_\varepsilon(t)| \leq \psi_\varepsilon(t)) \wedge |\textstyle\int \psi_\varepsilon| < \varepsilon.$$

La valeur de l'intégrale est la limite des $\int \varphi_\varepsilon$ quand ε tend vers 0. Comme pour la dérivée, l'écriture de l'intégrale de Riemann nécessite une preuve que la fonction est intégrable.

Les théorèmes usuels sur l'intégration (relation de Chasles, linéarité, positivité) ont été démontrés. Le théorème fondamental de l'analyse

$$\int_a^b f'(t)\,dt = f(b) - f(a)$$

a été démontré pour les fonctions dérivables sur $\mathbb{R}$ de dérivée continue, également sur tout le domaine. La raison de ces hypothèses globales est le manque de flexibilité lié à l'usage des types dépendants pour écrire les fonctions dérivées et les intégrales.

Fonctions élémentaires

Comme cela a été évoqué plus haut, la fonction exponentielle et les fonctions trigonométriques ont été définies en utilisant les séries entières :

$$\exp(x) = \sum_{n\in\mathbb{N}} \frac{x^n}{n!}, \quad \cos(x) = \sum_{n\in\mathbb{N}} \frac{(-1)^n}{(2n)!}\left(x^2\right)^n \quad \text{et} \quad \sin(x) = x \cdot \sum_{n\in\mathbb{N}} \frac{(-1)^n}{(2n+1)!}\left(x^2\right)^n.$$

La forme choisie pour le sinus et le cosinus permet de construire les séries entières en évitant de distinguer les indices pairs et impairs et ainsi démontrer plus facilement la convergence des séries à l'aide du critère de d'Alembert.

La fonction exponentielle est utilisée pour définir les fonctions hyperboliques. Les tangentes trigonométrique et hyperbolique ont été définies par les quotients usuels.

Les fonctions logarithme et racine carrée ont été définies comme les fonctions inverses respectives de l'exponentielle et de la fonction carrée. Ces fonctions étant partielles, elles ont été construites en deux temps afin de fournir à l'utilisateur des fonctions totales : une première version est définie sur l'ensemble de définition de ces fonctions, puis elle est prolongée par 0 pour les autres nombres réels. Comme pour la fonction `Rinv`, le domaine de définition n'apparaît que dans les hypothèses des théorèmes les utilisant. On peut tout de même noter quelques exceptions : en raison de la façon de prolonger la racine carrée, cette fonction est positive pour tous les nombres réels, cela donne au final un lemme plus facile à appliquer. Cette fonction est également continue en tout point, mais le théorème de continuité prend tout de même comme hypothèse la positivité.

Concernant les fonctions inverses des fonctions trigonométriques, seule la fonction arc-tangente `atan` a été définie.

2.1.4 Automatisation de la différentiabilité

La bibliothèque standard de Coq dispose d'une tactique `reg` permettant de démontrer automatiquement la continuité, la dérivabilité et la valeur des dérivées. Pour ce dernier point, la tactique fonctionne par réécritures successives : elle cherche un terme de la forme `derive_pt f x pr`, applique le lemme correspondant à l'opérateur en tête de f, puis s'appelle récursivement sur les termes `derive_pt` créés par le lemme. On remarque que, pour pouvoir écrire une expression contenant `derive_pt f x pr` sur laquelle appliquer la tactique, l'utilisateur doit fournir le terme de preuve `pr` que la tactique `reg` peut construire par un procédé similaire.

Un avantage de la tactique `reg` est son exhaustivité : elle gère toutes les fonctions pour lesquelles la bibliothèque standard connaît des propriétés de dérivabilité, par exemple les fonctions trigonométriques. Malheureusement, elle ne traite pas la dérivation de primitives et d'intégrales ou de fonctions définies par l'utilisateur.

2.2 Autres bibliothèques et prouveurs

Il existe d'autres formalisations d'analyse standard. Je présente dans cette section quelques-uns des assistants de preuve disposant d'un développement d'analyse dans leur bibliothèque standard ainsi que quelques bibliothèques ayant la même légitimité.

Ces assistants de preuves peuvent être classés suivant plusieurs caractéristiques. Tout d'abord, la saisie peut être faite de façon déclarative ou procédurale [Har96]. Dans un système déclaratif, par exemple Mizar, l'utilisateur indique explicitement les états intermédiaires, tandis que les étapes de preuves sont implicites. Au contraire, le style procédural, par exemple en PVS, utilise des langages basés sur des tactiques, ce qui rend explicites les étapes de la démonstration et rend implicites les états intermédiaires.

Certains assistants de preuve suivent l'approche LCF [Pau87] : ils s'appuient sur un petit noyau écrit dans un langage du style ML, où seules les règles d'inférence de base sont autorisées, et l'utilisateur peut écrire les tactiques lui permettant de démontrer les théorèmes. Les autres approches seront détaillées au fil des besoins.

Cette section est consacrée à la présentation des différents systèmes et bibliothèques étudiés dans cet état de l'art. Cette présentation commence par Mizar dans la section 2.2.1, ACL2(r) dans la section 2.2.2 et PVS dans la section 2.2.3. Les prouveurs de la famille HOL sont présentés ensuite avec HOL Light dans la section 2.2.4, HOL4 dans la section 2.2.5, ProofPower-HOL dans la section 2.2.6 et enfin Isabelle/HOL dans la section 2.2.7. Ce panorama se termine avec la bibliothèque C-CoRN pour Coq dans la section 2.2.8.

2.2.1 Mizar

Mizar [3] est un système formel généraliste basé sur la logique classique et le système de déduction naturelle de Jaskowski [Try93, NK09]. Une preuve est un script Mizar, qui est vérifié par le système.

L'objectif principal est que les preuves soient proches du langage naturel des mathématiques, de sorte que les mathématiciens puissent facilement l'utiliser. Par conséquence, le langage est purement déclaratif. Cela rend les preuves plus longues, mais également plus faciles à lire. La bibliothèque mathématique de Mizar est conséquente : 9 400 définitions de concepts mathématiques et plus de 49 000 théorèmes ont été accumulés depuis 1989. Mizar est également utilisé à des fins pédagogiques [RZ04].

La logique est fondée sur les axiomes de la théorie des ensembles de Tarski-Grothendieck (une extension de la théorie des ensembles de Zermelo-Fraenkel). Tous les objets sont des ensembles, certains étant appelés entiers ou réels. L'inconvénient est que Mizar est un système figé : il ne peut pas être étendu ou programmé par l'utilisateur ; il n'a pas de capacité de calcul ni d'automatisation. La définition des nombres réels est décrite dans la section 2.3.3.

2.2.2 ACL2(r)

ACL2 [4] est un système basé sur une logique du premier ordre avec une règle d'inférence pour l'induction [KMM00]. Parmi les systèmes considérés ici, ACL2 est le seul utilisant la logique du premier ordre. Il est largement utilisé dans l'industrie [MLK98] car il est robuste et dispose de bonnes heuristiques de preuve.

3. `http://mizar.org/`
4. `http://www.cs.utexas.edu/users/moore/acl2`

ACL2 est écrit en Common Lisp. L'utilisateur soumet uniquement les définitions et les théorèmes, et le système essaye de déduire les preuves. En cas d'échec de la preuve, l'utilisateur peut alors ajouter des lemmes tels que des règles de réécriture, ou donner des indications comme la désactivation d'une règle ou l'instanciation d'un théorème. Les quantificateurs ne sont pas directement supportés, mais ACL2 offre différentes façons de les simuler, par exemple par skolemisation. Après avoir développé une théorie en ACL2, l'utilisateur peut l'exécuter : par exemple après avoir formalisé un microprocesseur, il est possible d'en simuler le fonctionnement.

ACL2(r) est une extension de ACL2 qui offre un support permettant de raisonner avec les nombres irrationnels et complexes. Il modifie la logique de ACL2 en introduisant des notions d'analyse non standard [GK01]. Ses nombres réels sont décrits dans la section 2.3.4.

2.2.3 PVS et la NASA Library

PVS [5] est un système formel basé sur la logique classique d'ordre supérieur [ORS92]. Contrairement à l'approche LCF, PVS n'a pas un petit noyau mais un grand système monolithique contenant le vérificateur et de nombreuses méthodes d'automatisation. Il utilise largement les prédicats, sous-types et types dépendants [ROS98]. Les TCCs (type-correctness conditions) sont des obligations de preuve générées par le typechecker de PVS, par exemple pour empêcher les divisions par zéro.

PVS est principalement écrit en Common Lisp et l'interface utilisateur repose sur Emacs. Il y a un fichier pour les spécifications écrit par l'utilisateur. Les preuves sont faites de façon interactive et regroupées dans un fichier distinct qui décrit un arbre de preuve sur le modèle du calcul des séquents. Il y a aussi une interface appelée ProofLite permettant de réaliser des scripts de preuves à l'image des autres systèmes. L'ergonomie est facilitée grâce à des procédures de décision efficaces et une utilisation habile des informations de type pour les preuves. La formalisation des nombres réels est décrite à la section 2.3.1. Une vaste bibliothèque, maintenue par la NASA [6], contient une formalisation d'analyse réelle.

2.2.4 HOL Light

HOL Light [7] est un système formel basé sur la logique classique d'ordre supérieur avec les axiomes de l'infini, d'extensionnalité et du choix sous la forme de l'opérateur ε de Hilbert [Har09]. Il suit l'approche LCF avec un petit noyau écrit en OCaml [Har06]. Les règles d'inférence de base peuvent être composées et des procédés d'automatisation sont disponibles. L'utilisateur peut également programmer ses propres méthodes d'automatisation. HOL Light a été presque entièrement écrit par J. Harrison. Cependant, il s'appuie sur les versions antérieures de HOL, notamment sur le travail original [GM93] et l'amélioration de l'implémentation par Slind [SN08].

Sa formalisation de l'analyse réelle était au départ celle que J. Harrison a conçu pour HOL98 [Har98] pendant sa thèse. La bibliothèque d'analyse réelle de HOL Light a été généralisée en une formalisation aux espaces euclidiens $\mathbb{R}^n$ tout en gardant la même construction des nombres réels [Har13]. Je ne parlerai que de cette dernière formalisation. Les détails sur la formalisation antérieure peuvent être déduits de la description de la bibliothèque d'analyse réelle de HOL4 car il n'y a pas eu beaucoup de modifications. Les nombres réels d'HOL Light sont décrits dans la section 2.3.2.

2.2.5 HOL4

HOL4 [8] est la quatrième version de HOL. Il suit l'approche LCF avec un petit noyau écrit en SML. Il s'appuie sur HOL98, mais aussi sur des idées et outils de HOL Light. La logique est également la même que celle des autres prouveurs de la famille HOL [HOL12]. Les programmes externes, comme les solveurs SMT ou par BDD, peuvent être appelés en utilisant un système d'oracle. Ses nombres réels sont décrits dans la section 2.3.3.

2.2.6 ProofPower-HOL

ProofPower-HOL [9] est un autre héritier du système HOL. Il est écrit en SML et partage la même logique que HOL Light et HOL4. Un point distinctif est que ProofPower-HOL fournit en plus un support

5. `http://pvs.csl.sri.com/`
6. `http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library/`
7. `http://www.cl.cam.ac.uk/users/jrh/hol-light/index.html`
8. `http://hol.sourceforge.net/`
9. `http://www.lemma-one.com/ProofPower/index/`

pour les spécifications et les preuves en Z en utilisant une intégration sémantique de Z dans HOL [AK96]. Il contient également un outil permettant de transformer des spécifications écrites en Z en programmes dans le langage SPARK.

La bibliothèque d'analyse réelle a été développée depuis 2001 afin de tester la formalisation des nombres réels, décrits dans la section 2.3.3. Maintenant, la bibliothèque d'analyse réelle est utilisée pour démontrer les théorèmes d'analyse de la liste de 100 théorèmes célèbres de la page internet de F. Wiedijk [10].

2.2.7 Isabelle/HOL

Isabelle/HOL [11] est un système de preuve formelle basé sur la logique d'ordre supérieur [NPW02]. Isar est une extension spécifique pour le style déclaratif [Wen02, WW02]. Isabelle utilise plusieurs outils génériques de preuve (réécriture à l'ordre supérieur, recherche de preuve classique, arithmétique, etc) et propose de nombreux types de données (ensembles et types de données inductifs, enregistrements extensibles, etc). Comme les autres prouveurs dans le style LCF, toutes les preuves découlent d'inférences primitives dans le noyau. Les développements de théories et de preuves sont interactifs, à la fois pour les scripts de preuve non structurés et pour les textes de preuve structurés en Isar. Ses nombres réels sont décrits dans la section 2.3.2.

2.2.8 La bibliothèque C-CoRN/MathClasses

Comme évoqué au début de la section 2.1, une bibliothèque d'analyse constructive a été formalisée en Coq. Elle a été développée à Nimègue aux Pays-Bas. La formalisation des nombres réels constructifs est appelée C-CoRN [CFGW04] (pour the Constructive Coq Repository at Nijmegen) et la bibliothèque d'analyse réelle conçue au dessus d'une axiomatisation de ces nombres réels s'appelle MathClasses [SvdW11, KS13]. L'idée principale de ces bibliothèques est la formalisation des mathématiques constructives à la Bishop [Bis67]. Il a été démontré que les nombres réels de la bibliothèque standard de Coq vérifient les axiomes de la bibliothèque MathClasses [12]. Les bibliothèques C-CoRN et MathClasses sont décrites dans la section 2.3.2.

2.3 Les formalisations des nombres réels

Avant même de s'attaquer à la question de l'analyse réelle, il faut comprendre comment les nombres réels sont représentés dans les différents systèmes et bibliothèques. À cause des systèmes logiques ou de choix techniques, toutes ces formalisations sont très différentes. L'organisation de cette section n'est donc pas par prouveur mais par type de définition. Je présente tout d'abord l'axiomatisation des nombres réels de PVS dans la section 2.3.1. J'aborde ensuite les nombres réels construits à partir de suites de Cauchy dans la section 2.3.2 et à partir de coupures de Dedekind dans la section 2.3.3. Pour finir, je présente les nombres réels de ACL2(r) définis à partir de réels non standards dans la section 2.3.4.

Il est à noter que Isabelle/HOL dispose également d'une bibliothèque de réels non standards, mais elle est définie à parti des nombres réels. L'objectif de cette bibliothèque n'est donc pas de faire de l'analyse réelle, mais bien de faire de l'analyse non standard et ne sera donc pas présentée ici.

2.3.1 Les nombrées réels axiomatisés de PVS

Dans PVS, comme dans la bibliothèque standard de Coq, les nombres réels sont définis comme un corps totalement ordonné, archimédien et complet. Cela conduit à des formalisations courtes et intuitives, mais il y a toujours un léger doute quant à savoir si les axiomes supplémentaires entraînent de l'incohérence avec les axiomes natifs de ces systèmes. Les définitions de ces deux systèmes sont au final assez proches, mais les mécanismes de PVS permettent de simplifier un certain nombre de démonstrations.

Grâce au système de sous-typage, PVS adopte une approche du plus au moins complexe. Au lieu de partir des entiers naturels et de construire au-dessus des structures de plus en plus complexes (comme cela se fait usuellement), PVS part d'un type `number` qui est un sur-ensemble de tous les nombres [OS03]. Cet ensemble comprend les nombres entiers, rationnels et réels. On peut remarquer que les nombres entiers ne sont pas formalisés dans PVS : il s'agit des entiers natifs du système Lisp.

10. `http://www.cs.ru.nl/F.Wiedijk/100/index.html`
11. `http://isabelle.in.tum.de/`
12. `https://github.com/robbertkrebbers/corn/tree/master/coq_reals`

PVS définit alors un sous-type `number_field` de `number` qui possède les opérations de corps et axiomatise ses propriétés. Les nombres réels, complexes et hyper-réels, ... sont tous définis comme des sous-types de `number_field`. Voici une sélection de quelques déclarations de la théorie qui commence cette hiérarchie :

```
number: NONEMPTY_TYPE
number_field: NONEMPTY_TYPE FROM number
nonzero_number: NONEMPTY_TYPE = {r: number_field | r /= 0} CONTAINING 1
/: [number_field, nonzero_number -> number_field]
n0x: VAR nonzero_number
inverse_mult: AXIOM n0x * (1/n0x) = 1
```

Une chose remarquable est que la division est une opération entre un nombre et un nombre non nul. Cela signifie que, chaque fois que l'utilisateur écrit une division, PVS ajoute une condition préalable implicite à la formule qui exige une preuve que le dénominateur n'est pas nul.

Pour les axiomes de `number_field` concernant les corps simples, neuf d'entre eux donnent les propriétés de l'addition, de la multiplication et de leurs inverses. Les deux autres définissent la soustraction et la division à partir de l'opposé et de l'inverse. Un point à noter est qu'il n'y a pas d'axiome établissant que $0 \neq 1$. En effet, cette propriété est gratuite car les entiers de PVS sont hérités de ceux de Lisp.

Les nombres réels sont alors définis comme un sous-type de `number_field` qui est clos pour les opérations de corps et qui dispose d'un ordre total compatible avec celles-ci. Voilà un extrait de cette théorie :

```
real: NONEMPTY_TYPE FROM number_field
nonzero_real: NONEMPTY_TYPE = {r: real | r /= 0} CONTAINING 1
nzreal_is_nznum: JUDGEMENT nonzero_real SUBTYPE_OF nonzero_number
x, y: VAR real
n0z: VAR nonzero_real
closed_divides: AXIOM real_pred(x / n0z)
<(x, y):  bool
posreal_add_closed: POSTULATE x > 0 AND y > 0 IMPLIES x + y > 0
trichotomy: POSTULATE x > 0 OR x = 0 OR 0 > x
```

On peut remarquer le mot clé `POSTULATE` qui signifie que ces propriétés, alors qu'elles sont théoriquement des axiomes, peuvent être démontrées grâce aux procédures de décision de PVS. Cela signifie que, en raison de leur implémentation, les procédures de décision viennent avec leur propre ensemble d'axiomes concernant les nombres réels. Cela est similaire au cas de $0 \neq 1$ décrit ci-dessus. En conséquence, c'est la combinaison des axiomes explicites de la théorie et des axiomes implicites des procédures de décision de PVS qui définit les nombres réels de PVS.

Maintenant que les nombres réels sont un corps totalement ordonné, la complétude de $\mathbb{R}$ est établie par l'existence de la borne supérieure de tout ensemble non vide et borné de réels [Dut96].

```
S: VAR (nonempty?[real])
upper_bound?(x, S): bool = FORALL (s: (S)): s <= x
least_upper_bound?(x, S): bool = upper_bound?(x, S) AND
  FORALL y: upper_bound?(y, S) IMPLIES (x <= y)
real_complete: AXIOM FORALL S:
  (EXISTS y: upper_bound?(y, S)) IMPLIES
  (EXISTS y: least_upper_bound?(y, S))
```

Pour finir, les nombres rationnels de PVS sont définis comme un sous-type des réels, les entiers comme un sous-type des rationnels et ainsi de suite. Comme pour $0 \neq 1$, le fait que les constantes littérales 0, 1, 2, etc, sont des éléments de tous ces types et différents les uns des autres est codé en dur dans le système. On peut remarquer que l'approche du haut vers le bas ne s'oppose pas à l'ajout de sur-ensembles au dessus des nombres réels (par exemple les nombres complexes). Bien que les réels soient déclarés comme un sous-ensemble de `number_field`, des jugements de sous-typage peuvent être ajoutés afin d'en faire un sous-ensemble d'autres ensembles.

En plus de l'axiomatisation de $\mathbb{R}$ présentée ci-dessus, PVS contient également une bibliothèque de nombres réels calculables développée par D. Lester [Les08]. Ce développement ne disposant pas de bibliothèque d'analyse réelle à proprement parler (seules quelques fonctions usuelles ont été définies), je vais juste donner la définition de ces nombres réels calculables. Ils sont basés sur des suites d'entiers (c_p) permettant de représenter chaque élément de x `real` de telle sorte que $c_p 2^{-p}$ est proche de x à 2^{-p} près.

```
x: VAR real
```

```
p: VAR nat
c: VAR [nat->int]
cauchy_prop(x, c): bool  =
  (FORALL p: c(p)-1 < x*2^p AND x*2^p < c(p)+1)
```

`cauchy_real` est alors le type des suites `c` telles qu'il existe un réel `x` vérifiant `cauchy_prop(x, c)`. Chaque opération (addition, multiplication, ...) et fonction de base (exponentielle, sinus, logarithme népérien, ...) est définie et reliée à sa version dans les réels. Pour la multiplication, cela donne le théorème suivant :

```
x,y : VAR real
cx, cy: VAR cauchy_real
mul_lemma: LEMMA cauchy_prop(x,cx) AND cauchy_prop(y,cy)
                 => cauchy_prop(x*y, cauchy_mul(cx,cy))
```

2.3.2 Les suites de Cauchy

Cette section détaille quelques formalisations de réels construits à partir de suites de Cauchy. Un point important est que ces formalisations, ainsi que celles basées sur les coupures de Dedekind, n'introduisent pas de nouveaux axiomes. Au lieu de ça, un ensemble de nombres est construit, puis s'avère avoir les propriétés des nombres réels. Ces constructions dépendant fortement du système logique sous-jacent, il n'est pas toujours possible d'exporter une construction d'un système vers un autre.

HOL Light

Cette formalisation, originaire de HOL98, a ensuite servi de base à la bibliothèque standard de HOL Light [Har98]. Plutôt que d'utiliser directement des suites de rationnels, elle est basée sur des suites quasi-additives d'entiers naturels. Le prédicat `is_nadd` donné ci-dessous caractérise une telle suite `x`, où `dist` est la fonction retournant la distance entre deux entiers naturels. Remarquons que dans les langages de type HOL, `?` est le quantificateur existentiel et `!` est le quantificateur universel.

```
let is_nadd = new_definition
  `is_nadd x <=> (?B. !m n. dist(m*x(n), n*x(m)) <= B * (m + n))`;;
```

Cette propriété correspond au critère de Cauchy pour la suite x_n/n, et donc de telles suites convergent et la limite ℓ vérifie $\forall n,\ |x_n/n - \ell| < B/n$. Les nombres réels positifs sont alors construits comme des classes d'équivalence sur les suites quasi-additives par la relation d'équivalence `===` suivante :

```
let nadd_eq = new_definition
  `x === y <=> ?B. !n. dist(x(n), y(n)) <= B`;;
```

Grâce à cette relation d'équivalence, l'ensemble des nombres réels est défini comme un type quotient `hreal`. Pour finir, l'ensemble des nombres réels est construit à partir de l'ensemble des paires de `hreal` quotienté par la relation d'équivalence suivante :

```
let treal_eq = new_definition
  `(x1,y1) treal_eq (x2,y2) <=> (x1 + y2 = x2 + y1)`;;
```

Les différentes propriétés des nombres réels sont d'abord démontrées sur les suites quasi-additives, puis il est démontré qu'elles sont stables par passage au quotient.

L'ordre `<<=` est défini de la façon suivante :

```
let nadd_le = new_definition
  `x <<= y <=> ?B. !n. x(n) <= y(n) + B`;;
```

L'existence de la borne supérieure d'un ensemble non vide et borné de suites quasi-additives est ensuite démontré. Ce théorème est étendu pour démontrer la complétude de l'ensemble des nombres réels.

L'addition de deux suites quasi-additive est effectuée point par point tandis que la multiplication correspond à la composition de suites ($x_{y_n}/n = (x_{y_n}/y_n) \cdot (y_n/n)$). La commutativité et l'associativité de ces opérations sont démontrées, et on peut construire un inverse pour la multiplication possédant les propriétés attendues.

Isabelle/HOL

Bien que la bibliothèque d'analyse réelle de Isabelle/HOL soit largement inspirée de HOL Light, ce n'est pas le cas pour la construction des nombres réels. En effet, cette formalisation repose sur des suites de nombres rationnels plutôt que sur des suites quasi-additives d'entiers naturels. Remarquons que ce n'est pas la première formalisation en Isabelle/HOL : les réels y étaient auparavant construits à partir de coupures de Dedekind [Fle00], mais il n'y en a plus de trace dans la bibliothèque d'analyse présentée ici. Les définitions ci-dessous définissent les suites de Cauchy, ainsi que la notion de convergence vers 0 :

```
definition cauchy :: "(nat => rat) => bool"
  where "cauchy X <-> (∀r>0. ∃k. ∀m≥k. ∀n≥k. |X m - X n| < r)"
definition vanishes :: "(nat => rat) => bool"
  where "vanishes X = (∀r>0. ∃k. ∀n≥k. |X n| < r)"
```

Une relation d'équivalence entre suites de Cauchy est ensuite définie comme le fait que leur différence tende vers 0. Cette relation est utilisée pour construire un type quotient `real`.

```
definition realrel :: "((nat => rat) × (nat => rat)) set"
  where "realrel = {(X, Y).
    cauchy X ∧ cauchy Y ∧ vanishes (λn. X n - Y n)}"
lemma equiv_realrel: "equiv {X. cauchy X} realrel"
typedef real = "{X. cauchy X} // realrel"
```

De plus, comme la définition du quotient en Isabelle/HOL est

```
definition quotient :: "'a set => ('a × 'a) set => 'a set set"
  where "A//r =
    (\<Union>x ∈ A. {r``{x}})"  -- {* set of equiv classes *}
```

avec `{r``{x}}` l'ensemble des éléments en relation avec `x`, un nombre réel est par définition la classe d'équivalence d'une suite de Cauchy donnée.

Comme cela a été vu pour HOL Light, les opérations arithmétiques sont définies sur les suites et passées au type quotient. Il est ensuite démontré que cet ensemble est un corps archimédien possédant la propriété de la borne supérieure.

C-CoRN/MathClasses

Les bibliothèques C-CoRN et MathClasses de Coq ont une approche bien différente des autres formalisations présentées ici. En effet, l'analyse réelle y est formalisée dans le cadre des mathématiques constructives plutôt que dans celui de la logique classique. De ce fait, ces bibliothèques s'appuient sur la logique intuitionniste de Coq. Combiné avec le mécanisme d'extraction de ce système, cela signifie qu'à partir d'une preuve de la propriété $\forall x \exists y\ R(x, y)$ pour une relation R, on obtient une fonction f telle que $\forall x\ R(x, f(x))$. Cela n'est pas une conséquence de l'axiome du choix. En effet, la fonction f peut calculer de façon effective un résultat pour chaque entrée. La rapidité de ce calcul dépendant de la façon dont la propriété a été démontrée.

De nombreuses difficultés découlent de l'usage des mathématiques constructives et cela se reflète dans les développements Coq. En effet, les preuves nécessitent une discipline beaucoup plus stricte et peuvent être plus compliquées car les astuces usuelles n'ont plus cours. Par exemple, le tiers-exclu, l'axiome du choix, la décidabilité de l'égalité, ainsi que d'autres propriétés usuelles ne sont plus disponibles. Certaines notions d'analyse sont également reformulées ou tout simplement inexistantes, comme par exemple les fonctions discontinues.

Comme pour les formalisations précédentes, les nombres réels de C-CoRN sont construits à partir des suites de Cauchy [GN02]. Cependant, contrairement aux systèmes de type HOL, Coq offre peu de support pour les types quotient. En conséquence la formalisation de C-CoRN est basée sur la notion de *sétoïde* qui est un ensemble de base muni d'une relation entre ses éléments et de quelques preuves que la relation est une relation d'équivalence.

Une autre caractéristique de cette bibliothèque est que l'analyse réelle n'est pas construite directement sur le sétoïde des nombres réels construits à partir des suites de Cauchy, mais plutôt sur le type abstrait `CReals` ci-dessous (un peu simplifié) :

```
Record CReals : Type := {
  carrier :> COrdField;
  limit : CauchySeq carrier -> carrier;
  ax_Lim : forall s : CauchySeq carrier, SeqLimit s (limit s);
  ax_Arch : forall x : carrier, {n : N | x <= nring n} }.
```

Le premier champ indique que le type utilisé comme base pour les nombres réels constructifs est un corps ordonné. Les deuxième et troisième champs indiquent qu'il existe une fonction des suites de Cauchy sur ce corps dans le corps, et que cette fonction calcule la limite. Le dernier champ indique que le corps est archimédien. La formalisation est alors complétée en postulant l'existence d'un ensemble `IR` vérifiant ces propriétés :

```
Axiom IR : CReals.
```

Un développement séparé explique comment construire un espace métrique complet à partir d'un espace métrique, ce qui permet de construire une instance de `IR` pour l'extraction. Ceci se fait en utilisant des fonctions "régulières" : étant donné un nombre rationnel positif, elles retournent un élément de l'espace métrique qui est suffisamment proche de l'élément qu'elles sont censées représenter. Comme on peut le voir ci-dessous, cette propriété est liée au critère de convergence de Cauchy.

```
Definition is_RegularFunction (x:QposInf -> X) : Prop :=
  forall (e1 e2:Qpos), ball (m:=X) (e1+e2) (x e1) (x e2).
```

Il est démontré que le sétoïde `Q` des nombres rationnels construit dans C-CoRN est un espace métrique qui est ensuite complété. Pour finir, le complété est prouvé isomorphe à l'ensemble axiomatisé `IR`.

La bibliothèque MathClasses est basée sur la monade de complétion $\mathfrak{C}$ définie dans [O'C07]. C'est une monade sur la catégorie des espaces métriques et des fonctions uniformément continues entre eux. $\mathbb{R}$ est alors défini comme étant $\mathfrak{C}(\mathbb{Q})$. Un nombre réel x (vu comme une fonction régulière) est dit positif si

$$\forall \varepsilon \in \mathbb{Q}^+, \quad -\varepsilon \leq_{\mathbb{Q}} x(\varepsilon).$$

L'ordre $x \leq_{\mathbb{R}} y$ est alors défini par "$y - x$ est positif". Les fonctions transcendantes peuvent d'abord être définies de $\mathbb{Q}$ dans $\mathbb{R}$, puis transformées en fonctions de $\mathbb{R}$ dans $\mathbb{R}$ par continuité [O'C08]. Des techniques ingénieuses de compression et de réduction d'arguments sont utilisées pour calculer plus efficacement.

Les interfaces abstraites sont largement utilisées pour faciliter les déclarations et les preuves [KS13]. Grâce aux classes de types [CS12], la hiérarchie algébrique (sétoïde, groupe, anneau, ...) peut être facilement utilisée. En effet, les lemmes définis pour une classe de types peuvent être utilisés par les classes de types en héritant. Par exemple, un lemme démontré sur les groupes peut être utilisé pour démontrer des propriétés sur les anneaux.

Pour finir, la bibliothèque C-CoRN et la bibliothèque standard de Coq ne sont pas sans lien : l'équivalence entre ces deux bibliothèques a été démontrée en utilisant les axiomes de la bibliothèque standard [KO09].

2.3.3 Les coupures de Dedekind

Les coupures de Dedekind sont une autre façon de compléter l'ensemble des nombres rationnels $\mathbb{Q}$, et donc de construire l'ensemble des réels $\mathbb{R}$. Un sous-ensemble A d'un ensemble ordonné totalement ordonné X est une coupure de Dedekind si :

- A est non vide, *i.e.* $\exists x,\ x \in A$,
- A est majoré, *i.e.* $\exists M \in X,\ \forall x \in A,\ x < M$,
- A est clos par le bas, *i.e.* $\forall x \in A,\ \forall y < x,\ y \in A$,
- A n'a pas de plus grand élément, *i.e.* $\forall x \in A,\ \exists y \in A,\ x < y$.

Cette façon de définir les nombres réels est utilisée, avec quelques variantes, dans Mizar, HOL4 et ProofPower-HOL.

Mizar

Alors qu'à l'origine les nombres réels de Mizar étaient axiomatiques, la formalisation a été changée pour une construction basée sur les coupures de Dedekind. Comme Mizar est basé sur la théorie des ensembles, cette formalisation s'intègre facilement dans ce système. La hiérarchie commence en définissant les entiers naturels et les rationnels positifs `RAT+`. Les coupures de Dedekind sont ensuite construites en utilisant la définition usuelle :

```
func DEDEKIND_CUTS -> Subset-Family of RAT+ equals
  { A where A is Subset of RAT+ :
    for r being Element of RAT+ st r in A holds
    ( ( for s being Element of RAT+ st s <=' r holds s in A ) &
      ex s being Element of RAT+ st ( s in A & r < s ) )
  } \ {RAT+};
```

ce qui se traduit par A est une coupure de Dedekind si c'est un sous-ensemble strict de $\mathbb{Q}^+$ tel que

$$\forall r \in A,\ (\forall s \in \mathbb{Q}^+,\ s \leq r \Rightarrow s \in A) \wedge (\exists s \in \mathbb{Q}^+,\ r < s \wedge s \in A).$$

Bien que l'ensemble vide soit généralement exclu des coupures de Dedekind, ce n'est pas le cas dans Mizar. En effet, les coupures servent usuellement à représenter les réels strictement positifs, alors qu'ici ces coupures visent à représenter les réels positifs ou nuls.

Les nombres réels positifs sont alors définis en prenant l'union des nombres rationnels positifs et des coupures de Dedekind précédemment définies. Par conséquent, l'injection des nombres rationnels positifs dans les réels positifs est la fonction identité. L'ensemble des coupures de Dedekind ainsi construites représente aussi bien les nombres réels irrationnels que les nombres rationnels. Afin d'éviter de dupliquer les nombres rationnels dans ce nouvel ensemble, les coupures correspondant à ces nombres ont été exclues de cette union :

```
func REAL+ -> set equals (RAT+ \/ DEDEKIND_CUTS)
  \ { { s where s is Element of RAT+ : s < t }
      where t is Element of RAT+ : t <> 0 };
```

Pour finir, les nombres réels sont définis en dupliquant les nombres réels positifs afin de construire les nombres négatifs et en supprimant la nouvelle version de zéro afin d'assurer l'unicité de la représentation. Dans la définition ci-dessous, `[:{0},REAL+:]` doit être compris comme l'ensemble des paires $(0, x)$ avec x un réel positif, et `[0,0]` représente la paire $(0, 0)$.

```
func REAL -> set equals (REAL+ \/ [:{0},REAL+:]) \ {[0,0]};
```

Puisque les nombres entiers et rationnels négatifs ont été construits de la même façon (une paire avec le premier élément égal à zéro), les nombres entiers et rationnels forment des sous-ensembles des nombres réels.

Les opérations arithmétiques et leurs propriétés sont d'abord définies et démontrées sur les coupures de Dedekind, puis étendues aux réels positifs puis à l'ensemble des réels. Par exemple, la commande ci-dessous définit l'addition sur les coupures de Dedekind :

```
func A + B -> Element of DEDEKIND_CUTS equals
  { (r + s) where r, s is Element of RAT+ : ( r in A & s in B ) };
```

Pour finir, la complétude a été démontrée sous différentes formes. Ci-dessous, la propriété de la borne supérieure est exprimée sous la forme d'une fonction. Étant donné un ensemble non vide et majoré, cette fonction retourne la borne supérieure (notée par le mot clé `it` dans la définition).

```
let X be real-membered set;
assume A1: ( not X is empty & X is bounded_above );
func upper_bound X -> real number means
  ( ( for r being real number st r in X holds r <= it ) &
    ( for s being real number st 0 < s holds
      ex r being real number st ( r in X & it - s < r ) ) );
```

HOL4

Comme dans Mizar, les nombres réels de HOL4 sont définis à partir des coupures de Dedekind [Har94] :

```
val isacut = new_definition("isacut",
--‘isacut C =
 (?x. C x)                              /\      (* Non vide             *)
 (?x. ~C x)                             /\      (* Majoré                 *)
 (!x y. C x /\ y hrat_lt x ==> C y) /\      (* Clos par le bas *)
 (!x. C x ==> ?y. C y /\ x hrat_lt y)‘--); (* Pas de plus grand élément *)
```

avec `hrat_lt` l'ordre strict sur les nombres rationnels positifs. Dans cette définition, une coupure `C` est un sous-ensemble de $\mathbb{Q}_+^* \equiv \{(x+1, y+1) \mid x, y \in \mathbb{N}\}$.

Pour définir les opérations et démontrer les théorèmes, deux fonctions, `hreal`, des sous-ensembles de $\mathbb{Q}^+$ vers $\mathbb{R}^+$, et `cut`, de $\mathbb{R}^+$ vers les coupures de Dedekind, ont été définies par

```
val hreal_tybij =
 define_new_type_bijections
   {name="hreal_tybij",ABS="hreal",REP="cut",tyax=hreal_tydef};
```

où `define_new_type_bijections` génère automatiquement des fonctions `hreal` et `cut` telles que

$$(\forall a,\ \texttt{hreal}(\texttt{cut}(a)) = a) \wedge (\forall r,\ \texttt{isacut}(r) \Leftrightarrow (\texttt{cut}(\texttt{hreal}(r)) = r)).$$

En dehors de l'inverse multiplicatif, les opérations sur les nombres réels ont été définies de la même façon qu'en Mizar :

$$\begin{aligned} \sup S &= \bigcup S \\ X + Y &= \{x + y \mid x \in X \wedge y \in Y\} \\ XY &= \{xy \mid x \in X \wedge y \in Y\} \\ X^{-1} &= \{w \mid \exists d < 1,\ \forall x \in X,\ wx < d\} \end{aligned}$$

Pour finir, comme dans HOL Light, les nombres réels ont été définis comme le quotient des paires de nombres réels par la relation d'équivalence suivante :

$$\forall x_1, y_1, x_2, y_2,\ ((x_1, y_1) \approx (x_2, y_2)) \Leftrightarrow (x_1 + y_2 = x_2 + y_1)$$

ProofPower-HOL

Comme Mizar et HOL4, les nombres réels de ProofPower-HOL sont définis en utilisant les coupures de Dedekind. La définition des coupures de Dedekind ci-dessous utilise la syntaxe de ProofPower : • est un séparateur entre les quantificateurs et les formules, \$≪ est une relation arbitraire (sur laquelle la formule est universellement quantifiée), et ≪ est la notation infixe pour \$≪.

$$\begin{aligned} \forall X\ \${\ll}\ A \bullet A \in \mathit{Cuts}(X, \${\ll}) \quad \Leftrightarrow \quad & A \subseteq X \wedge \neg A = \{\} \wedge \mathit{UnboundedAbove}(A, \${\ll}) \\ & \wedge (\exists x \bullet x \in X \wedge \mathit{UpperBound}(A, \${\ll}, x) \\ & \wedge (\forall a\ b \bullet a \in A \wedge b \in X \wedge b \ll a \Rightarrow b \in A)). \end{aligned}$$

À la différence des systèmes précédents, les coupures de Dedekind définies ici sont des sous-ensembles des nombres dyadiques $\mathbb{D} = \{(2m+1)/2^n \mid m \in \mathbb{N} \text{ and } n \in \mathbb{Z}\}$. Le manuel de référence [Pro06] ne décrit que la multiplication et les puissances entières sur cet ensemble dans la théorie `DYADIC`.

Malheureusement, la construction des nombres réels négatifs et des opérations n'est pas décrite. De plus, une seconde formalisation des nombres réels basée sur les coupures de Dedekind dans $\mathbb{Z}\left[\sqrt{2}\right]$ est décrite dans l'article [Art01], mais elle n'est pas disponible.

2.3.4 Les réels non standard de ACL2(r)

L'analyse non standard introduit les notions de prédicats et de fonctions *standard* et *non-standard*. Les termes *interne* et *classique* peuvent également être utilisés à la place de standard. Ces notions ne font pas référence à la logique classique : standard signifie que les formules utilisent uniquement des outils venant de l'analyse standard. En particulier, il est interdit de tester si un nombre est standard, ou d'extraire sa partie standard dans une formule standard. Le principal théorème de l'analyse non standard est le principe de *transfert* : tout prédicat standard, avec uniquement des constantes standards, qui est vrai pour tous les nombres standards est également vrai pour tous les nombres, standard ou non.

ACL2 étant basé sur la logique du premier ordre, il n'est pas adapté pour faire de l'analyse réelle avec les formules usuelles pour les limites de la forme "$\forall \varepsilon > 0,\ \exists \delta > 0,\ \ldots$". Pour contourner cette difficulté, plutôt que de formaliser l'analyse standard, ACL2 a été étendu pour supporter l'analyse non standard et ainsi éviter les quantificateurs [GK01].

Le système modifié ACL2(r) introduit trois nouveaux symboles : le prédicat `standard-numberp` qui teste si un nombre est standard, la fonction `standard-part` qui retourne la partie standard d'un nombre, et la constante `i-large-standard` qui est un entier non standard (plus grand que tout entier standard).

Comme cette extension réutilise la bibliothèque de ACL2, tous les axiomes et les théorèmes qui étaient implicitement quantifiés sur les nombres rationnels s'appliquent désormais aux hyper-réels (*i.e.* réels standards et non standards).

Le principe d'induction sur les entiers n'est pas celui utilisé en analyse standard. En effet, en présence d'une formule non standard il faut que le prédicat soit vérifié pour tous les entiers non standard (alors que pour les entiers standard, vérifier l'hypothèse d'induction traditionnelle $P(n) \Rightarrow P(n+1)$ suffit). Cela permet de maintenir la cohérence du système, mais cela signifie aussi que le principe d'induction n'est plus utilisable pour une formule contenant des nombres irrationnels. En effet, ces nombres ne sont pas

standards d'un point de vue syntaxique. Par exemple, la constante de Neper e peut être définie comme la partie standard de $\sum_{n=0}^{N} 1/n!$ avec N égal à `i-large-standard`.

Pour éviter ce problème, ACL2(r) fournit une commande alternative pour définir des fonctions (`defun-std` au lieu de `defun`), de façon à pouvoir considérer la formule comme étant standard même si elle contient des symboles non standard. Plus précisément, étant donnée une fonction non standard qui renvoie des nombres standards pour toute entrée standard, les axiomes de l'analyse non standard permettent de s'assurer qu'il existe une unique fonction standard égale à la fonction donnée sur les entrées standards. Cette fonction standard est celle créée par `defun-std`, une fois qu'il a été prouvé que les sorties sont standards chaque fois que les entrées le sont [GCK04]. Dans le cas de la constante de Neper ci-dessus, la fonction renvoyant une partie standard, elle satisfait trivialement ces hypothèses.

La fonction standard ainsi obtenue n'est généralement pas égale à la fonction d'origine sur les entrées non standards. Par exemple, la fonction `standard-part` est égale à la fonction identité sur toutes les entrées standard, la fonction construite ci-dessous est donc l'identité, par unicité [GK01].

```
(defun-std std-pt (x) (standard-part x))
```

De la même façon, le système propose une commande pour définir les théorèmes (`defthm-std` au lieu de `defthm`) basée sur le principe de transfert : un théorème est vrai si son énoncé est standard et que l'utilisateur démontre qu'il est vrai pour toute entrée standard.

2.4 Les formalisations de l'analyse réelle

Regardons maintenant comment les différents systèmes définissent les notions usuelles de l'analyse réelle : suites, séries, continuité, dérivabilité et intégration. Dans la plupart des formalisations, celles-ci ne sont traitées que dans le cadre des suites et fonctions réelles à une seule variable. HOL4, Isabelle/HOL, HOL Light, et C-CoRN les définissent dans des cadres plus généraux tels que les espaces métriques ou topologiques. Comme cela a été expliqué dans la section 2.3.4, ACL2(r) les traite dans le cadre de l'analyse non standard.

L'impact de la logique sous-jacente est visible principalement lors de la formalisation des nombres réels. Au niveau de l'analyse réelle, les différences entre les prouveurs disparaissent et les choix de conception sont principalement guidés par le type d'analyse : constructive, standard ou non standard. Cette section est donc organisée suivant les différentes notions étudiées et non suivant les prouveurs.

2.4.1 Suites et séries

Presque toutes les formalisations définissent les suites de réels comme des fonctions des entiers naturels dans les nombres réels. Seul Mizar adopte une définition légèrement différente en raison de sa définition des fonctions provenant de la théorie des ensembles. Toutefois, la définition de la convergence diffère d'un système à un autre.

Suites de réels

Dans C-CoRN/MathClasses, Mizar [Kot90a], PVS [Dut96] et ProofPower [Art01], la convergence d'une suite (u_n) vers une limite $\ell \in \mathbb{R}$ est définie de la même façon qu'en Coq :

$$\forall \varepsilon \in \mathbb{R}_+^*,\ \exists N \in \mathbb{N},\ \forall n \in \mathbb{N},\ n \geq N \Rightarrow |u_n - \ell| < \varepsilon. \tag{2.1}$$

Notons que dans C-CoRN/MathClasses, cette notion est définie pour tout corps ordonné et pas uniquement pour les nombres réels. On remarque aussi que, dans cette bibliothèque, le nom de cette propriété est `Cauchy_prop` alors qu'elle n'est pas équivalente à la convergence de Cauchy dans les ensembles qui ne sont pas complets.

La définition (2.1) est limitée aux suites de réels, voire aux suites dans les espaces métriques. Cette notion peut cependant être définie d'une façon beaucoup plus générale pour être utilisée plus tard pour la convergence des fonctions. Par exemple, en Isabelle/HOL, la convergence des suites est définie en utilisant une notion de convergence proche de la convergence pour les espaces topologiques [HIH13] :

$$\forall V \in \mathcal{G}, u^{-1}(V) \in \mathcal{F} \tag{2.5}$$

où $\mathcal{F}$ et $\mathcal{G}$ sont des filtres propres, c'est-à-dire

$$\begin{aligned}&\mathcal{F} \neq \emptyset,\ \emptyset \notin \mathcal{F},\\ &\forall A, B,\ A \in \mathcal{F} \wedge A \subseteq B \Rightarrow B \in \mathcal{F},\ \text{et}\\ &\forall A, B \in \mathcal{F},\ A \cap B \in \mathcal{F}.\end{aligned}$$

Pour les suites, les filtres choisis sont

$$\mathcal{F} = \{A \subseteq \mathbb{N} \mid \exists N \in \mathbb{N},\ \forall n \in \mathbb{N},\ n \geq N \Rightarrow n \in A\} \text{ et } \mathcal{G} = \{P \mid \exists S \in \{\text{ouvert}\},\ x \in S \wedge S \backslash \{x\} \subseteq P\}.$$

La définition (2.5) est également utilisée pour définir la convergence des fonctions réelles en choisissant des filtres semblables à $\mathcal{G}$ (voir à la section 2.4.2).

Dans HOL4 et HOL Light, la définition de la convergence dans les espaces topologiques est formalisée en utilisant des suites généralisées :

$$\forall \mathcal{V} \in \{\text{voisinage de } \ell\},\ \exists n,\ \forall m,\ m \succeq n \Rightarrow u_m \in \mathcal{V} \tag{2.6}$$

où $\succeq$ est un ordre partiel. Pour les suites, $\mathcal{V}$ est un voisinage au sens de la topologie engendrée par l'espace métrique $(\mathbb{R}, (x, y) \mapsto |y - x|)$ et $\succeq$ est l'ordre sur les entiers naturels. Cette formalisation est complété par la fonction `lim` qui retourne la limite de la fonction, si elle existe, en utilisant l'epsilon de Hilbert.

Pour finir, en ACL2(r), une suite (u_n) converge vers $\ell \in \mathbb{R}$ si $u_N \approx \ell$ (c'est-à-dire que u_N et ℓ sont infiniment proches) pour tout entier N non standard [GK01]. La limite est la partie standard de u_N pour un entier non standard N arbitraire. Ces définitions pour la convergence et la limite montrent l'intérêt d'avoir choisi l'analyse non standard pour ACL2(r). En effet, cela permet de donner une définition presque sans quantificateurs.

Pour toutes ces formalisations, la compatibilité de la limite avec les opérations arithmétiques ($+$, $-$, $\times$, et $/$) et les relations de comparaison ($=$, $\leq$) est démontrée. Les théorèmes usuels, comme la convergence des suites de Cauchy, sont également démontrés. Alors que la notion de sous-suite est très utile en pratique pour démontrer la non-convergence d'une suite, seuls C-CoRN et Mizar [Kot90b] la formalisent.

Séries de réels

Les séries entières sont le moyen usuel de définir les fonctions transcendantes telles que les fonctions exponentielle et trigonométriques. Il est donc naturel que la plupart des formalisations contiennent une implémentation des sommes partielles, des séries et des séries entières. Seuls Mizar, HOL Light et C-CoRN/MathClasses ont développé une bibliothèque permettant de faire plus que simplement définir les fonctions élémentaires transcendantes. PVS contient également un développement assez complet d'environ 300 théorèmes à propos des séries et des séries entières. Je ne le présente pas car l'équivalence avec les définitions du développement principal n'est pas démontrée.

Dans toutes les formalisations, les séries sont définies, avec de légères variations, en utilisant la définition usuelle :

$$\sum_{n \in \mathbb{N}} u_n \text{ converge vers } \ell \text{ si la suite } n \mapsto \sum_{k=0}^{n} u_k \text{ converge vers } \ell. \tag{2.4}$$

La bibliothèque C-CoRN/MathClasses commence, comme la bibliothèque standard de Coq, par définir les sommes partielles de 0 à n, puis les utilise pour définir les sommes de n à m. À la différence de la bibliothèque standard, $\sum_{k=n}^{m} u_k$ est définie par $\sum_{k=0}^{(m+1)-1} u_k - \sum_{k=0}^{n-1} u_k$ où (u_n) est une suite à valeurs dans un groupe abélien. Cette définition, sans donner 0 dans le cas $m < n$, possède quand même de meilleures propriétés que les sommes partielles de la bibliothèque standard comme pour la relation de Chasles qui ne nécessite aucune hypothèse. La convergence est définie en utilisant la définition (2.4) ; elle réutilise donc la notion de convergence définie pour les suites.

En HOL Light, HOL4 [Har98], Isabelle/HOL et PVS [Got00], les formalisations des sommes partielles sont basées sur celle de J. Harrison :

$$\texttt{sum}(u, n, m) = \begin{cases} 0 & \text{if } m = 0\\ u_{n+m-1} + \texttt{sum}(u, n, m-1) & \text{else}\end{cases}$$

Cette définition peut être lue comme "la somme de m éléments de la suite u à partir de l'indice n". La sommes partielle $\sum_{k=n}^{m} u_k$ est alors donnée par $\texttt{sum}(u, n, m-n+1)$. Comme dans C-CoRN, la convergence des séries réutilise la convergence des suites décrite dans la section 2.4.1. Dans ces formalisations, le critère

de d'Alembert et la convergence des séries alternées ont été démontrés. Dans HOL4 et Isabelle/HOL, une fonction retournant la valeur de la série a été définie en utilisant l'opérateur epsilon de Hilbert.

Mizar [RN91b] et ProofPower [Art12a] définissent les séries en utilisant explicitement la définition (2.4) avec des sommes partielles semblables à celles de la bibliothèque standard de Coq. De nombreux théorèmes de convergence sont démontrés, comme par exemple le critère de d'Alembert et les théorèmes de convergence comparée [13].

Le produit de Cauchy a été démontré dans ces formalisations car il peut être utilisé pour démontrer la propriété usuelle sur la fonction exponentielle $\forall x, y \in \mathbb{R},\ e^{x+y} = e^x \cdot e^y$:

$$\left(\sum_{n\in\mathbb{N}} a_n\right)\left(\sum_{n\in\mathbb{N}} b_n\right) = \sum_{n\in\mathbb{N}}\left(\sum_{k=0}^{n} a_k b_{n-k}\right). \tag{2.7}$$

ACL2(r) ne formalise pas explicitement les sommes partielles et les séries [GK01]. Il utilise uniquement des suites pour définir les fonctions exponentielle et trigonométriques.

Séries entières et séries de fonctions

Les séries entières sont les fonctions $f : x \mapsto \sum_{n\in\mathbb{N}} a_n x^n$ avec (a_n) une suite à valeurs dans les nombres réels ou dans un anneau. Ces fonctions sont définies pour tout x tel que la série $\sum_{n\in\mathbb{N}} a_n x^n$ est convergente.

HOL Light propose une formalisation avancée de l'analyse complexe, par exemple le théorème donnant l'équivalence entre fonctions holomorphes et analytiques y a été démontré. Ce prouveur contient également une bibliothèque riche concernant les séries entières et les séries de fonctions.

Les autres formalisations sont beaucoup moins complètes. En effet, très peu de théorèmes sont nécessaires pour définir les fonctions exponentielle et trigonométriques. En conséquence, la majorité des prouveurs fournissent juste la définition des séries entières et quelques théorèmes à propos d'elles. C'est le cas pour HOL4 [Har94] et PVS [Got00]. Il y a quelques critères de convergence comme celui de d'Alembert. Il y a également des théorèmes à propos du rayon de convergence, mais celui-ci n'est pas défini explicitement en raison de l'absence de point $+\infty$ dans ces développements. Le théorème de différentiabilité des séries entières a été démontré dans HOL4.

Une autre façon de représenter les séries entières est de les regarder comme des séries de fonctions. Cela a été fait dans C-CoRN, Isabelle/HOL, Mizar [Per92] et ProofPower [Art12a]. En conséquence, les théorèmes usuels à propos de la continuité et de la différentiabilité pour les limites de suite ont été utilisés pour étudier les fonctions exponentielle et trigonométriques.

2.4.2 Fonctions réelles

Passons maintenant en revue la façon dont sont traitées les différentes notions d'analyse pour étudier les fonctions de $\mathbb{R}$ dans $\mathbb{R}$: les fonctions partielles, les limites, la continuité et la différentiabilité.

Fonctions partielles

Les fonctions réelles sont usuellement définies d'un sous-ensemble de $\mathbb{R}$ dans $\mathbb{R}$. Il y a deux façons dans les systèmes de preuve formelle de définir les fonctions partielles : soit le domaine apparaît dans le type de la fonction, soit la fonction est totale et le domaine de définition est précisé dans les énoncés. Par exemple, pour la fonction racine carrée, dans le premier cas l'hypothèse $x \geq 0$ dans le théorème $\forall x, (\sqrt{x})^2 = x$ peut être intégrée au type de x, ce qui peut s'exprimer par $\forall x \in \mathbb{R}^+,\ (\sqrt{x})^2 = x$, alors que dans le second cas elle doit être précisée explicitement, ce qui peut s'exprimer par $\forall x \in \mathbb{R},\ x \geq 0 \Rightarrow (\sqrt{x})^2 = x$. Le moyen de définir les fonctions est très dépendant de la logique sous-jacente.

Dans Mizar, les fonctions sont définies par leur graphe [Byl90b] : ce sont les ensembles X tels que $\forall x, y_1, y_2,\ \langle x, y_1\rangle \in X \wedge \langle x, y_2\rangle \in X \Rightarrow y_1 = y_2$. Avec cette approche, le domaine de définition d'une fonction est simplement $\{x \mid \exists y,\ \langle x, y\rangle \in X\}$ et pour tout x dans le domaine, $f(x)$ est la seule valeur telle que $\langle x, f(x)\rangle \in X$. Mizar étant basé sur la théorie des ensembles, cette approche est particulièrement naturelle.

Il n'est pas utile en PVS de faire la distinction entre fonctions totales et fonctions partielles. En effet, l'usage des TCC permet de générer automatiquement les domaines de définition comme des sous-types des réels. Dans les cas les plus simples, les conditions d'application des théorèmes sont automatiquement démontrées.

13. Par exemple, si $\sum a_n$ converge et $\forall n \in \mathbb{N},\ |b_n| \leq a_n$, alors $\sum b_n$ converge.

En C-CoRN, les fonctions doivent être compatibles avec la relation d'équivalence, de sorte qu'une fonction appliquée à deux suites de Cauchy représentant le même nombre réel donne le même résultat. Une fonction totale est donc un couple avec une fonction et une preuve qu'elle est bien définie. Les fonctions partielles sont définies sur le même principe avec en plus un domaine de définition et la preuve que ce domaine est bien défini [14].

Dans les systèmes basés sur HOL, dont ProofPower-HOL, la seconde approche est utilisée : toutes les fonctions sont totales et les hypothèses des théorèmes contiennent le domaine de définition. En effet, ces systèmes offrant peu de facilités pour le sous-typage, il est plus simple que les fonctions réelles aient toutes le même type. C'est également le cas pour certaines fonctions de la bibliothèque standard de Coq tel que l'inverse multiplicatif et la racine carrée.

En ACL2, la façon de définir des fonctions partielles se fait au cas par cas [MM03]. La situation est différente de celle des autres systèmes puisque la question du sous-typage n'existe pas : la manipulation des types est en dehors du champ d'application d'un système du premier ordre.

Limites

La notion de limite est définie en PVS et ProofPower [Art12a], comme dans la bibliothèque standard de Coq, en utilisant la définition usuelle en ε-δ. Sur un domaine D, une fonction $f : \mathbb{R} \to \mathbb{R}$ converge vers une limite ℓ en un point $x \in \overline{D}$ si

$$\forall \varepsilon \in \mathbb{R}_+^*,\ \exists \delta \in \mathbb{R}_+^*,\ \forall y \in D,\ |y - x| < \delta \Rightarrow |f(y) - \ell| < \varepsilon. \tag{2.8}$$

En Isabelle/HOL et HOL Light, la convergence pour les fonctions réelles à une variable est définie de la même façon que les limites de suites (2.5) en utilisant le filtre des voisinages pointés [15] $\mathcal{F} = \{P \mid \exists S \in \{\text{ouverts}\}, x \in S \wedge S\backslash\{x\} \subseteq P\}$. Ces formalisations fournissent également une démonstration que la définition choisie est équivalente à (2.2) dans un espace métrique.

HOL4 utilise la même définition que celle utilisée pour la convergence des suites (2.6), seul l'ordre partiel utilisé est différent : $\forall y, y' \in \mathbb{R},\ y \succeq y' \Leftrightarrow |y - x| \leq |y' - x|$. À la différence des limites de suites et de séries, il n'y a pas de notation spécifique pour les limites de fonctions.

La convergence des fonctions réelles en Mizar [Kot91a] est définie en utilisant les limites de suites de la façon suivante :

$$\left\{ \begin{array}{l} x \in \overline{\text{dom } f} \\ \exists \ell \in \mathbb{R},\ \forall (x_n) \in \mathbb{R}^{\mathbb{N}}, \end{array} \begin{array}{c} \forall n \in \mathbb{N}, x_n \in \text{dom } f\backslash\{x\} \\ \wedge \lim\limits_{n \to +\infty} x_n = x \end{array} \right\} \Rightarrow \lim_{n \to +\infty} f(x_n) = \ell. \tag{2.9}$$

L'équivalence avec la définition (2.8) a été prouvée dans le cas particulier où $D = \text{dom } f\backslash\{x\}$. Les notions de limite vers l'infini [Kot91b] et de limites à droite et à gauche [Kot91c] ont également été formalisées.

Les définitions (2.8) et (2.9) ont été formalisées dans C-CoRN pour les fonctions totales. L'équivalence entre ces deux définitions a également été démontrée. Les limites pour les fonctions partielles n'ont cependant pas été définies.

Pour finir, les limites en ACL2(r) et dans la bibliothèque d'analyse non standard de Isabelle/HOL ont été définies en utilisant les nombres réels non standards :

$$\forall y,\ y \approx x \Rightarrow f(y) \approx \ell. \tag{2.10}$$

avec $x \approx y$ signifiant que x et y ont la même partie standard.

Continuité

La continuité d'une fonction f en un point x se présente usuellement de deux façons différentes : soit comme dans la bibliothèque standard de Coq en utilisant la limite définie auparavant $\lim_{y \to x} f(y) = f(x)$, soit en utilisant le prédicat adapté :

$$\forall \varepsilon \in \mathbb{R}^+,\ \exists \delta \in \mathbb{R}^+,\ \forall y \in \mathbb{R},\ |y - x| < \delta \Rightarrow |f(y) - f(x)| < \varepsilon. \tag{2.11}$$

En PVS, la définition 2.11 est adaptée dans le cadre des fonctions partielles, et la stabilité par les opérations arithmétiques est démontrée [Dut96].

14. $\forall x\ , D(x) \Rightarrow \forall y,\ x \equiv y \Rightarrow D(y)$
15. Les ensembles contenus dans le filtre peuvent ne pas contenir le point considéré.

Comme pour les limites, Mizar utilise les suites pour définir la notion de continuité [RS90a] :

$$\forall (x_n) \in \mathbb{R}^{\mathbb{N}},\ (\forall n \in \mathbb{N}, x_n \in \text{dom } f) \wedge \lim_{n\to+\infty} x_n = x \Rightarrow f(x) = \lim_{n\to+\infty} f(x_n). \tag{2.12}$$

Pour cette définition, comme cela avait été fait pour les limites, l'équivalence avec la définition en $\forall\varepsilon\exists\delta$ de la continuité (2.11) a été démontrée. L'équivalence avec la version topologique (2.5) a également été démontrée. Dans ProofPower [Art12a], l'autre sens a été choisi : la définition est (2.11), et l'équivalence avec (2.12) a été démontrée.

Dans HOL4 et HOL Light, la limite est utilisée pour définir la continuité d'une fonction.

En ce qui concerne la bibliothèque C-CoRN, le cas est particulier car il s'agit de formaliser des mathématiques constructives. Dans ce cadre, toutes les fonctions sont continues (car une fonction discontinue permettrait de décider l'égalité de deux nombres réels). Il n'est pas possible de démontrer sans rajouter d'axiome que la continuité sur un intervalle fermé et borné $[a; b]$ implique la continuité uniforme. La continuité uniforme a donc été formalisée et est utilisée dans tous les théorèmes où les hypothèses de continuité sont en fait utilisées pour en déduire la continuité uniforme.

Différentiabilité

En ce qui concerne la différentiabilité, PVS et ProofPower [Art12a] utilisent, comme dans la bibliothèque standard de Coq, le taux d'accroissement de Newton (2.3) directement ou en utilisant la limite définie précédemment.

En HOL4 et en HOL Light, le taux d'accroissement est également utilisé, mais en passant par leurs définitions topologiques respectives de la limite ((2.5) et (2.6)). Une seconde définition de la différentiabilité est disponible dans HOL Light : la différentiabilité de Frechet dans les espaces vectoriels réels normés [Har13]. Dans la bibliothèque C-CoRN, comme c'était la cas pour la continuité, c'est la différentiabilité uniforme pour tout intervalle fermé et borné $[a; b]$ qui a été définie, c'est-à-dire qu'il existe un unique δ commun à tous les points de l'intervalle $[a; b]$. En Mizar la différentiabilité est définie pour les fonctions à plusieurs variables de la façon usuelle en supposant l'existence d'une fonction linéaire L telle que $f(x + h) - f(x) - L(h) = o(h)$ [RS90b]. Pour finir, dans la bibliothèque d'analyse non standard de ACL2(r) [Gam00], la différentiabilité est définie en utilisant la limite (2.10) sur le taux d'accroissement.

Dans toutes ces formalisations, les théorèmes usuels comme le théorème des accroissements finis et le théorème des valeurs intermédiaires ont été démontrés.

2.4.3 Les intégrales

Les définitions vues précédemment pour les limites et la différentiabilité sont toutes équivalentes lorsqu'on se place dans le cadre des fonctions à une seule variable. Pour les intégrales, ce n'est cependant pas le cas. Par exemple, la fonction indicatrice $\mathbf{1}_{\mathbb{Q}}$ qui vaut 1 sur les entrées rationnelles et 0 sinon est intégrable au sens de Lebesgue mais pas au sens de Riemann. Ces deux intégrales sont les plus courantes. Seul PVS les formalise toutes les deux : l'intégrale de Riemann est celle qui est principalement utilisée pour l'analyse réelle, tandis que l'intégrale de Lebesgue est utilisée pour la théorie de la probabilité. L'égalité de ces deux intégrales a été démontrée formellement en PVS pour les fonctions intégrables à la fois au sens de Riemann et de Lebesgue.

Toutes les intégrales présentées ci-dessous ont été formalisées pour le type privilégié de fonction de chaque prouveur : les fonctions totales dans les prouveurs basés sur HOL, et les fonctions partielles pour Mizar, PVS et C-CoRN/MathClasses.

L'intégrale de Riemann

PVS [But09] et Mizar [EK99] définissent l'intégrabilité au sens de Riemann comme la convergence des sommes de Riemann $S(f, \sigma, \xi) = \sum_{i=0}^{n}(\sigma_{i+1} - \sigma_i)f(\xi_i)$, avec σ et ξ deux suites finies telles que $\forall i \in [\![0, n]\!],\ \sigma_i \leq \xi_i \leq \sigma_{i+1}$. Cette définition est équivalente à la définition choisie en Coq et elle permet d'obtenir directement la valeur de l'intégrale comme limite de ces sommes.

Les intégrales de C-CoRN sont définies pour les fonctions uniformément continues entre a et b, avec $a \leq b$, comme la limite des sommes de Riemann avec $\sigma = (a + k \cdot \frac{b-a}{n+1})_{0\leq k\leq n+1}$ et $\xi = (a + k \cdot \frac{b-a}{n+1})_{0\leq k\leq n}$. Comme pour la convergence (section 2.4.1), il y a un problème de nommage : ces sommes sont appelées "sommes de Darboux" au lieu de "sommes de Riemann". L'intégrabilité n'est pas formalisée car seules les fonctions uniformément continues sont considérées ici et sont donc intégrables.

Dans MathClasses, la définition de l'intégrale de Riemann est légèrement différente. Elle est définie d'abord sur les fonctions en escalier, puis, en utilisant plusieurs monades dont la monade de complétion,

les fonctions en escalier sont lissées pour définir l'intégrabilité pour les fonctions plus générales. Comme pour les autres constructions de MathClasses, l'intégrale peut être calculée de façon effective. On peut noter au passage, que l'intégrale de Stieltjes [OS10] est également formalisée dans cette bibliothèque.

L'intégrale de Riemann a également été définie dans ACL2(r). Dans le même esprit que pour les limites, elle a été définie dans le cas où la fonction f est continue comme étant la partie standard de $S(f,\sigma,\xi) = \sum_{i=0}^{N}(\sigma_{i+1}-\sigma_i)f(\xi_i)$ avec N un entier non standard $\sigma = (a + k\cdot\frac{b-a}{N+1})_{0\leq k\leq N+1}$ et $\xi = (a + k\cdot\frac{b-a}{N+1})_{0\leq k\leq N}$.

L'intégrale de Henstock-Kurzweil

HOL4 [SGL+13], HOL Light [Har13], Isabelle/HOL et ProofPower [Art12a] disposent d'une définition de l'intégrale de Henstock-Kurzweil, aussi appelée intégrale de jauge. I_f est l'intégrale de Henstock-Kurzweil de la fonction f entre a et b si

$$\begin{aligned}&\forall\varepsilon\in\mathbb{R}^+,\exists g:[a;b]\to\mathbb{R}^+,\forall\sigma\in\{\text{subdivision de }[a;b]\},\ \forall\xi\in\mathbb{R}^{|\sigma|-1},\\&\quad(\forall i<|\sigma|,\ \sigma_i\leq\xi_i\leq\sigma_{i+1}\wedge\sigma_{i+1}-\sigma_i<g(\xi_i))\Rightarrow|S(f,\sigma,\xi)-I_f|<\varepsilon\end{aligned}\tag{2.13}$$

Cette intégrale est une généralisation de l'intégrale de Riemann.

Théorie de la mesure et intégrale de Lebesgue

La théorie de la mesure a été formalisée dans les bibliothèques de HOL4 [MHT10], Isabelle/HOL [HH11], Mizar et PVS. Elle est utilisée pour définir l'intégrale de Lebesgue ainsi que la théorie de la probabilité. Ces formalisations commencent par définir une σ-algèbre $\mathcal{A}$ sur un ensemble X de la façon suivante :

$$\begin{cases}\emptyset\in\mathcal{A}\\ \forall A\in\mathcal{A},\ X\backslash A\in\mathcal{A}\\ \forall(A_n)\in\mathcal{A}^{\mathbb{N}},\ \bigcup_{n\in\mathbb{N}}A_n\in\mathcal{A}\end{cases}\tag{2.14}$$

Ensuite, une mesure $\mu:\mathcal{A}\to\mathbb{R}\cup\{+\infty\}$ est définie par

$$\begin{cases}\mu(\emptyset)=0\\ \forall\,(A_n)\in\mathcal{A}^{\mathbb{N}},\ (\forall i,j\in\mathbb{N},\ i\neq j\Rightarrow A_i\cap A_j=\emptyset)\Rightarrow\displaystyle\sum_{n\in\mathbb{N}}\mu\,(A_n)=\mu\left(\bigcup_{n\in\mathbb{N}}A_n\right)\end{cases}\tag{2.15}$$

Ces définitions nécessitent un point en plus des nombres réels pour représenter $+\infty$. HOL4 et Isabelle/HOL définissent $\overline{\mathbb{R}}=\mathbb{R}\cup\{+\infty;-\infty\}$ comme un type algébrique et définissent l'ordre, l'addition et la multiplication dessus. PVS n'étend que les réels positifs comme une paire formée d'un booléen et d'un nombre réel. Cela est suffisant pour définir la théorie de la mesure. De plus, l'addition et la multiplication sont plus simples à définir sur $\mathbb{R}^+\cup\{+\infty\}$ que sur $\overline{\mathbb{R}}$.

Dans toutes ces formalisations, l'intégrale est définie dans un premier temps sur les fonctions étagées :

$$\int\left(\sum_{k=0}^{n}\alpha_i\chi_{A_i}\right)d\mu=\sum_{k=0}^{n}\alpha_i\mu(A_i)\tag{2.16}$$

puis sur les fonctions positives :

$$\int f\,d\mu=\sup\left\{\int g\,d\mu\;\middle|\;g\text{ une fonction étagée telle que }\mu\{x|g(x)\leq f(x)\}=0\right\}.\tag{2.17}$$

Pour finir, une fonction f est intégrable si les intégrales des fonctions $f^+=\max\{f,0\}$ et $f^-=\max\{-f,0\}$ sont toutes les deux finies. La valeur est alors

$$\int f\,d\mu=\int f^+\,d\mu-\int f^-\,d\mu.\tag{2.18}$$

Dans toutes ces formalisations, l'intégrale de Lebesgue est définie en utilisant les intégrales définies ci-dessus avec la σ-algèbre engendrée par les intervalles et en utilisant la mesure de Lebesgue λ engendrée par $\tilde{\lambda}([a;b])=b-a$. La théorie de la probabilité est également formalisée dans ces systèmes en utilisant les intégrales basées sur la théorie de la mesure.

L'approche choisie dans HOL Light est légèrement différente : la mesure d'un ensemble est définie comme l'intégrale de jauge de la fonction indicatrice sur cet ensemble. De plus, une fonction est mesurable s'il existe une suite de fonctions continues convergeant presque sûrement vers cette fonction [Har13].

En ACL2(r), il existe une formalisation de la mesure de Lebesgue, mais elle n'est utilisée que pour démontrer qu'il existe des ensembles non mesurables au sens de Lebesgue [CG10].

Théorème fondamental de l'analyse

Tous les systèmes disposant d'une intégrale démontrent, sous des hypothèses variées, l'une ou les deux de ces relations entre dérivée et intégrale :

$$\int_a^b f'(t)\,dt = f(b) - f(a) \tag{2.19}$$

$$\text{ou} \qquad \frac{d}{dx}\left(x \mapsto \int_a^x f(t)\,dt\right) = f(x) \tag{2.20}$$

La formule (2.19) est le théorème fondamental de l'analyse. Il est démontré dans tous les prouveurs concernés ici pour leur intégrale principale [Har98, SGL+13, EWS01, CF03, But09, Art12a, KMM00]. En dehors des systèmes utilisant l'intégrale de jauge, le théorème suppose que la fonction f est continue sur l'intervalle $[a; b]$.

2.4.4 Nombres complexes

Toutes les bibliothèques présentées ici disposent d'une formalisation des nombres complexes. Ils sont généralement définis comme une paire de réels, mais il y a quelques différences en fonctions du système utilisé.

Mizar [Byl90a] les définit comme l'union de l'ensemble $\mathbb{R}$ des réels décrits dans la section 2.3.3 et de l'ensemble des fonctions $f : \{0, 1\} \to \mathbb{R}$ telles que $f(1) \neq 0$. Comme dans la construction des nombres réels, cette approche permet d'une part d'obtenir naturellement l'inclusion des nombres réels dans l'ensemble des nombres complexes, et d'autre part de garantir l'unicité de la représentation de chaque réel vu comme un complexe.

PVS dispose de deux définitions des nombres complexes. La définition principale est une axiomatisation : `complex` est un sous-type de `number_field` contenant un élément `i` tel que `i * i = -1` et dont les éléments s'écrivent `x + y * i` avec `x` et `y` réels. Cette définition permet d'obtenir gratuitement tous les résultats d'arithmétique sur les corps et il est démontré qu'il s'agit d'un sur-type des nombres réels `real`. Dans la définition alternative, les nombres complexes sont des paires de réels.

HOL Light définit les nombres complexes comme une notation pour l'ensemble $\mathbb{R}^2$. À ce titre, les nombres complexes disposent par construction de toutes les définitions et tous les théorèmes disponibles pour l'espace euclidien $\mathbb{R}^2$. Les seuls théorèmes spécifiques sont ceux concernant la multiplication.

Les nombres complexes de C-CoRN [GPWZ02] sont définis comme un corps sur `CC_set := {Re : IR; Im : IR}`, où `IR` est le type représentant les réels. Cela permet de récupérer toutes les définitions, les théorèmes et les tactiques d'arithmétique disponibles pour cette structure.

Il est à noter que les nombres complexes de ACL2(r) sont une généralisation des rationnels complexes existants dans ACL2 : en raison de l'absence de type en ACL2, le constructeur `(complex x y)` est utilisé pour construire les rationnels complexes avec les hypothèses `(rationalp x)` et `(rationalp y)`. Il permet maintenant de construire les nombres complexes en utilisant les hypothèses `(realp x)` et `(realp y)`.

Concernant les théorèmes d'arithmétique, deux stratégies ont été adoptées : soit redémontrer tous les théorèmes à la main, soit s'appuyer sur des structures existantes pour récupérer tous les théorèmes concernant les groupes, les anneaux, voire même les corps. En Mizar, HOL4 [SLG+13], ProofPower-HOL [Art12b], ainsi que dans la bibliothèque des complexes comme paires de réels de PVS, ces théorèmes ont été démontrés à la main. Dans la bibliothèque axiomatique de PVS et dans C-CoRN, les théorèmes concernant les opérations de corps sont une conséquence directe de la définition des nombres complexes. Dans HOL Light, les propriétés de groupe étaient déjà disponibles et il a suffi d'ajouter les théorèmes concernant la multiplication. Pour finir, dans Isabelle/HOL, ces théorèmes sont obtenus en démontrant que les nombres complexes forment un corps normé.

Quelques fonctions usuelles comme l'exponentielle complexe $e^{x+iy} = e^x(\cos y + i \sin y)$, le module et l'argument d'un nombre complexe, ont été définies dans ces différents systèmes. Concernant l'argument, PVS le définit pour un nombre complexe non nul comme l'ensemble

$$\arg z = \{y \in \mathbb{R} \mid \exists x \in \mathbb{R},\ z = \exp x\,(\cos y + i \sin y)\}.$$

Les autres prouveurs choisissent l'élément de cet ensemble dans $]-\pi;\pi]$ ou dans $[0;2\pi[$.

Isabelle/HOL, HOL Light, Mizar et la bibliothèque de PVS définissant les complexes par des paires disposent de théorèmes d'analyse pour les fonctions à valeurs complexes. Dans Mizar, la convergence de suites et de séries [SK97], la continuité et la différentiabilité de fonctions complexes [MWS01, PYSN09], ainsi que l'intégrabilité au sens de Lebesgue de fonctions complexes [NES08] ont été formalisées. L'intégrale de Lebesgue pour les fonctions à valeurs complexes a été formalisée dans PVS. En ce qui concerne Isabelle/HOL et HOL Light, une grande partie de l'analyse complexe provient de l'analyse dans $\mathbb{R}^2$, par construction dans HOL Light et par démonstration dans Isabelle/HOL. La dérivée complexe a été définie dans ces deux systèmes. Dans HOL Light, des notions spécifiques aux nombres complexes ont été formalisées, à l'image des intégrales de chemin [Har07].

2.4.5 Fonctions élémentaires

Les notions formalisées ci-dessus sont utilisées pour définir les fonctions élémentaires : l'exponentielle, le logarithme, les fonctions trigonométriques et leurs inverses. La plupart des prouveurs disposent de toutes ces fonctions, mais leurs définitions varient, par exemple en utilisant des séries entières ou des intégrales.

En utilisant les séries entières

Comme cela a déjà été signalé, les séries et séries entières sont souvent utilisées pour définir la fonction exponentielle, le sinus et le cosinus. Les bibliothèques de C-CoRN, HOL4 [Har94], la bibliothèque sur les séries entières de D. Lester en PVS et ACL2(r) [GK01] les utilisent :

$$\exp(x) = \sum_{n\in\mathbb{N}} \frac{x^n}{n!}, \qquad \sin(x) = \sum_{n\in\mathbb{N}} \frac{(-1)^n x^{2n+1}}{(2n+1)!}, \qquad \cos(x) = \sum_{n\in\mathbb{N}} \frac{(-1)^n x^{2n}}{(2n)!}.$$

Dans MathClasses, exp, sin et arctan sont également définies en utilisant les séries entières [O'C08, KS13]. Puis, d'autres fonctions, comme cos et ln, sont définies à partir de celles-ci. Pour finir, π est défini à partir de arctan en utilisant une formule semblable à la formule de Machin.

HOL Light définit la fonction exponentielle complexe avec les séries entières, puis l'utilise pour définir la fonction exponentielle réelle et les fonctions trigonométriques. Mizar utilise la même approche pour définir les fonctions trigonométriques, mais pas pour la fonction exponentielle réelle. Ces deux formalisations sont les seules à définir les séries entières sur les nombres complexes.

En utilisant les intégrales

Dans PVS et C-CoRN, le logarithme naturel est défini pour tout réel strictement positif x en utilisant l'intégrale de Riemann :

$$\ln x = \int_1^x \frac{dt}{t}.$$

PVS et Mizar définissent l'inverse de la tangente en utilisant l'intégrale :

$$\arctan(x) = \int_0^x \frac{dt}{1+t^2}.$$

Cette définition de arctan est alors utilisée pour définir l'inverse des fonctions sinus et cosinus.

En utilisant l'inverse de fonctions

ACL2(r) définit un opérateur générique [GC09] pour construire les fonctions inverses des fonctions à la fois injectives et surjectives. Comme la majorité des fonctions usuelles sont continues, il est naturel d'utiliser le théorème des valeurs intermédiaires pour démontrer la surjectivité, et ainsi définir l'inverse des fonctions continues. C'est ce que fait ACL2(r) pour définir le logarithme naturel et l'inverse des fonctions trigonométriques.

HOL Light, Isabelle/HOL, Mizar [RN91a] et ProofPower [Art12a] définissent également le logarithme naturel (et le logarithme généralisé en Mizar) à partir de l'inverse de la fonction exponentielle, mais sans définir de théorie générale sur les fonctions inverses.

PVS fait le chemin inverse en définissant la fonction exponentielle comme l'inverse de la fonction logarithme. Les fonctions sinus et cosinus sont également définies à partir de leurs inverses arcsin et arccos.

Autres approches

Mizar [Rac91] définit l'exponentielle générale en utilisant les limites :

$$\forall a, b \in \mathbb{R},\ \forall s \in \mathbb{Q}^{\mathbb{N}},\ \lim_{n\to+\infty} s_n = b \Rightarrow \lim_{n\to+\infty} a^{s_n} = a^b \tag{2.21}$$

avec $\forall a \in \mathbb{R},\ \forall (p,q) \in \mathbb{Z} \times (\mathbb{N}\backslash\{0\}),\ a^{\frac{p}{q}} = \sqrt[q]{a^p}$. La constante de Neper e est définie comme la limite de la suite convergente $n \mapsto (1 + \frac{1}{n+1})^{n+1}$.

Pour définir le sinus et le cosinus, Mizar et HOL Light utilisent la fonction exponentielle complexe avec les formules d'Euler :

$$\sin(x) = \frac{e^{ix} - e^{-ix}}{2}, \qquad \cos(x) = \frac{e^{ix} + e^{-ix}}{2}.$$

Dans toutes les formalisations étudiées, la fonction tangente est définie par le quotient usuel $\tan(x) = \frac{\sin(x)}{\cos(x)}$.

ProofPower [Art12a] définit la fonction exponentielle à partir d'une équation différentielle : c'est la solution de $f' = f$ telle que $f(0) = 1$. Les fonctions sinus et cosinus sont définies comme solution du système d'équations différentielles : $\sin(0) = 0,\ \cos(0) = 1,\ \sin' = \cos$ et $\cos' = -\sin$. Ce système utilise une façon peu usuelle pour définir π : c'est le générateur positif de l'ensemble des racines de la fonction sinus, c'est-à-dire $\pi > 0 \wedge \pi\mathbb{Z} = \{x \in \mathbb{R} \mid \sin x = 0\}$.

On peut noter également que, dans une ancienne version de PVS, les fonctions trigonométriques avaient été axiomatisées. Par exemple, ces fonctions étaient supposées vérifier les égalités $\sin^2(a) + \cos^2(a) = 1$ et $\cos(a) = \sin(a + \pi/2)$. Des restes de cette formalisation apparaissent toujours dans la version actuelle.

2.4.6 Automatisations de la différentiabilité

Pour faciliter leur utilisation, les systèmes de preuve formelle disposent de différentes méthodes visant à réduire la quantité de preuves ou de lemmes intermédiaires qu'un utilisateur a besoin d'écrire pour démontrer formellement un théorème. Par exemple, les systèmes disposant de preuve de style déclaratif, comme Mizar, vont tenter de déduire automatiquement la suite de théorèmes à appliquer pour passer d'une affirmation de l'utilisateur à la suivante. Comme il n'y a pas de langage pour les scripts de preuve en ACL2, ce système dispose du même type d'instanciation automatique que Mizar. On peut remarquer que plusieurs mots existent pour désigner ces procédés automatiques en fonction du système : stratégie, tactique, méthode, etc. Je les utilise de façon interchangeable.

Certains prouveurs disposent également de procédures de décisions, potentiellement incomplètes ou ne terminant pas, qu'un utilisateur peut appliquer pour terminer une branche de preuve. Par exemple, ils peuvent résoudre des formules propositionnelles du premier ordre, effectuer des simplifications à partir de bases de données, rechercher des preuves à la façon de Prolog, utiliser des systèmes de réécriture, utiliser un solveur SMT et bien d'autres. Je ne détaillerai pas ces procédures génériques.

En matière d'analyse réelle, il est souvent nécessaire de démontrer que certaines fonctions données sont différentiables. Comme ces énoncés sont nombreux mais généralement simples à prouver, de nombreux systèmes fournissent des méthodes automatiques pour les résoudre. Je les aborde dans cette section.

HOL Light

En HOL Light [Har98], la tactique `DIFF_CONV` démontre des résultats à propos de la différentiabilité d'expression. Cette méthode est syntaxique et applique de façon répétitive les règles de différentiabilité à propos des sommes, différences, produits et quotients. `DIFF_CONV` applique également les règles de différentiabilité de fonctions de base comme x^n ainsi que de quelques fonctions élémentaires. L'utilisateur peut augmenter lui-même cet ensemble de fonctions. La fonction dérivée peut également être déterminée par le système.

C-CoRN

Plusieurs tactiques sont disponibles dans la bibliothèque C-CoRN. La première est `Deriv` et consiste à appliquer une liste de lemmes (les règles de base) en fonction de la forme du but. Une tactique plus avancée, `New_Deriv`, vise à résoudre les buts de différentiabilité tout en gardant une trace du domaine de différentiabilité.

ACL2(r)

Ce système dispose d'une méthode `defderivative` [RG11]. Étant donnée une fonction, cette méthode définit une fonction égale à sa dérivée. Dans la pratique, la fonction est obtenue automatiquement mais le plus souvent, le résultat n'est pas utilisable immédiatement car il y a très peu de simplifications. Cependant, l'utilisateur peut démontrer l'équivalence du résultat avec une forme plus adaptée.

Comme dans les autres prouveurs, cette méthode est basée sur les règles élémentaires de différentiabilité pour l'identité, les fonctions constantes, l'addition, la multiplication, la composition et l'inverse. Il y a également un ensemble de fonctions enregistrées avec leurs dérivées, comme les fonctions élémentaires, et l'utilisateur peut en ajouter d'autre. Cette méthode repose sur la preuve de la différentiabilité de exp [RG11]. En ce qui concerne la correction, cette méthode n'est pas prouvée correcte, mais elle génère une preuve de correction qui peut être vérifiée par ACL2(r). Le domaine de différentiabilité était initialement un intervalle, mais il peut maintenant être défini comme le domaine de définition d'une fonction.

Comme ce prouveur est seulement du premier ordre, et que les règles de différentiabilité ne le sont pas, les contraintes doivent être encapsulées afin de simuler leur comportement. Les dérivées partielles sont également traitées, mais uniquement pour des fonctions à deux variables.

2.5 Conclusion

Le tableau 2.2 résume les différentes formalisations présentées dans cet état de l'art, bibliothèque standard de Coq comprise. Il montre que les systèmes de preuve formelle possèdent une grande variété dans les définitions des nombres réels qui couvre toutes les approches usuelles : axiomatisation, ou construction comme coupures de Dedekind ou suites de Cauchy. On remarque que C-CoRN/MathClasses est la seule formalisation d'analyse constructive. Cette variété est également due à la variété des cadres logiques : la logique a une forte influence sur la façon dont les nombres réels, et donc l'analyse réelle, ont été construits. Il est à noter que la formalisation de J. Harrison dans HOL Light a inspiré de nombreux développements dans les autres systèmes de la famille HOL ainsi que dans PVS, à l'image de la bibliothèque Multivariate de HOL Light portée sur Isabelle/HOL. La hiérarchie algébrique plus fine de Isabelle/HOL permet une meilleure compatibilité entre les notions formalisées (limites, dérivées, intégrales, ...) et les différents espaces (réels, complexes, ...). La définition par graphes dans Mizar et le système de sous-typage de PVS permettent quant à eux de définir et de manipuler facilement les fonctions partielles.

Cette variété de formalisation peut entraîner des difficultés pour les utilisateurs car certaines approches sont plus adaptées pour traiter tel ou tel problème et qu'aucun système n'est adapté pour tous les usages. Cette variété est également une source de problèmes pour les échanges de théorèmes et de preuve entre les systèmes, comme cela a été fait dans la famille de prouveurs HOL [Hur11]. Plus de détails sur les aspects historiques et logiques de la formalisation des nombres réels sont donnés dans [May12]. Différentes applications des assistants de preuve sur l'analyse réelle peuvent être vues sur la page web de F. Wiedijk [16]. Il y présente 100 théorèmes célèbres (certains d'entre eux dans l'analyse réelle) et leur preuve formelle dans différents systèmes.

Il apparaît que certaines formalisations, aussi élégantes soient-elles, ont été construites indépendamment de toute application, ou dans le but de prouver un théorème précis, sans se soucier des applications ultérieures. Cela entraîne des difficultés d'utilisation en raison de lemmes usuels oubliés et de l'absence de méthodes spécifiques d'automatisation. Cela peut servir de leçon pour la construction d'une bibliothèque d'analyse : tenir compte des demandes d'un maximum d'utilisateurs plutôt que tenter d'imposer une formalisation élégante mais peu fonctionnelle. Il faut de plus s'efforcer de maintenir une certaine homogénéité entre les définitions, comme par exemple dans le nommage des théorèmes.

Cet état de l'art est loin d'être exhaustif concernant les systèmes et bibliothèques formalisant l'analyse réelle. Il existe de nombreuses autres formalisations des nombres réels que je n'ai pas présentées en raison de leur faible diffusion ou utilisation. Pour donner quelques exemples, R.L. Constable, S.F. Allen et d'autres décrivent dans [CAB+85] une formalisation des nombres réels en Nuprl ; C. Jones montre comment compléter les espaces métriques en LEGO et construire des nombres réels au dessus de l'ensemble des nombres rationnels [Jon93] ; A. Ciaffaglione et P. Di-Gianantonio formalisent des nombres réels comme des flux infinis de chiffres dans $\{-1, 0, 1\}$ [CDG06] ; et pour finir, en Coq, C. Cohen présente dans [Coh12] une formalisation des nombres algébriques.

Il faut aussi citer certains travaux qui étendent les bibliothèques présentées dans cet état de l'art. Par exemple, J. Avigad et K. Donnelly formalisent la notion de O en Isabelle/HOL [AD04]. S. Richter

16. `http://www.cs.ru.nl/~freek/100/`

	ACL2(r)	Coq	C-CoRN	HOL4	PPHol	Isabelle	HOL Light	Mizar	PVS
Logique	premier ordre	types dépendants	types dép. + constr	HOL	HOL	HOL	HOL	ZF	TCC
Fonctions partielles	n.a.	fonctions totales + types dép.	types dépendants	..... fonctions totales, ε de Hilbert				graphe de fonctions	TCC
Définition des réels	analyse non-std	axiomes	suites de Cauchy	coupures de Dedekind	coupures de Dedekind	suites de Cauchy	suites de Cauchy	coupures de Dedekind	axiomes
Intégrales	Riemann	Riemann	Riemann + Stieltjes	jauge + Lebesgue	jauge	jauge + Lebesgue	jauge + Lebesgue	Riemann + Lebesgue	Riemann + Lebesgue
Analyse multivariée	1D	1D	2D	1D	1D	nD+	nD	nD+	2D
Analyse complexe	+	non	+	+	+	++	++	++	++
Automatisation dédiée	+	++	+	++	+	++	++		++

TABLEAU 2.2 – Formalisations de l'analyse réelle en un coup d'oeil.

La ligne "Logique" donne une version simplifiée du cadre logique : "types dép." correspond aux types dépendants, "constr" pour constructif, et "ZF" pour la théorie des ensembles de Zermelo-Fraenkel. La ligne "Fonctions partielles" est liée à la façon dont sont manipulées les fonctions partielles comme la division ou les fonctions dérivées. La ligne "Définition des réels" résume la façon dont sont définis les nombres réels : "suites de Cauchy" et "coupures de Dedekind" signifient qu'ils sont construits avec ces objets, "axiomes" signifie qu'ils sont définis par des axiomes, "analyse non-std" signifie que les nombres réels sont construits à partir de prédicats non standard. La ligne "Intégrales" indique les types d'intégrales formalisés. La ligne "Analyse multivariée" décrit sa disponibilité : "1D" signifie une seule variable, "2D" signifie que certains résultats ont été généralisés à $\mathbb{R}^2$, "nD" signifie une bonne analyse à plusieurs variables et "nD+" signifie que certains résultats ont été étendus à des espaces topologiques autres que $\mathbb{R}^n$. La ligne "Analyse complexe" décrit son degré de développement : "non" signifie que les nombres complexes n'ont pas été formalisés, "+" que les théorèmes sur les nombres complexes se limitent à l'arithmétique et "++" signifie l'existence d'une bibliothèque permettant de faire de l'analyse avec des fonctions à valeurs complexes. La ligne "Automatisation dédiée" tente d'évaluer le nombre, la puissance et l'étendue des méthodes automatiques sur les nombres réels : plus il y a de +, mieux c'est.

définit l'intégrale de Lebesgue en Isabelle/HOL et l'utilise pour raisonner sur des algorithmes probabilistes [Ric04]. D.R. Lester développe une théorie de la topologie en PVS [Les07]. N. Julien et I. Paşca combinent la bibliothèque standard de Coq avec des nombres réels calculables définis comme des flux infinis [JP09].

Pour conclure, on peut également citer quelques développements formels construits grâce aux systèmes présentés ici. Tout d'abord, il y a toutes les formalisations liées à la vérification de programmes utilisant des nombres à virgule flottante. Comme l'arithmétique flottante est utilisée en pratique pour approcher l'arithmétique réelle, la vérification formelle a dû suivre. La majorité des travaux ont été développés en HOL Light, par exemple [Har97], et en Coq, par exemple [BM11]. Il y a également quelques développements en ACL2, comme [MLK98], mais on doit remarquer qu'ils ont évité les nombres irrationnels et donc n'ont pas besoin d'utiliser l'extension ACL2(r).

Une autre application est l'étude des équations différentielles ordinaires et partielles. Pour les équations différentielles ordinaires, la preuve de la méthode d'Euler a été faite en Isabelle/HOL avec une spécification exécutable [IH12]. Encore pour les équations différentielles ordinaires, un opérateur de Picard exécutable a été prouvé dans C-CoRN/MathClasses [MS13]. En ce qui concerne les équation différentielles aux dérivées partielles, un développement majeur à propos de la résolution numérique de l'équation des ondes s'appuie sur la bibliothèque standard de Coq [BCF+13].

Chapitre 3

Vers une bibliothèque plus facile à utiliser

Comme cela a été vu dans les chapitres précédents, la bibliothèque standard de Coq contient des notions d'analyse réelle. Elle est malheureusement difficile à utiliser pour diverses raisons, entre autres :

- le manque d'homogénéité pour le nommage des théorèmes entraîne des difficultés pour retrouver les théorèmes souhaités, par exemple les lemmes concernant les sommes sont parfois nommés de façon complètement arbitraire, comme `tech1` qui dit qu'une somme de termes strictement positifs est strictement positive.
- il est difficile d'écrire et de manipuler des énoncés contenant des limites, des dérivées ou des intégrales à cause des types dépendants.
- certaines notions usuelles comme les limites vers $\pm\infty$ et les intégrales à paramètres sont absentes,

Les travaux présentés dans ce chapitre ont donc visé à améliorer les notions présentes dans la bibliothèque standard et à en faciliter l'utilisation. Ainsi, un effort particulier a été fait pour homogénéiser les définitions et le nommage des théorèmes (section 3.1). Un certain nombre de théorèmes supplémentaires concernant l'analyse réelle à une variable ont été démontrés (section 3.2), en particulier concernant les limites généralisées, les séries et séries entières et la continuité. Une nouvelle définition a été choisie pour l'intégrale de Riemann (section 3.3). Des théorèmes concernant les fonctions à deux variables ont été démontrés (section 3.4). Des fonctions permettant d'écrire plus naturellement les limites, dérivées et intégrales ont été définies (section 3.5). Pour finir, une tactique a été implémentée (section 3.6) afin d'automatiser les démonstrations de dérivabilité.

3.1 Méthodologie

3.1.1 Fonctions et prédicats

La formalisation des fonctions partielles n'est pas facile en Coq, contrairement à PVS. Dans la bibliothèque standard, on peut voir deux stratégies pour définir une fonction partielle f :

- étendre f pour construire une fonction totale et exprimer les conditions d'existence dans les hypothèses des théorèmes les utilisant. Cette méthode a été utilisée pour définir les fonctions logarithme népérien `ln` et racine carrée `sqrt`.
- définir un prédicat signifiant "ℓ est la valeur de la fonction f pour les entrées $x_1, \ldots, x_n$" à l'image de `derivable_pt_lim` pour les dérivées, un prédicat dans `Set` caractérisant l'existence d'une telle valeur ℓ pour des entrées données (`derivable_pt` pour les dérivées), et pour finir la fonction prenant en paramètre les entrées et une preuve d'existence et renvoyant la valeur ℓ (la fonction `derive_pt` de type `forall (f : R -> R) (x : R), derivable_pt f x -> R` pour les dérivées).

Même si la définition de l'intégrale de Riemann ne comprend pas de prédicat liant la valeur de l'intégrale à l'intégrabilité, elle entre tout de même dans la seconde catégorie dans la mesure où la fonction `RiemannInt` prend en argument une preuve d'intégrabilité.

Les deux stratégies décrites ci-dessus ont chacune leurs avantages et leurs inconvénients : dans le premier cas, l'utilisateur dispose d'une fonction facile à utiliser, mais qui n'est pas toujours facile, voire possible, à définir par le concepteur. Dans le second cas, le prédicat permet de simuler la fonction partielle de façon précise, mais les fonctions définies par cette méthode sont difficiles à manipuler en raison du terme de preuve qu'elles prennent en argument.

Un effort a été fait dans la bibliothèque Coquelicot pour avoir la même organisation pour toutes les fonctions partielles formalisées. Ainsi, les dérivées et intégrales de Riemann pour les fonctions réelles sont définies suivant le même schéma que l'addition et la multiplication pour les réels étendus. Ce schéma est inspiré de la seconde méthode pour définir les fonctions partielles : étant donné une fonction *fonction*, j'ai défini

1. un prédicat `ex_`*`fonction`*$(param)$ dans `Prop`, pour caractériser l'existence de la fonction considérée au point *param* donné. Par exemple, `ex_Rbar_plus`(a, b) signifie que $a +_{\overline{\mathbb{R}}} b$ est bien défini.
2. une fonction totale *`Fonction`*$(param)$ pouvant être utilisée dans une formule mathématique. Elle renvoie la même valeur que *fonction* sur son domaine de définition et une valeur arbitraire sinon. Pour reprendre l'exemple précédent, la fonction `Rbar_plus`(a, b) renvoie $a +_{\overline{\mathbb{R}}} b$ dans la cas où a et b ne sont pas des infinis de signes opposés et 0 sinon.
3. un prédicat `is_`*`fonction`*$(param, res)$ dans `Prop` signifiant "*fonction* est bien définie pour les paramètres *param* et renvoie *res*".

Usage des fonctions partielles

Dans la pratique, plutôt que d'utiliser le prédicat `ex_`*`fonction`* et la fonction totale *`Fonction`*, il est plus simple d'ajouter une variable et d'utiliser le prédicat `is_`*`fonction`*. Par exemple, pour exprimer en Coq le théorème $\lim(u_n + v_n) = \lim u_n +_{\overline{\mathbb{R}}} \lim v_n$ la première forme nous donne

```
Lemma is_lim_seq_plus (u v : nat -> R) (lu lv : Rbar) :
  is_lim_seq u lu -> is_lim_seq v lv
  -> ex_Rbar_plus lu lv
  -> is_lim_seq (fun n => u n + v n) (Rbar_plus lu lv).
```

et la seconde

```
Lemma is_lim_seq_plus (u v : nat -> R) (lu lv l : Rbar) :
  is_lim_seq u lu -> is_lim_seq v lv
  -> is_Rbar_plus lu lv l
  -> is_lim_seq (fun n => u n + v n) l.
```

Le premier énoncé est certes plus facile à lire, mais il peut entraîner des duplications dans la suite de la démonstration car il faudra faire les mêmes manipulations sur `Rbar_plus` et sur `ex_Rbar_plus`. Pour donner un exemple chiffré, la démonstration de $\lim(2n + 3) = +\infty$, c'est-à-dire en français "la suite $(2n + 3)_{n\in\mathbb{N}}$ est convergente et admet pour limite $+\infty$", en appliquant les lemmes un par un, demande
- 8 lignes avec le second énoncé,
- 15 lignes avec le premier énoncé (dont 3 lignes pour prouver que `ex_Rbar_plus (Rbar_mult 2 p_infty) 3` et 7 lignes pour prouver que `Rbar_plus (Rbar_mult 2 p_infty) 3 = p_infty`).

Théorèmes d'unicité

Pour concilier les facilités de lecture offertes par les fonctions totales et les démonstrations plus simples des buts de la forme `is_`*`fonction`*, les deux lemmes suivants ont été démontrés pour chaque fonction partielle étudiée :

1. une preuve d'unicité

   Lemma `is_`*`fonction`*`_unique` $(param, res)$ `:`
   `is_`*`fonction`*$(param, res) \to$ *`Fonction`*$(param)$ `=` res`.`

2. une preuve de correction

   Lemma *`Fonction`*`_correct` $(param)$ `:`
   `ex_`*`fonction`*$(param) \to$ `is_`*`fonction`*$(param,$ *`Fonction`*$(param))$`.`

Dans la bibliothèque standard, ces lemmes n'étaient pas nécessaires en raison de la forte dépendance entre les deux prédicats et la fonction.

En utilisant ces lemmes, il est alors possible dans l'exemple $\lim(2n + 3) = +\infty$ de déduire les deux énoncés "lisibles" `ex_lim_seq (fun n => 2 * INR n + 3)` et `Lim_seq (fun n => 2 * INR n + 3) = p_infty` en une ligne une fois démontré `is_lim_seq (fun n => 2 * INR n + 3) p_infty`.

Généralisation

Dans le cadre de la généralisation qui sera présentée dans le chapitre 4, il peut arriver que la fonction totale ne soit pas disponible. Dans ce cas, le prédicat `is_fonction` est utilisé systématiquement et le théorème d'unicité devient

Lemma `is_fonction_unique` $(param, res_1, res_2)$:
`is_fonction`$(param, res_1) \rightarrow$ `is_fonction`$(param, res_2) \rightarrow res_1 \simeq res_2$.

avec $\simeq$ une relation d'équivalence adaptée au type du résultat. Le plus souvent, cette équivalence $x \simeq y$ peut être comprise comme "$\forall \varepsilon > 0,\ d(x, y) < \varepsilon$" avec d une "métrique" sur l'espace considéré.

Un certain nombre de fonctions partielles développées dans la bibliothèque Coquelicot sont formalisées de façon souvent incomplète dans la bibliothèque standard. Par exemple, le prédicat définissant la convergence des suites `is_lim_seq` dans la bibliothèque Coquelicot est équivalent dans le cas des limites finies au prédicat `Un_cv` de la bibliothèque standard et à `cv_infty` dans le cas où la limite est $+\infty$. Pour toutes les notions redéfinies dans la bibliothèque Coquelicot, l'équivalence avec les versions existant dans la bibliothèque standard a été démontrée. Cela permet aux utilisateurs d'utiliser les possibilités de Coquelicot sans perdre les théorèmes énoncés en utilisant la bibliothèque standard et augmente donc la compatibilité.

3.1.2 Nommage des théorèmes

Théorèmes sur les fonctions partielles

Comme je l'ai expliqué juste au dessus, la définition des fonctions partielles ainsi que le nommage des théorèmes d'équivalence ont été uniformisés. Cela permet une utilisation plus facile de la bibliothèque. En effet, avec les théorèmes évoqués jusqu'à présent, pour transformer un but de la forme `Un_cv u l` afin de travailler avec la bibliothèque Coquelicot, il suffit de se souvenir que les limites de suites ont pour mot-clé `lim_seq` et que les théorèmes d'équivalence finissent par `_Reals`. Dans cet exemple, le théorème à appliquer est donc `is_lim_seq_Reals`. Il y a quelques exceptions, en particulier dans le cas de l'intégrale de Riemann dont la construction est plus complexe dans la bibliothèque standard, mais à défaut de fournir le théorème exact, ce nommage permet de reconnaître plus facilement les théorèmes à utiliser.

Pour les autres théorèmes concernant les fonctions partielles, le nom du prédicat ou de la fonction est suivi par le nom

- de la fonction sur lequel il est appliqué. Par exemple, pour l'addition, le mot `_plus` est ajouté, ce qui donne pour les limites de suites le théorème `is_lim_seq_plus` cité plus haut. De même, `_id` ou `_exp` sont ajoutés pour les fonctions identité et exponentielle.
- de la propriété, comme `_ext` pour l'extensionnalité globale qui donne pour les suites le théorème

```
Lemma is_lim_seq_ext (u v : nat -> R) (l : Rbar) :
  (forall n : nat, u n = v n) -> is_lim_seq u l -> is_lim_seq v l.
```

Les théorèmes autour des prédicats présentés dans le chapitre 4, `filterlim` et `filterdiff`, qui généralisent respectivement les notions de limite et de dérivée, suivent les mêmes conventions de nommage.

Théorèmes de réécriture

Pour les théorèmes de réécriture, la convention de nommage est légèrement différente. Dans la plupart des cas, les noms correspondent à une version abrégée du membre gauche de la conclusion. Par exemple, le théorème sur la distributivité de l'opposé par rapport à l'addition $\forall(x, y),\ -(x + y) = (-x) + (-y)$ se nomme `opp_plus`.

Comme ces théorèmes s'utilisent dans les deux sens, soit $-(x+y) \rightarrow (-x)+(-y)$, soit $(-x)+(-y) \rightarrow -(x + y)$, il y a plusieurs choix pour exprimer et nommer ces théorèmes. Pour les théorèmes concernant le développement/factorisation d'expression comme dans l'exemple, le sens développement a été le plus souvent choisi, pour les autres, le sens de la simplification.

Il faut aussi noter qu'un grand nombre de théorèmes ont été généralisés (voir chapitre 4 pour plus de détails). Ainsi l'exemple donné ci-dessus ne s'applique pas uniquement aux nombres réels, mais à tout couple (x, y) d'éléments d'un groupe abélien. Cela permet d'avoir des théorèmes qui s'appliquent aussi bien aux nombres réels et complexes qu'aux matrices de réels, mais peut entraîner des difficultés pour les recherches de théorèmes existants. Par exemple, un théorème sur l'addition peut avoir été démontré pour les nombres réels, mais ne pas posséder d'équivalent dans la bibliothèque généralisée : il n'y aura alors pas de résultat en cherchant ce théorème avec l'addition des groupes abéliens `plus`, alors qu'il y en aura en cherchant ce théorème avec l'addition des réels `Rplus`.

3.2 L'analyse réelle

3.2.1 L'ensemble des réels étendus $\overline{\mathbb{R}}$

Les réels n'étant pas suffisants pour parler de toutes les limites usuelles, j'ai été amenée à définir les réels étendus $\overline{\mathbb{R}} = \mathbb{R} \cup \{+\infty; -\infty\}$, ainsi que des opérations et des relations d'ordre dessus. Dans de nombreux usages, comme le rayon de convergence d'une série entière ou les intégrales de Lebesgue, un seul point infini est suffisant car on ne considère que les réels positifs. Dans ce cas, il est possible de définir des opérations d'addition et de multiplication relativement simples car l'addition est une fonction totale et il n'y a que deux cas d'indétermination pour la multiplication : $0 \times_{\overline{\mathbb{R}}} \infty$ et $\infty \times_{\overline{\mathbb{R}}} 0$. L'usage est alors de donner la valeur 0 pour une meilleur cohérence de l'intégrale de Lebesgue. Cet unique infini n'est pas suffisant lorsqu'il s'agit de traiter les opérations sur les limites de suites et de fonctions réelles.

Pour construire les réels étendus, je me suis inspirée de [HH11] :

```
Inductive Rbar :=
  | Finite  : R -> Rbar
  | p_infty : Rbar
  | m_infty : Rbar.
```

Par définition, la fonction `Finite` est une injection des réels de Coq dans les réels étendus. Pour donner un exemple, $\lim \frac{1}{n} = 0$ se formalise en `Lim_seq (fun n => / INR n) = Finite 0`. Pour simplifier son usage, je l'ai déclarée comme étant une coercion des réels dans les réels étendus. La coercion ainsi définie nous permet d'écrire plus naturellement `Lim_seq (fun n => / INR n) = 0` car `0` est alors interprété comme étant `Finite 0`.

Dans certains cas, il peut être utile de faire une coercion des réels étendus vers les réels finis pour travailler dans $\mathbb{R}$. Cela se retrouve par exemple dans la définition des fonctions totales permettant d'écrire les séries de nombres réels : le nombre `Series a` représentant la séries de terme général `a` doit pouvoir être utilisé comme un nombre réel, or il représente également la limite des sommes partielles `sum_n a` et peut donc s'écrire `Lim_seq (sum_n a)` qui est à valeur dans `Rbar`. La définition la plus simple est donc `Series a := real (Lim_seq (sum_n a))`, avec `real : Rbar -> R` la fonction qui renvoie la valeur des réels étendus finis et 0 dans les cas infinis. Comme pour la fonction `Finite`, j'ai déclaré `real` comme une coercion afin d'en simplifier l'usage.

Les opérations arithmétiques sur $\overline{\mathbb{R}}$

Comme je l'ai fait remarquer plus haut dans le cas avec un seul infini, les opérations arithmétiques sur les réels étendus ne sont en général pas des fonctions totales. Cela reste vrai dans le cas des réels étendus avec deux infinis, comme par exemple $(+\infty) + (-\infty)$ qui n'est pas défini. En fait, la seule opération vraiment totale est l'opposé qui se définit très naturellement par

```
Definition Rbar_opp (x : Rbar) :=
  match x with
    | Finite x => Finite (- x)
    | p_infty  => m_infty
    | m_infty  => p_infty
  end.
```

Même si la fonction inverse n'est pas une fonction totale, elle a été définie de façon analogue :

```
Definition Rbar_inv (x : Rbar) : Rbar :=
  match x with
    | Finite x => Finite (/ x)
    |    _     => Finite 0
  end.
```

Un autre choix aurait été de définir `Rbar_inv 0 = p_infty`. En effet, cela revient à définir la fonction `Rbar_inv` comme la limite à droite en chaque point de la fonction inverse. Ce choix est cependant plus arbitraire : en effet, il n'y a pas de raison de choisir la limite à droite plutôt que la limite à gauche qui donnerait une fonction différente. La solution choisie offre de plus la possibilité d'utiliser les théorèmes de réécriture sur la fonction inverse `Rinv` car on a la propriété que pour tout `x` réel, `Finite (/ x) = Rbar_inv (Finite x)`. Le cas de non définition en 0 est hérité de la fonction inverse `Rinv` sur les nombres réels qui a été axiomatisée comme une fonction totale dont la valeur en 0 n'est pas spécifiée. Cela permet de ne faire apparaître l'hypothèse "`x` $\neq$ `0`" que dans les théorèmes utilisant les propriétés de la fonction inverse.

$+_{\overline{\mathbb{R}}}$	$-\infty$	x	$+\infty$
$-\infty$	$-\infty$	$-\infty$	non def.
y	$-\infty$	$x+y$	$+\infty$
$+\infty$	non def.	$+\infty$	$+\infty$

$\times_{\overline{\mathbb{R}}}$	$-\infty$	$x<0$	$x=0$	$x>0$	$+\infty$
$-\infty$	$+\infty$	$+\infty$	non def.	$-\infty$	$-\infty$
$y<0$	$+\infty$	$x \cdot y$			$-\infty$
$y=0$	non def.				non def.
$y>0$	$-\infty$				$+\infty$
$+\infty$	$-\infty$	$-\infty$	non def.	$+\infty$	$+\infty$

TABLEAU 3.1 – Définition de `Rbar_plus` et `Rbar_mult`

Concernant la définition de l'addition `Rbar_plus` et de la multiplication `Rbar_mult`, le choix de la formalisation n'a pas été aussi simple. En effet, comme on peut le voir dans les tableaux 3.1, le domaine de définition de ces fonctions est plus complexe que celui des fonctions opposé et inverse définies ci-dessus. Les cases contenant "non def." correspondent aux cas où ces fonctions étendues ne sont pas continues : il n'est, par exemple, pas possible de trouver une valeur $\ell \in \overline{\mathbb{R}}$ telle que pour toutes suites u et v, $\lim u = +\infty \wedge \lim v = -\infty \Rightarrow \lim(u+v) = \ell$. J'ai donc choisi d'aborder la définition de ces fonctions partielles selon le plan présenté dans la section 3.1 avec un prédicat d'existence, une fonction totale et un prédicat permettant d'exprimer "le réel étendu z est la somme, respectivement le produit, de x et y".

L'addition a été définie en passant par une fonction auxiliaire `Rbar_plus'` à valeur dans `option Rbar` qui correspond au tableau 3.1. Cela permet de définir simplement les deux prédicats et la fonction totale correspondant à l'addition :

```
Definition Rbar_plus (x y : Rbar) :=
  match Rbar_plus' x y with Some z => z | None => Finite 0 end.
Definition is_Rbar_plus (x y z : Rbar) : Prop :=
  Rbar_plus' x y = Some z.
Definition ex_Rbar_plus (x y : Rbar) : Prop :=
  match Rbar_plus' x y with Some _ => True | None => False end.
```

La multiplication a été définie de la même façon que l'addition avec une fonction `Rbar_mult'` à valeur dans `option Rbar` puis une fonction totale `Rbar_mult` et un prédicat `is_Rbar_mult` définis à partir de celle-ci. Concernant le prédicat d'existence `ex_Rbar_mult`, l'utilisation de `Rbar_mult'` a été rapidement abandonné car cela imposait de distinguer les cas $x = 0$, $x > 0$ et $x < 0$ pour démontrer l'existence ou non des produits comme $x \times_{\overline{\mathbb{R}}} (+\infty)$, alors qu'il est suffisant de vérifier si x est nul ou non. Le prédicat `ex_Rbar_mult` a donc été modifié pour ne garder que la distinction nul ou non pour la multiplication entre un nombre réel et un infini.

Pour finir la définition des opérations arithmétiques, la soustraction et la division ont été définies en utilisant les opérations précédentes de la même façon que dans la bibliothèque standard, c'est-à-dire à partir de l'addition et de l'opposé pour la soustraction :

```
Definition is_Rbar_minus (x y z : Rbar) : Prop :=
  is_Rbar_plus x (Rbar_opp y) z.
```

et à partir de la multiplication et de l'inverse pour la division :

```
Definition is_Rbar_div (x y z : Rbar) : Prop :=
  is_Rbar_mult x (Rbar_inv y) z.
```

Les prédicats d'existence et les fonctions totales correspondantes ont été définis de la même façon.

Il faut noter que la division dispose d'une seconde version dans le cas où le diviseur est strictement positif :

```
Definition Rbar_div_pos (x : Rbar) (y : posreal) :=
  match x with
    | Finite x => Finite (x/y)
    |    _     => x
  end.
```

Cette version n'est utilisée que rarement en raison du type `posreal` du diviseur, mais elle permet de définir et de manipuler plus facilement les fonctions totales définies dans la section 3.5.

Autres prédicats et fonctions usuels sur $\overline{\mathbb{R}}$

Comme pour les opérations arithmétiques, les fonctions sur les réels étendus `Rbar_min` renvoyant le minimum et `Rbar_abs` renvoyant la valeur absolue, ainsi que les prédicats d'ordre `Rbar_lt` pour l'ordre strict et `Rbar_le` pour l'ordre large, ont été définis en utilisant les définitions de la bibliothèque standard, puis en donnant les valeurs manquantes pour les deux infinis. Par exemple la définition de l'ordre strict est

```
Definition Rbar_lt (x y : Rbar) : Prop :=
  match x,y with
    | p_infty, _ | _, m_infty => False
    | m_infty, _ | _, p_infty => True
    | Finite x, Finite y => x <ℝ y
  end.
```

Pour `Rbar_lt`, ce choix de construction est naturel car il n'est pas possible de le construire sans utiliser l'inégalité stricte des réels. Les autres prédicats et fonctions aurait pu être reconstruits dans $\overline{\mathbb{R}}$, mais cela impose de démontrer des théorèmes de compatibilité dans le cas fini et aurait compliqué certaines démonstrations. Par exemple, pour définir `Rbar_le`, ce choix est moins naturel. En effet, il est possible de le définir sur le même modèle que `Rle` :

```
Definition Rle (r1 r2:R) : Prop := r1 < r2 \/ r1 = r2.
```

Ce choix aurait cependant obligé à refaire les démonstrations de `Rle` pour `Rbar_le`. Pour donner un exemple chiffré, la démonstration de la transitivité $\forall x, y, z,\ x \leq y \wedge y \leq z \Rightarrow x \leq z$ demande 6 lignes de preuve avec la définition `Rbar_lt x y \/ x = y`, alors qu'elle ne demande que 2 lignes avec la définition que nous avons choisie. Cela est lié au fait que les cas où au moins une des variables est infinie sont prouvés automatiquement avec notre définition, il ne reste alors que le cas où toutes les variables sont réelles. Avec l'autre définition, il faut recommencer la démonstration faite pour les réels en traitant séparément chacun des cas `Rbar_lt x y` et `x = y` avec l'autre définition.

3.2.2 Limite de suites et de fonctions réelles

L'ajout des valeurs $+\infty$ et $-\infty$ aux nombres réels de Coq a permis de généraliser la notion de limite. Cependant, les définitions en ε-δ usuelles comme

$$\begin{array}{rcl} \lim\limits_{n\to+\infty} (u_n)_{n\in\mathbb{N}} = \ell & \Leftrightarrow & \forall \varepsilon > 0, \quad \exists N \in \mathbb{N}, \quad \forall n \in \mathbb{N}, \qquad N \leq n \Rightarrow |u_n - \ell| < \varepsilon, \quad \text{et} \\ \lim\limits_{x\to a} f(x) = +\infty & \Leftrightarrow & \forall M \in \mathbb{R}, \quad \exists \delta > 0, \quad \forall x \in \mathbb{R}\backslash\{a\}, \quad |x - a| < \delta \Rightarrow M < f(x) \end{array}$$

conduisent à considérer 12 cas :

- 3 pour les limites de suites : $\lim u_n \in \mathbb{R}$, $\lim u_n = +\infty$ et $\lim u_n = -\infty$, et
- 3×3 pour les limites de fonctions : $\lim_{x\to a} f(x) \in \mathbb{R}$, $\lim_{x\to a} f(x) = +\infty$ et $\lim_{x\to a} f(x) = -\infty$ avec une définition différente suivant que a est fini, ou égal à $+\infty$ ou $-\infty$.

Ces 12 cas entraînent une augmentation des cas à traiter comme par exemple pour le théorème suivant

$$\forall c \in \mathbb{R},\ \forall a, \ell \in \overline{\mathbb{R}}, \quad \lim_{x\to a} f(x) = \ell \Rightarrow \lim_{x\to a} (c \times_{\mathbb{R}} f(x)) = c \times_{\overline{\mathbb{R}}} \ell$$

qui engendre 21 cas à étudier dans le cas des fonctions et 7 dans le cas des suites : pour chacun des cas $a \in \mathbb{R}$, $a = +\infty$ et $a = -\infty$ pour les fonctions et $a = +\infty$ pour les suites, il faut traiter 1 but dans le cas où ℓ est fini et 3 buts pour $\ell = +\infty$ et $\ell = -\infty$ pour traiter le signe de c. Même s'il est possible de réduire le nombre de cas à étudier, en montrant par exemple que l'on peut supposer $c \geq 0$ et $\ell \geq 0$, il faut tout de même faire quatre fois les mêmes démonstrations pour chacun des théorèmes.

Dans un premier temps, j'ai introduit la notion de voisinage autour d'un point $a \in \overline{\mathbb{R}}$ par un prédicat caractérisant la propriété "être un voisinage de a" :

```
Definition Rbar_locally (a : Rbar) (P : R -> Prop) : Prop :=
  match a with
    | Finite a => exists eps : posreal, forall x : R, Rabs (x - a) < eps -> P x
    | p_infty => exists M : R, forall x : R, M < x -> P x
    | m_infty => exists M : R, forall x : R, x < M -> P x
  end.
```

Cette première définition de voisinage n'est cependant pas suffisante pour pouvoir exprimer les limites souhaitées. En effet, les ensembles appartenant à `Rbar_locally a` contiennent tous le point `a` dans le cas où celui-ci est fini. En utilisant cette définition de voisinage, il n'est donc pas possible de démontrer des théorèmes comme $\lim_{x\to 0} x^{-2} = +\infty$ car il faut démontrer que la fonction $x \mapsto x^{-2}$ prend des valeurs aussi grandes que souhaitées sur les intervalles de la forme $]-\varepsilon;\varepsilon[$, or la valeur de la fonction `Rinv` en 0 n'étant pas axiomatisée, il n'est pas possible de démontrer quoi que ce soit sur la valeur de 0^{-2}. Une seconde notion de voisinage, nommée `Rbar_locally'`, a été définie afin de représenter les voisinages "éventuellement" pointés, c'est-à-dire en remplaçant le prédicat pour le cas fini de `Rbar_locally` par `∃ eps : posreal, ∀ x : R, Rabs (x - a) < eps → x ≠ a → P x`. Les voisinages considérés peuvent ne pas contenir le point `a`.

Afin d'homogénéiser les définitions, le voisinage de $+\infty$ dans les entiers naturels a également été défini :

```
Definition eventually (P : nat -> Prop) : Prop :=
  exists N : nat, forall n, (N <= n)%nat -> P n.
```

En utilisant ces deux définitions, les limites définies ci-dessus deviennent

$$\lim_{n\to+\infty} (u_n)_{n\in\mathbb{N}} = \ell \quad \Leftrightarrow \quad \forall \varepsilon > 0,\ (n \in \mathbb{N} \mapsto |u_n - \ell| < \varepsilon) \in \texttt{eventually}, \text{ et}$$
$$\lim_{x\to a} f(x) = +\infty \quad \Leftrightarrow \quad \forall M \in \mathbb{R},\ (x \in \mathbb{R} \mapsto M < f(x)) \in \texttt{Rbar\_locally'}(a)$$

L'usage de `Rbar_locally'` a ainsi permis de réduire les définitions pour les fonctions à 3 cas au lieu de 9 en s'affranchissant de la distinction entre a fini ou non. Cela donne donc un total de 6 cas pour définir les limites généralisées. `Rbar_locally'` a également permis de limiter les cas à traiter dans un certain nombre de théorèmes usuels sur les limites de fonctions. En effet, les théorèmes concernant les opérations arithmétiques et l'ordre ne dépendant pas du point auquel la limite est considérée, seuls 3 cas par limite sont à étudier au lieu de 9.

Dans la version actuelle de Coquelicot, une définition encore plus générale est utilisée : le prédicat `filterlim` défini par

$$\texttt{filterlim}(f, \texttt{F}, \texttt{G}) \quad \Leftrightarrow \quad \forall P \in \texttt{G},\ f^{-1}(P) \in \texttt{F},$$

où `F` et `G` sont des collections de sous-ensembles des espaces de départ et d'arrivée de la fonction f. Ces collections sont représentées par le type `(U → Prop) → Prop` où `U` est le type de l'ensemble considéré. Dans le cas des familles de voisinages `Rbar_locally` et `Rbar_locally'`, on a `U = R` et `filterlim(`f`, Rbar_locally' a, Rbar_locally l)` se lit "l'image réciproque de tout voisinage de `l` est un voisinage de `a`, éventuellement privé du point `a`".

Les limites de fonctions et de suites sont définies comme deux instances du prédicat `filterlim` :

```
Definition is_lim_seq (u : nat -> R) (l : Rbar) :=
  filterlim u eventually (Rbar_locally l).
Definition is_lim (f : R -> R) (x l : Rbar) :=
  filterlim f (Rbar_locally' x) (Rbar_locally l).
```

En utilisant ces définitions, l'utilisateur écrit "`is_lim_seq (fun n ⇒ 2 * INR n + 3) p_infty`" au lieu de "`filterlim (fun n ⇒ 2 * INR n + 3) eventually (Rbar_locally p_infty)`" pour exprimer "la suite $(2n+3)_{n\in\mathbb{N}}$ converge vers $+\infty$". On remarque au passage qu'il est également possible d'exprimer cet énoncé en regardant cette suite comme une suite d'entiers avec "`filterlim (fun n ⇒ (2 * n + 3)%nat) eventually eventually`". Cela permet en particulier de démontrer que si une suite u converge vers $\ell \in \overline{\mathbb{R}}$, alors la suite $(u_{2n+3})_{n\in\mathbb{N}}$ converge également vers ℓ en appliquant le théorème de composition `filterlim_comp`.

Les propriétés utilisées que doivent vérifier les voisinages sont les suivantes :
- l'espace tout entier appartient à la collection : `U ∈ F`,
- stabilité par intersection : `P ∈ F` et `Q ∈ F ⇒ P ∩ Q ∈ F`,
- stabilité par sur-ensemble : `P ∈ F` et `P ⊆ Q ⇒ Q ∈ F`.

Ces propriétés définissent un filtre, caractérisé par le prédicat `Filter` dans la bibliothèque. Cette notion sera développée dans la section 4.1.2. La majorité des théorèmes sur ces limites généralisées nécessite cependant des contraintes moins fortes sur les familles d'ensembles `F` et `G` que celles des voisinages. L'exemple le plus marquant à ce sujet est le théorème de composition `filterlim_comp`[1] pour lequel il n'y a aucune hypothèse sur les familles d'ensembles considérées.

1. `filterlim_comp` f g `F G H`:= Si `filterlim`$(f, \texttt{F}, \texttt{G})$ et `filterlim`$(g, \texttt{G}, \texttt{H})$, alors `filterlim`$(f \circ g, \texttt{F}, \texttt{H})$.

Limite des fonctions usuelles

Les théorèmes concernant les opérations arithmétiques, comme le théorème `is_lim_seq_plus` déjà cité, ont été démontrés en utilisant la continuité des opérations arithmétiques et la composition pour les fonctions à une et deux variables. Ainsi, les théorèmes `is_lim_seq_plus` pour les sommes de suites et `is_lim_plus` pour les sommes de fonctions ont été démontrés en utilisant le théorème

```
Lemma filterlim_Rbar_plus (x y z : Rbar) :
  is_Rbar_plus x y z ->
  filterlim (fun z => fst z + snd z)
    (filter_prod (Rbar_locally x) (Rbar_locally y)) (Rbar_locally z).
```

où `filter_prod` permet de construire la collection des voisinages du point `(x,y)` à partir de `Rbar_locally x` et `Rbar_locally y`. L'une des choses à noter est que, dans le cas fini, ce théorème a été démontré en utilisant une version beaucoup plus générale : le théorème `filterlim_plus` dit que l'addition dans un module normé est une fonction continue, c'est-à-dire que $\lim_{(u,v)\to(x,y)}((u,v)\mapsto u+v) = x+y$. Cette approche permet de factoriser les démonstrations en les ramenant à des problèmes de composition. Pour donner un exemple chiffré, le théorème `filterlim_Rbar_plus` a été démontré en environ 90 lignes et permet de démontrer les théorèmes `is_lim_seq_plus` et `is_lim_plus` en 3 lignes.

La formalisation des limites vers $+\infty$ et $-\infty$ a permis de compléter les théorèmes concernant les limites de fonctions élémentaires existant dans la bibliothèque standard. En fait, la seule limite infinie démontrée dans la bibliothèque standard est celle de la suite $(2^n)_{n\in\mathbb{N}}$. Les autres limites de fonctions élémentaires de la bibliothèque standard sont des conséquences de la continuité de ces fonctions et sont donc limitées au cas fini.

La généralisation de la notion de limite a permis de démontrer les limites infinies de plusieurs fonctions élémentaires comme la fonction exponentielle. En ce qui concerne les limites en 0 de fonctions non définies en ce point comme la fonction inverse et la fonction logarithme, la solution était au départ de "symétriser" la fonction afin de pouvoir travailler avec une fonction définie à droite et à gauche de 0. Par exemple, pour la limite du logarithme en 0, il fallait se contenter de la limite de la fonction $x \mapsto \ln|x|$. L'introduction du prédicat `filterlim`, ainsi que des familles de filtres `at_right` et `at_left`, a permis d'exprimer de façon plus simple ces limites. La limite du logarithme en 0 s'écrit maintenant :

```
Lemma is_lim_ln_0 : filterlim ln (at_right 0) (Rbar_locally m_infty).
```

Il est possible de simplifier l'usage des théorèmes faisant intervenir les filtres `at_right` et `at_left` en fournissant les versions "composées" dans lesquelles la limite et la positivité sont traitées séparément :

```
Lemma filterlim_ln_0 {U F} {FF : Filter F} (f : U -> R) :
  filterlim f F (locally 0) -> F (fun x => 0 < f x)
  -> filterlim (fun x => ln (f x)) F (Rbar_locally m_infty).
```

Cette version du théorème demande de démontrer séparément que la limite de la fonction `f` est 0 et que cette fonction est positive. Cette séparation permet de limiter le nombre de théorèmes à fournir dans la bibliothèque Coquelicot et ainsi faciliter la recherche des théorèmes à appliquer.

3.2.3 Séries et séries entières

Sommes partielles

Les séries sont usuellement définies comme étant la limite des sommes partielles :

$$\sum_{n\in\mathbb{N}} a_n = \lim_{n\to+\infty} \sum_{k=0}^{n} a_k.$$

Les sommes partielles `sum_f_R0` de la bibliothèque standard sont bien adaptées pour formaliser cette définition. La définition des sommes `sum_f` entre deux entiers ne se manipule malheureusement pas aussi facilement. Elle est formulée à partir de `sum_f_R0` par

```
Definition sum_f (s n : nat) (a : nat -> R) : R :=
  sum_f_R0 (fun x : nat => a (x + s)%nat) (n - s).
```

Dans le cas où `n < s`, cette fonction renvoie a_0 ce qui complique inutilement les preuves (contrairement à renvoyer une valeur comme 0)

Nous avons donc choisi de redéfinir les sommes partielles à partir d'une forme plus générale, inspirée de `bigop` de la bibliothèque `ssreflect` [BGOBP08]. Les sommes partielles ont été construites à partir de l'itérateur suivant :

```
Fixpoint iter {I T : Type} (op : T -> T -> T) (x0 : T)
  (l : list I) (f : I -> T) : T :=
  match l with
  |   []    => x0
  | h :: l' => op (f h) (iter l' f)
  end.
```

avec `op` une opération binaire associative et `x0` un élément neutre pour `op`. Ces propriétés d'associativité et de neutralité permettent de retrouver des résultats naturels comme
`a = iter op x0 (a :: [])  (fun x => x)` et `op a b = iter op x0 (a :: b :: [])  (fun x => x)`.

La forme très générale de cet opérateur permet d'avoir une base de théorèmes commune pour les sommes partielles et les produits indexés. `iter_nat` a été défini à partir de `iter` pour simplifier le cas particulier où la liste `l` est la suite des entiers de `n` à `m`. On remarque au passage que lorsque `m < n`, cet itérateur renvoie l'élément neutre.

Pour finir, la somme `sum_n_m` de `n` à `m` a été définie à partir de `iter_nat` et la somme partielle usuelle `sum_n` de 0 à `n` est un cas particulier de `sum_n_m`. Les théorèmes `sum_n_Reals` et `sum_n_m_Reals` montrent l'égalité entre les sommes partielles de la bibliothèque standard et celles de Coquelicot et rendent donc les deux bibliothèques compatibles.

Séries et séries entières

J'ai défini les séries et séries entières en me basant sur la définition des limites de suites et des sommes partielles ci-dessus :

```
Definition is_series (a : nat -> R) (l : R) :=
   filterlim (sum_n a) eventually (locally l).
Definition is_pseries (a : nat -> R) (x : R) (l : R) :=
  is_series (fun k => x ^ k * a k) l.
```

Comme ces deux notions sont des limites de suites particulières, les théorèmes concernant l'addition de suites et la multiplication d'une suite par un scalaire s'appliquent aux séries. Cependant, ils ne sont pas adaptés. Par exemple, pour l'addition de deux séries entières en utilisant les théorèmes sur les limites, il faut les 5 étapes suivantes :

$$
\begin{aligned}
\sum_{n\in\mathbb{N}}(a_n+b_n)x^n &= \lim_{n\to+\infty}\sum_{k=0}^{n}(a_k+b_k)\,x^k && \text{(définition)}\\
&= \lim_{n\to+\infty}\sum_{k=0}^{n}\left(a_kx^k+b_kx^k\right) && \text{(extensionnalité de lim et } \textstyle\sum\text{)}\\
&= \lim_{n\to+\infty}\left(\sum_{k=0}^{n}a_kx^k+\sum_{k=0}^{n}b_kx^k\right) && \text{(extensionnalité de lim)}\\
&= \lim_{n\to+\infty}\sum_{k=0}^{n}a_kx^k+\lim_{n\to+\infty}\sum_{k=0}^{n}b_kx^k && \text{(continuité de } +\text{)}\\
&= \sum_{n\in\mathbb{N}}a_nx^n+\sum_{n\in\mathbb{N}}b_nx^n && \text{(définition)}
\end{aligned}
$$

Ces 5 étapes, même si elles sont simples, sont cependant trop lourdes pour écrire des démonstrations faciles à relire. Les théorèmes sur l'opposé ou l'addition ont donc été adaptés pour les séries et les séries entières. Pour reprendre l'exemple de l'addition de deux séries :

```
Lemma is_series_plus (a b : nat -> R) (la lb : R) :
  is_series a la -> is_series b lb
    -> is_series (fun n => a n + b n) (la + lb).
```

Le produit de séries et de séries entières a également été démontré pour compléter les opérations arithmétiques usuelles sur les séries entières :

```
Lemma is_series_mult (a b : nat -> R) (la lb : R) :
  is_series a la -> is_series b lb
  -> ex_series (fun n => Rabs (a n)) -> ex_series (fun n => Rabs (b n))
  -> is_series (PS_mult a b) (la * lb).
```

où `PS_mult a b` est le produit de Cauchy des suites `a` et `b`. La somme partielle utilisée dans la définition de `PS_mult a b` est celle de la bibliothèque standard car ce théorème a été démontré avant que `sum_n` ne soit formalisé. Les hypothèses du théorème demandent que les deux séries soient absolument convergentes. Il existe une version plus générale de ce théorème (théorème de Cauchy-Mertens) supposant qu'une seule des deux séries est absolument convergente, mais elle n'a pas encore été formalisée dans la bibliothèque Coquelicot.

Critères de convergence

Les théorèmes précédents ne concernent que les séries et séries entières dont on peut déterminer la valeur. Or, comme pour les limites de suites, il est possible de démontrer la convergence sans exhiber de valeur. Cette méthode est d'ailleurs utilisée pour définir un certain nombre de fonctions usuelles comme la fonction exponentielle. J'ai donc démontré quelques théorèmes de convergence usuels comme le critère de Cauchy pour les séries et le critère de d'Alembert pour les séries et les séries entières.

Le critère des séries alternées n'a pas été démontré, mais j'ai démontré le théorème de sommation par partie qui en est une version plus générale :

```
Lemma partial_summation_R (a b : nat -> R) :
  (exists M, forall n, Rabs (sum_n b n) <= M)
  -> filterlim a eventually (locally 0)
  -> ex_series (fun n => Rabs (a (S n) - a n))
  -> ex_series (fun n => a n * b n).
```

Pour les séries entières, le rayon de convergence a été formalisé en utilisant les bornes supérieures définies sur $\overline{\mathbb{R}}$ (voir 3.5 pour plus de détails) :

$$\mathcal{C}_a = \sup \left\{ r \in \mathbb{R} \;\middle|\; \sum_{n\in\mathbb{N}} |a_n r^n| \text{ converge} \right\}. \tag{3.1}$$

Comme cette définition n'est pas toujours facile à utiliser, j'ai démontré que

$$\mathcal{C}_a = \sup \{ r \in \mathbb{R} \mid \exists M \in \mathbb{R},\ \forall n \in \mathbb{N},\ |a_n r^n| \leq M \}. \tag{3.2}$$

Les théorèmes permettant de trouver le rayon de convergence à partir du critère de d'Alembert ont également été démontrés. Afin de valider notre définition du rayon de convergence et de pouvoir l'utiliser dans des preuves, j'ai démontré d'une part que la série est absolument convergente à l'intérieur du cercle de convergence :

```
Lemma CV_disk_inside (a : nat -> R) (x : R) :
  Rbar_lt (Finite (Rabs x)) (CV_radius a)
    -> ex_series (fun n => Rabs (a n * x ^ n)).
```

et d'autre part que la série diverge grossièrement à l'extérieur. En d'autres termes, la suite $(a_n x^n)_{n\in\mathbb{N}}$ ne tend même pas vers 0 :

```
Lemma CV_disk_outside (a : nat -> R) (x : R) :
  Rbar_lt (CV_radius a) (Finite (Rabs x))
    -> not (is_lim_seq (fun n => a n * x ^ n) 0).
```

Afin de démontrer la convergence d'une série à partir du rayon de convergence, les théorèmes permettant de le majorer par construction ont été démontrés. Par exemple, pour l'addition :

```
Lemma CV_radius_plus (a b : nat -> R) :
  Rbar_le (Rbar_min (CV_radius a) (CV_radius b)) (CV_radius (PS_plus a b)).
```

où `PS_plus a b` est la somme des suites `a` et `b`.

Dérivées et intégrales de séries entières

Les résultats concernant la continuité, la dérivabilité et l'intégrabilité des séries entières ont été démontrés à partir de résultats plus généraux sur les suites et séries de fonctions en les regardant comme la suite de fonctions $f_n : x \mapsto \sum_{k=0}^{n} a_k x^k$ ou la série de terme général $g_n : x \mapsto a_n x^n$. Cette approche a permis de développer une base suffisamment générale pour plus tard étudier d'autres séries de fonctions usuelles comme les séries de Fourier.

Comme pour les opérations arithmétiques, le rayon de convergence des séries dérivées et intégrales a été relié au rayon de convergence de la série de départ.

3.2.4 Continuité dans $\mathbb{R}$

Dans la bibliothèque standard, la liste des théorèmes concernant la continuité des fonctions réelles est assez complète. Quelques théorèmes ont cependant pu être améliorés dans la bibliothèque Coquelicot.

Compacité

Tout d'abord, le théorème disant que toute fonction continue sur un intervalle $[a;b]$ est uniformément continue sur cet intervalle utilise l'axiome du tiers-exclu. Cette dépendance provient du théorème de compacité utilisé. Afin de ne pas dépendre d'axiomes autres que ceux définissant les nombres réels, de nouveaux théorèmes concernant la compacité ont été prouvés.

Une autre motivation pour de nouveaux théorèmes de compacité était la démonstration de théorèmes sur les intégrales à paramètres. Ces théorèmes faisant appel à de la continuité uniforme sur des pavés $[a;b] \times [c;d]$, nous avons eu besoin de théorèmes de compacité sur $\mathbb{R}^2$.

Nous avons donc fait le choix de démontrer ces théorèmes sur $\mathbb{R}^n$ pour plus de généralité : étant donnés un pavé $[\mathbf{a};\mathbf{b}] = [a_1;b_1] \times \cdots \times [a_n;b_n]$ et une jauge $\delta : \mathbb{R}^n \to \mathbb{R}_+^*$, le lemme `compactness_value` nous donne l'existence (dans `Set`) d'une valeur $\delta' > 0$ telle que

$$\forall x \in [\mathbf{a};\mathbf{b}],\ \neg\neg\exists t \in [\mathbf{a};\mathbf{b}],\ |x-t| < \delta(t) \wedge \delta' \leq \delta(t).$$

L'axiome de tiers-exclu a pu être évité en introduisant une double négation devant $\exists t$. En effet, alors qu'il est possible de déterminer explicitement la valeur de δ' en utilisant une borne supérieure, il n'y a pas de moyen en Coq d'extraire la liste de points permettant de définir la couverture finie. Dans la pratique, cette double négation n'est pas un problème car ce théorème est utilisé dans des preuves par contradiction.

Théorème des valeurs intermédiaires

Un autre théorème disposant d'une nouvelle version dans la bibliothèque Coquelicot est le théorème des valeurs intermédiaires. En effet, l'ajout des réels étendus et des limites généralisées a permis d'étendre ce théorème au cas où les différentes bornes considérées sont des réels étendus :

```
Lemma IVT_Rbar_incr (f : R -> R) (a b la lb : Rbar) (y : R) :
  is_lim f a la -> is_lim f b lb
  -> (forall x : R, Rbar_lt a x -> Rbar_lt x b -> continuity_pt f x)
  -> Rbar_lt a b
  -> Rbar_lt la y /\ Rbar_lt y lb
  -> {x : R | Rbar_lt a x /\ Rbar_lt x b /\ f x = y}.
```

Même s'il est plus général que celui de la bibliothèque standard, il est encore possible de l'améliorer. En effet, pour l'appliquer à des fonctions qui ne sont pas totales, par exemple la fonction ln entre 0 et un autre réel étendu strictement positif, il est nécessaire de passer par une fonction auxiliaire ($x \mapsto \ln|x|$ dans l'exemple) afin de pouvoir calculer les limites `is_lim f a la` et `is_lim f b lb`. La formalisation des limites à droite et à gauche permettrait maintenant de généraliser ce théorème.

3.3 L'intégrale de Riemann

La construction de l'intégrale de Riemann présentée dans la section 2.1.3 est certes usuelle, mais elle n'est pas facile à manipuler en Coq. En effet, l'absence de prédicat liant l'existence de l'intégrale à sa valeur oblige à dupliquer les démonstrations pour les propriétés portant à la fois sur l'intégrabilité et sur la valeur de l'intégrale. Pour démontrer une nouvelle propriété, les mêmes opérations sur les fonctions en escalier doivent être faites d'une part pour démontrer l'intégrabilité et d'autre part pour démontrer la valeur de l'intégrale.

La recherche d'une construction semblable à celle présentée dans la section 3.1, et en particulier la construction d'une fonction totale pour écrire les intégrales de Riemann, m'a conduite à prouver l'équivalence de l'intégrale telle qu'elle est définie dans la bibliothèque standard avec la définition suivante :

$$\forall \varepsilon > 0,\ \exists \delta > 0,\ \forall (\sigma,\xi) \text{ subdivision pointée}^{\,2} \text{ entre } a \text{ et } b \text{ telles que } \max|\sigma_{i+1}-\sigma_i| < \delta,$$
$$\left|\textstyle\sum_i (\sigma_{i+1}-\sigma_i) f(\xi_i) - \ell\right| < \varepsilon.$$

2. $\forall i,\ \sigma_i \leq \xi_i \leq \sigma_{i+1}$

Pour cela, j'ai commencé par définir une nouvelle représentation des fonctions en escalier, présentée dans la section 3.3.1, car la représentation disponible dans la bibliothèque standard s'est avérée difficile à manipuler. La définition choisie pour l'intégrale étant proche de la définition d'une limite, cela a permis d'utiliser les travaux précédents sur les limites pour formaliser l'intégrale de Riemann de la bibliothèque Coquelicot qui est présentée dans la section 3.3.2. Pour finir, l'équivalence entre cette nouvelle définition et l'intégrale définie dans la bibliothèque standard a été démontrée.

3.3.1 Les fonctions en escalier généralisées

En travaillant sur les sommes de Riemann, je me suis aperçue rapidement que pour un certain nombre de propriétés, seule la liste des réels représentant la subdivision de l'intervalle d'intégration $(\sigma_i)_{i\leq n+1}$ et la liste des valeurs sur chacun des sous-intervalles ainsi décrits $(\xi_i)_{i\leq n}$ étaient utiles. J'ai donc défini une nouvelle structure, nommée `SF_seq`[3], afin de manipuler plus facilement ces couples de listes particuliers :

```
Record SF_seq {T : Type} := mkSF_seq {SF_h : R ; SF_t : seq (R * T)}.
```

avec `SF_h` $= \sigma_0$ et `SF_t` $= ((\sigma_{i+1}, \xi_i))_{i\leq n}$. Des fonctions `SF_lx` et `SF_ly` ont été définies afin de récupérer ces listes à partir d'une `SF_seq`. J'ai également définie la fonction réciproque `SF_make` permettant de construire une `SF_seq` à partir de deux listes, la première ayant un élément de plus que la seconde. Ces différentes fonctions ont permis, avec quelques autres, de faire la passage entre les fonctions en escalier `StepFun` de la bibliothèque standard et les `SF_seq`.

J'ai défini ces nouvelles fonctions en escalier avec la suite $(\xi_i)_{i\leq n}$ à valeurs dans un type quelconque afin de prendre plus facilement en compte les fonctions en escalier sur les réels étendus. Elles sont en effet utiles pour démontrer l'équivalence entre la nouvelle définition de l'intégrale et celle de la bibliothèque standard. Pour faciliter le raisonnement par récurrence, j'ai défini les fonctions `SF_nil : R` $\to$ `SF_seq` et `SF_cons : R * T` $\to$ `SF_seq` $\to$ `SF_seq` permettant de construire des `SF_seq` de la même façon que des listes. Le lemme

```
Lemma SF_cons_ind {T : Type} (P : SF_seq -> Type) :
  (forall x0 : R, P (SF_nil x0))
    -> (forall (h : R * T) (s : SF_seq), P s -> P (SF_cons h s))
      -> (forall s, P s).
```

a alors permis d'effectuer des raisonnements par récurrence sur les `SF_seq` semblables à ceux faits sur les listes. Un certain nombre d'autres fonctions et lemmes ont également été formalisés afin de simplifier l'usage des `SF_seq` en le rapprochant de celui des listes usuelles. Pour plus de clarté, j'exprimerai les théorèmes suivant en utilisant la notation $ptd = ((\sigma_i)_{i\leq n+1}, (\xi_i)_{i\leq n})$ pour désigner les subdivisions pointées.

Au-delà de cette utilisation initiale pour représenter les fonctions en escalier, les `SF_seq` servent le plus souvent pour représenter des subdivisions pointées. En effet, il s'agit d'une paire de deux listes σ et ξ de nombres réels, l'une ayant un élément de plus que l'autre, triées dans le sens "$\forall i,\ \sigma_i \leq \xi_i \leq \sigma_{i+1}$". C'est avec cette signification qu'elles sont utilisées pour définir les sommes de Riemann :

Definition `Riemann_sum (f : R -> T) (ptd : SF_seq) : T :=` $\sum_{i=0}^{n}(\sigma_{i+1} - \sigma_i)\cdot f(\xi_i)$.

En Coq, la liste des $((\sigma_{i+1} - \sigma_i)\cdot f(\xi_i))_{i\leq n}$ a été construite avec la fonction `pairmap` et la somme des éléments de cette liste avec la fonction `foldr`. L'usage de ces fonctions de la bibliothèque `ssreflect` plutôt qu'une définition ad-hoc permet de disposer des théorèmes déjà démontrés dans la bibliothèque des listes de `ssreflect`. L'usage de `foldr` étant cependant laborieux, de nombreux théorèmes de réécriture sur les `Riemann_sum` ont été démontrés pour que l'utilisateur n'ait pas à remonter jusqu'à la définition.

Parmi les théorèmes démontrés à propos des sommes de Riemann, on peut remarquer la relation de Chasles. Elle a été démontrée sous deux versions en raison de l'évolution des besoins. La plus ancienne, nommée `SF_Chasles`, utilise un découpage simple des `SF_seq` :

$$\begin{aligned} &\forall f,\ \forall ptd = ((\sigma_i)_{i\leq n+1}, (\xi_i)_{i\leq n}),\ \forall x \in [\sigma_0; \sigma_{n+1}], \\ \texttt{Riemann\_sum}(f, ptd) \quad &= \quad \texttt{Riemann\_sum}(f, \texttt{SF\_cut\_down'}(ptd, x)) \\ &\qquad + \texttt{Riemann\_sum}(f, \texttt{SF\_cut\_up'}(ptd, x)) \end{aligned}$$

Les deux fonctions `SF_cut_down'` et `SF_cut_up'` permettant de découper une `SF_seq` en trouvant le premier indice j tel que $x < \sigma_{j+1}$. La première renvoie `SF_rcons`$(((\sigma_i)_{i\leq j}, (\xi_i)_{i\leq j-1}); (x, \xi_j))$ et la seconde renvoie la `SF_seq` ayant pour valeur de tête x et pour suite $((\sigma_{i+1}, \xi_i))_{j\leq i\leq n}$.

3. le préfixe `SF` étant une abréviation de Step Function

Le théorème `SF_Chasles` a été utilisé pour démontrer que l'intégrale de la bibliothèque standard et celle de la bibliothèque Coquelicot sont les mêmes. Dans ce lemme, on retrouve l'égalité usuelle de la relation de Chasles, mais le découpage des `SF_seq` choisi fait perdre le fait que les nouvelle subdivisions sont des subdivisions pointées. Cette propriété n'étant pas utile dans la démonstration d'équivalence, cette version plus simple était parfaitement adaptée. Cela n'est cependant pas suffisant pour démontrer la relation de Chasles pour les intégrales. J'ai donc été amenée à concevoir un découpage plus complexe des `SF_seq` qui conserve les propriétés sur les subdivisions pointées, mais qui introduit une erreur bornée :

$$\begin{aligned}&\forall f,\ \forall ptd = ((\sigma_i)_{i\leq n+1}, (\xi_i)_{i\leq n}),\ \forall M \in \mathbb{R},\ \forall x \in [\sigma_0, \sigma_{n+1}],\ \forall \varepsilon > 0,\\ &(\forall t \in [\sigma_0, \sigma_{n+1}], |f(t)| < M) \wedge (\forall i \leq n,\ \sigma_i \leq \xi_i \leq \sigma_{i+1}) \wedge |\sigma| < \varepsilon\\ \Rightarrow |(&\texttt{Riemann\_sum}(f, \texttt{SF\_cut\_down}(ptd, x)) + \texttt{Riemann\_sum}(f, \texttt{SF\_cut\_up}(ptd, x)))\\ &\qquad -\texttt{Riemann\_sum}(f, ptd)| \quad < \quad 2\varepsilon M.\end{aligned}$$

Les fonctions `SF_cut_down` et `SF_cut_up` sont assez proches de celles définies précédemment, à la différence que la valeur intermédiaire ξ_j est remplacée par $\min(x, \xi_j)$ dans `SF_cut_down` et par $\max(x, \xi_j)$ dans `SF_cut_up` afin de conserver la propriété "$\forall i \leq n,\ \sigma_i \leq \xi_i \leq \sigma_{i+1}$". La notation $|\sigma|$ représente le pas de la suite et vaut $\max_i |\sigma_{i+1} - \sigma_i|$. Dans ce théorème, l'erreur provient du fait que l'une des valeurs $f(\xi_j)$ est remplacée par $f(x)$ et ne se compense donc plus.

En d'autres termes, si la fonction `f` est bornée sur l'intervalle considéré (ce qui est le cas pour les fonctions intégrables au sens de Riemann), alors la différence entre les sommes de Riemann obtenues par ce second découpage et la somme de Riemann de départ peut être rendue aussi petite que souhaitée en choisissant une subdivision pointée avec un pas ε suffisamment petit.

3.3.2 L'intégrale de Riemann vue comme une limite

En plus de lier la valeur de l'intégrale à un critère d'intégrabilité, la définition choisie pour l'intégrale possède également une forme proche de celle des limites dans la section 3.2.2 :

$$\forall \varepsilon > 0,\ \exists \delta > 0,\ \forall (\sigma, \xi) \text{ subdivision pointée entre } a \text{ et } b \text{ de pas} < \delta,\ |\textstyle\sum(\sigma_{i+1} - \sigma_i) f(\xi_i) - \ell| < \varepsilon.$$

Le critère usuel "$|x - a| < \delta$" est en effet remplacé ici par une borne sur le pas des subdivisions pointées. Partant de cette constatation, il est possible de définir l'intégrale de Riemann en utilisant `filterlim`

```
Definition is_RInt (f : R -> R) (a b : R) (If : R) :=
  filterlim (fun (ptd : SF_seq) => sign (b - a) * Riemann_sum f ptd)
    (Riemann_fine a b) (locally If).
```

avec `Riemann_fine a b` un filtre sur les subdivisions `SF_seq` qui sera détaillé dans la section 4.1.2.

Cela autorise la réutilisation de tous les théorèmes sur `filterlim` comme l'addition, l'opposé, la positivité et bien d'autres.

3.3.3 Compatibilité avec la bibliothèque standard

Comme mentionné dans la section 3.1, toutes les définitions de la bibliothèque standard disposant d'un équivalent dans la bibliothèque Coquelicot ont été démontrées équivalentes aux nouvelles définitions. J'ai ainsi démontré l'équivalence entre la nouvelle version de l'intégrale de Riemann `is_RInt` et celle de la bibliothèque standard donnée par `Riemann_integrable` et `RiemannInt`. Pour cela, j'ai utilisé les trois définitions usuelles de cette intégrale :

1. la convergence des sommes de Riemann, choisie pour `is_RInt`,
2. l'existence de fonctions en escalier arbitrairement proches, choisie dans la bibliothèque standard pour `Riemann_integrable`, et
3. la convergence des sommes de Darboux $\sum_{i=0}^{n}(x_{i+1}-x_i)\sup_{[x_i;x_{i+1}]} f$ et $\sum_{i=0}^{n}(x_{i+1}-x_i)\inf_{[x_i;x_{i+1}]} f$ vers une limite commune.

Même si seules les définitions 1 et 2 apparaissent dans les énoncés des lemmes démontrés, les sommes de Darboux sont utilisées pour exhiber des fonctions en escalier arbitrairement proches de la définition 2.

La première étape a été de démontrer que, si une fonction est intégrable au sens de la bibliothèque standard, alors elle est intégrable au sens de la bibliothèque Coquelicot. Qui plus est, la valeur est celle de la bibliothèque standard donnée par `RiemannInt` :

```
Lemma ex_RInt_Reals_aux_1 (f : R -> R) (a b : R) :
  forall pr : Riemann_integrable f a b, is_RInt f a b (RiemannInt pr).
```

Ce lemme permet de déduire immédiatement les deux corollaires suivants donnant l'un des sens de l'équivalence entre les deux intégrales ainsi que l'égalité des valeurs.

```
Lemma ex_RInt_Reals_1 (f : R -> R) (a b : R) :
  Riemann_integrable f a b -> ex_RInt f a b.
Lemma RInt_Reals (f : R -> R) (a b : R) :
  forall pr : Riemann_integrable f a b, RInt f a b = RiemannInt pr.
```

La réciproque a été démontrée en passant par la convergence des sommes de Darboux :

```
Lemma ex_RInt_Reals_0 (f : R -> R) (a b : R) :
  ex_RInt f a b -> Riemann_integrable f a b.
```

Ces théorèmes ont permis de démontrer les théorèmes usuels (linéarité, Chasles, positivité, ...) sur l'intégrale de Riemann à partir des théorèmes de la bibliothèque standard dans un premier temps. Ces démonstrations ont ensuite été refaites en utilisant uniquement les ressources de la bibliothèque Coquelicot. Le but initial était de supprimer la dépendance à l'axiome du tiers-exclu apparaissant dans certaines démonstrations de la bibliothèque standard afin de ne dépendre que des axiomes définissant l'ensemble des nombres réels `R`. La généralisation de la bibliothèque qui sera présentée dans le chapitre 4 a été vue comme une autre motivation puisque les théorèmes de la bibliothèque standard ne s'appliquaient alors plus.

3.4 L'analyse dans $\mathbb{R}^2$

Avant la généralisation de la bibliothèque présentée dans le chapitre 4, j'avais formalisé des définitions et théorèmes permettant de travailler avec les fonctions à deux variables Ce développement comprend des résultats sur la différentiabilité des intégrales à paramètre, les dérivées partielles, ainsi que la continuité et la différentiabilité des fonctions à deux variables. Bien que le but était de vérifier la formule de d'Alembert (section 5.3), de nombreux résultats, comme la continuité et la différentiabilité des fonctions arithmétiques, ont été démontrés afin d'en permettre l'utilisation pour la démonstrations de théorèmes d'analyse dans $\mathbb{R}^2$. Les théorèmes démontrés dans la version préliminaire développée pour vérifier la formule de d'Alembert ne sont pas disponibles dans la bibliothèque Coquelicot en raison de la généralisation de la bibliothèque qui permet en particulier d'étudier la différentiabilité de fonctions de $\mathbb{R}^n$ dans $\mathbb{R}^m$. Ils sont cependant disponibles sur ma page web `www.lri.fr/~lelay/`.

Continuité La continuité dans $\mathbb{R}^2$ a été définie de la façon suivante :

```
Definition continuity_2d_pt f x y :=
  forall eps : posreal, exists delta : posreal, forall u v : R,
    Rabs (u - x) < delta -> Rabs (v - y) < delta -> Rabs (f u v - f x y) < eps.
```

Pour la topologie sur l'espace de départ, plutôt que d'utiliser la norme euclidienne usuelle $(x, y) \mapsto \sqrt{x^2 + y^2}$, j'ai choisi d'utiliser la norme infinie $(x, y) \mapsto \max(|x|, |y|)$ qui est équivalente à la norme euclidienne [4] et plus facile à utiliser en Coq.

La continuité uniforme a également été formalisée dans $\mathbb{R}^2$ afin de démontrer les théorèmes sur les intégrales à paramètre. En effet, dans ces théorèmes, l'hypothèse de continuité uniforme est utilisée pour déterminer des bornes valides sur tout l'intervalle d'intégration.

Différentiabilité et dérivées partielles Comme pour la continuité, la différentiabilité a été définie en utilisant la norme infinie :

```
Definition differentiable_pt_lim (f : R -> R -> R) (x y : R) (lx ly : R) :=
  forall eps : posreal, exists delta : posreal, forall u v : R,
    Rabs (u - x) < delta -> Rabs (v - y) < delta ->
      Rabs (f u v - f x y - (lx * (u - x) + ly * (v - y))) <=
        eps * Rmax (Rabs (u - x)) (Rabs (v - y)).
```

J'ai choisi cette forme proche de $f(y) - f(x) - L_x(y - x) = o(y - x)$ afin d'éviter l'hypothèse $x \neq y$ qui peut être difficile à manipuler, en particulier dans la démonstration du théorème de composition $(f \circ g)' = g' \cdot (f' \circ g)$ où il faut distinguer les cas $g(x) = g(y)$ et $g(x) \neq g(y)$. L'équivalence avec

4. **Lemma** `sqrt_plus_sqr` : $\forall x, y \in \mathbb{R},\ \max(|x|, |y|) \leq \sqrt{x^2 + y^2} \leq \sqrt{2} \cdot \max(|x|, |y|)$

la dérivabilité dans le cas où la fonction `f` est constante par rapport à l'une des deux variables a été démontrée.

La différentiabilité de fonctions à deux variables a connu deux versions au cours de ma thèse. La première, héritée de mon stage de M2, prenait un seul paramètre `l : R * R`, mais elle a été changée pour la version ci-dessus plus facile à utiliser. De nombreux théorèmes de différentiabilité ont été démontrés pour la première version, en particulier concernant les sommes et produits de fonctions différentiables. Pour la seconde version, ceux-ci peuvent être déduits de la généralisation de la dérivabilité.

Un théorème donnant la différentiabilité à partir de l'existence des dérivées partielles a été démontré pour chacune des deux versions de la différentiabilité. La première version s'appuyait sur la bibliothèque standard pour la définition des dérivées partielles. La manipulation d'hypothèses locales était donc plus difficile en raison des termes de preuve. J'ai donc démontré que si les dérivées partielles f_1' et f_2' existent en tout point de $\mathbb{R}^2$ et sont continues, alors la fonction est différentiable en tout point et sa différentielle est $Df = (f_1', f_2')$. Pour la seconde version, les hypothèses s'expriment avec la fonction totale `Derive : (R -> R) -> R -> R` pour écrire la dérivée d'une fonction (voir section 3.5 pour les détails). L'absense de terme de preuve permet d'écrire plus facilement la conclusion, ce qui permet de formaliser le théorème avec des hypothèses locales :

Théorème. *Soient $f : \mathbb{R}^2 \to \mathbb{R}$ et $(x, y) \in \mathbb{R}^2$. Si*
- *la dérivée f_1' de f par rapport à sa première variable existe au voisinage de (x, y) et est continue en (x, y), et*
- *f est dérivable par rapport à sa deuxième variable en (x, y), de dérivée f_2',*

alors f est différentiable en (x, y) et $Df(x, y) = (f_1'(x, y), f_2'(x, y))$.

Il y a un point à remarquer dans ce théorème : contrairement à l'énoncé usuel, seule la première dérivée partielle est continue. Cette hypothèse de dérivabilité locale est représentée par

```
locally_2d (fun u v : R => ex_derive (fun z : R => f z v) u) x y,
```

où `locally_2d` est le prédicat permettant d'exprimer que les propriétés à deux variables réelles sont localement vérifiées.

Intégrales à paramètres Les éléments d'analyse formalisés dans $\mathbb{R}^2$ ont abouti au théorème de dérivation sous le signe intégral :

Théorème. *Soient $f : \mathbb{R}^2 \to \mathbb{R}$, $(a, b) \in \mathbb{R}^2$ et $x \in \mathbb{R}$. Si*
- *pour tout $t \in [a, b]$, la fonction f est dérivable par rapport à sa première variable au voisinage de (x, t),*
- *pour tout $t \in [a, b]$, la dérivée partielle par rapport à la première variable f_1' est continue en (x, t),*
- *au voisinage de x, la fonction f est intégrable entre a et b par rapport à sa seconde variable,*

alors la fonction $y \mapsto \int_a^b f(x, t)\,dt$ est dérivable en x et admet pour dérivée $\int_a^b f_1'(x, t)\,dt$.

La première version de ce théorème utilisait les intégrales de la bibliothèque standard. Les hypothèses de ce théorème avaient été formulées de façon globale (dérivabilité sur $\mathbb{R}$ tout entier et intégrabilité entre a et b pour tout $x \in \mathbb{R}$) afin de simplifier l'expression de la conclusion. Dans la seconde version, j'ai énoncé ce théorème en utilisant les fonctions totales `Derive` et `RInt` déjà évoquées (voir section 3.5). Les hypothèses de dérivabilité et d'intégrabilité n'intervenant plus dans l'écriture de ces objets, cela a permis d'exprimer les hypothèses locales de ce théorème.

Afin d'achever l'étude des intégrales à paramètre, le théorème

$$\frac{\partial}{\partial x}\left(\int_{a(x)}^{b(x)} f(x,t)\,dt\right) = \int_{a(x)}^{b(x)} \frac{\partial f}{\partial x}(x,t)\,dt - f(x,a(x))a'(x) + f(x,b(x))b'(x)$$

a été démontré pour gérer le cas où les bornes de l'intégrale ne sont pas constantes.

3.5 Fonctions totales

Comme cela a déjà été remarqué, les fonctions de la bibliothèque standard permettant d'écrire les dérivées et les intégrales sont difficiles à utiliser à cause des types dépendants. Suite aux travaux sur la formule de d'Alembert, nous avons donc cherché à réduire l'usage des types dépendants. Cela a abouti à la conception de fonctions totales permettant d'écrire et de manipuler plus facilement les dérivées et

les intégrales déjà présentes dans la bibliothèque standard, mais également les limites de suites et de fonctions.

L'une des remarques importantes pour cette partie est que chacune des notions traitées par la bibliothèque Coquelicot peut être approchée par une suite de réels bien choisie. Par exemple pour les intégrales, si une fonction f est intégrable entre a et b, alors

$$\int_a^b f(t)\,dt = \lim_{n\to+\infty} \sum_{k=0}^{n} (x_{k+1}^n - x_k^n) \cdot f\left(\frac{x_k^n + x_{k+1}^n}{2}\right), \quad \text{avec } x_k^n = \frac{(n+1-k)\cdot a + k\cdot b}{n+1}.$$

Il suffit donc de construire une fonction totale sur les suites de réels qui a pour valeur la limite de la suite en cas de convergence pour en déduire toutes les fonctions totales qui nous intéressent ici.

Les limites supérieure et inférieure d'une suite sont des fonctions totales : quelque soit la suite u, il est possible de déterminer sa limite supérieure $\overline{\lim}\ u$ et sa limite inférieure $\underline{\lim}\ u$ dans $\overline{\mathbb{R}}$. Elles offrent également l'avantage d'être égales à la limite dans le cas des suites convergentes. J'ai donc choisi de formaliser ces deux limites afin de les utiliser pour définir une fonction totale représentant les limites de suites convergentes ou non.

3.5.1 Principe d'Omniscience Limité

En mathématiques classiques, c'est-à-dire utilisant l'axiome du tiers-exclu et l'axiome du choix, les bornes supérieures et inférieures d'ensembles, de suites et de fonctions sur les réels sont toujours définies. Le choix fait dans la bibliothèque Coquelicot de ne pas utiliser d'axiomes autres que ceux des réels nous a donc conduits à construire les bornes sans utiliser les démonstrations usuelles. Les axiomes des réels nous permettent de démontrer le principe d'omniscience limité qui étend la décidabilité aux quantificateurs pour les prédicats satisfaisant le tiers-exclu sur les entiers naturels :

```
Lemma LPO : forall P : nat -> Prop, (forall n : nat, P n \/ not (P n)) ->
  {n : nat| P n} + {forall n : nat, not (P n)}.
```

Ce lemme fournit pour toute propriété vérifiant le tiers-exclu sur les entiers soit un entier la vérifiant, soit une preuve qu'elle est fausse. Comme les axiomes des réels fournissent la décidabilité sur l'ordre, ce théorème est suffisant pour déterminer si une suite est bornée ou non (voir ci-dessous).

La première étape de cette démonstration est la construction de la suite réelle

$$u_n = \begin{cases} 1/(n+1) & \text{si } \texttt{P n} \text{ est vrai} \\ 0 & \text{sinon} \end{cases}$$

Les valeurs prises par cette suite étant comprises entre 0 et 1, il est possible d'utiliser l'axiome `completeness` pour en déterminer la borne supérieure. Cette borne étant elle aussi comprise entre 0 et 1, l'axiome `total_order_T` permet alors de décider si elle est égale à 0 ou strictement positive. Dans la premier cas, le prédicat n'est jamais satisfait et dans le second, l'axiome `archimed` ainsi que l'ordre total permettent de démontrer que cette borne supérieure est de la forme $1/(n+1)$ avec n un entier vérifiant le prédicat `P`.

3.5.2 Des fonctions totales pour les suites

Le principe d'omniscience limité permettant de décider de la validité d'un prédicat sur les entiers et d'obtenir des témoins, j'ai ramené le problème de l'existence d'une borne supérieure sur un ensemble de nombres réels à l'existence d'un majorant pour des suites bien choisies. Cela m'a conduite à démontrer un lemme traitant spécifiquement cette question :

```
Lemma LPO_ub_dec : forall (u : nat -> R),
  {M : R | forall n, u n <= M} + {forall M : R, exists n, M < u n}.
```

Ce lemme se démontre en appliquant plusieurs fois le principe d'omniscience limité. Une première application à l'ensemble $E = \{M \in \mathbb{N} \mid \forall n \in \mathbb{N},\ u_n \leq M\}$ permet de déterminer, s'il existe, un majorant entier à la suite. Dans le cas où il n'existe pas de majorant, une seconde application à l'ensemble $E_M = \{\ n \in \mathbb{N} \mid M < u_n\ \}$ permet de déterminer des contre-exemples. Pour chacun de ces ensembles, la décidabilité pour chaque entier découle de la décidabilité de l'ordre dans l'ensemble des réels.

Bornes d'un sous-ensemble de $\mathbb{R}$

J'ai commencé par définir la notion de borne supérieure à valeur dans $\overline{\mathbb{R}}$ sur le même modèle que celui de la bibliothèque standard en remplaçant l'inégalité sur les réels `Rle` par l'inégalité sur les réels étendus `Rbar_le` :

```
Definition is_ub_Rbar (E : R -> Prop) (l : Rbar) :=
  forall x : R, E x -> Rbar_le x l.
Definition is_lub_Rbar (E : R -> Prop) (l : Rbar) :=
  is_ub_Rbar E l /\ (forall b : Rbar, is_ub_Rbar E b -> Rbar_le l b).
```

La démonstration de l'existence d'une borne supérieure s'est déroulée et deux temps : dans un premier temps j'ai démontré que la propriété "être majoré" est décidable, ce qui permet de renvoyer $+\infty$ pour les ensembles non majorés, et dans un deuxième temps j'ai démontré que, soit tous les nombres réels sont des majorants (dans ce cas l'ensemble est vide et la borne supérieure est $-\infty$), soit il existe un plus petit majorant qui est la borne supérieure. Pour chacune de ces étapes, je me suis ramenée à des suites de réels bien choisies pour appliquer soit le principe d'omniscience limité, soit son corollaire `LPO_ub_dec` présenté ci-dessus.

Pour décider si un ensemble est majoré ou non, la construction de la suite appropriée est assez simple : il s'agit des bornes supérieures des ensembles $E_n = (E \cap]-\infty\ ;\ n]) \cup \{0\}$. Ces ensembles étant majorés par n et contenant 0, l'axiome de la borne supérieure permet de construire une suite $u_n = \sup E_n$. Le lemme `LPO_ub_dec` permet alors de démontrer le lemme suivant concernant l'existence d'un majorant :

```
Lemma is_ub_Rbar_dec (E : R -> Prop) :
  {l : R | is_ub_Rbar E l} + { forall l : R, not (is_ub_Rbar E l) }.
```

Il est alors possible de décider si l'ensemble `E` est majoré ou non par un réel. Dans le second cas, la preuve que la borne supérieure est $+\infty$ se fait en quelques lignes.

Dans le premier cas, il est établi que l'ensemble est majoré. Avec cette hypothèse, dans le cas où l'ensemble n'est pas vide, la borne supérieure est celle fournie par l'axiome `completeness` de la bibliothèque standard. Il reste donc à déterminer si l'ensemble est vide ou non. On choisit des ensembles non vides permettant de déterminer cette propriété, ce n'est cependant pas aussi simple que pour décider de l'existence d'un majorant. Jusqu'à la bibliothèque Coquelicot 2.0, je n'étais pas parvenue à me passer de l'hypothèse "ensemble non vide". Je suis parvenue à me passer de cette hypothèse en considérant la suite v_n des bornes supérieures des ensembles $F_n = E \cup \{-n\}$. Comme pour la suite u étudiée précédemment, elle est définie à l'aide de l'axiome `completeness` car les ensembles considérés sont non vides (ils contiennent $-n$) et sont majorés par tout majorant de `E` plus grand que $-n$. Le principe d'omniscience limité appliqué à l'ensemble $\{n \in \mathbb{N} \mid v_n \neq -n\}$ permet alors de distinguer deux cas : soit il existe un indice n tel que $v_n \neq -n$ et dans ce cas v_n est la borne supérieure de `E`, soit pour tout n, $v_n = -n$ ce qui implique que tout nombre réel est un majorant de l'ensemble `E` qui est donc vide. La borne supérieure alors est égale à $-\infty$.

Le lemme d'existence d'une borne supérieure nous permet alors de construire une première fonction totale qui calcule la borne supérieure d'un ensemble de nombres réels :

```
Lemma ex_lub_Rbar (E : R -> Prop) : {l : Rbar | is_lub_Rbar E l}.
Definition Lub_Rbar (E : R -> Prop) := projT1 (ex_lub_Rbar E).
```

La borne inférieure d'un ensemble a été définie de façon semblable à la borne supérieure :

```
Definition is_lb_Rbar (E : R -> Prop) (l : Rbar) :=
  forall (x : R), E x -> Rbar_le l x.
Definition is_glb_Rbar (E : R -> Prop) (l : Rbar) :=
  is_lb_Rbar E l /\ (forall b, is_lb_Rbar E b -> Rbar_le b l).
```

L'existence de la borne inférieure, ainsi que l'analogue des autres lemmes démontrés à propos de la borne supérieure, ont été démontrés en quelques lignes en utilisant les lemmes permettant de passer de la borne supérieure à la borne inférieure par l'opposé. En effet, il suffit de passer de E à $-E = \{x \in \mathbb{R} \mid -x \in E\}$ pour passer des propriétés sur les majorants à celles sur les minorants et des propriétés sur les bornes supérieures aux propriétés sur les bornes inférieures.

Bornes d'un sous-ensemble de $\overline{\mathbb{R}}$

Pour l'existence d'une borne supérieure d'un sous-ensemble de $\overline{\mathbb{R}}$ (et non plus de $\mathbb{R}$), les raisonnements précédents ne sont pas adaptés car ils reposent sur le fait que $\mathbb{R}$ est archimédien. En effet, la démonstration

du lemme `is_ub_Rbar_dec` repose sur le fait que tout élément de l'ensemble `E` appartient à l'un des ensembles $(E \cap]-\infty; n]) \cup \{0\}$ et sera donc plus petit qu'un des entiers n. Dans le cas des sous-ensembles de $\overline{\mathbb{R}}$, cette propriété n'est pas vérifiée par le point $+\infty$. Le principal obstacle pour démontrer l'existence d'une borne supérieure sur des sous-ensembles de $\overline{\mathbb{R}}$ est donc de déterminer si le point $+\infty$ appartient ou non à un ensemble.

Le point fondamental de cette démonstration est le lemme de la bibliothèque Coquelicot

```
Lemma EM_dec : forall P : Prop, {not (not P)} + {not P}.
```

Ce lemme établit, à partir des axiomes de la bibliothèque standard, que toute propriété est soit non contradictoire, soit fausse. Il est démontré en considérant l'ensemble $\{x \in \mathbb{R} | x = 0 \vee (x = 1 \wedge \mathtt{P})\}$. Cet ensemble est non vide (il contient 0) et il est majoré par 1. Il est donc possible d'utiliser l'axiome `completeness` pour obtenir sa borne supérieure. En comparant cette borne supérieure à 1, on démontre par l'absurde le résultat.

Cette propriété permet de démontrer l'existence de la borne supérieure pour tout sous-ensemble de $\overline{\mathbb{R}}$.

```
Lemma Rbar_ex_lub (E : Rbar -> Prop) : {l : Rbar | Rbar_is_lub E l}.
```

En utilisant le lemme `EM_dec` sur "$+\infty \in E$", nous avons démontré que, dans le cas où $+\infty$ n'appartient pas à E, cela revient au même de considérer la borne supérieure de l'ensemble de réels $E \cap \mathbb{R}$, et dans l'autre cas on aboutit à une contradiction si $+\infty$ n'est pas la borne supérieure.

Bornes d'une suite et limite supérieure

L'une des caractérisations des limites supérieure et inférieure utilise les extrema de suites :

$$\overline{\lim}\, u = \inf_{n\in\mathbb{N}} \left(\sup_{m\in\mathbb{N}} u_{n+m} \right)_n \quad \text{et} \quad \underline{\lim}\, u = \sup_{n\in\mathbb{N}} \left(\inf_{m\in\mathbb{N}} u_{n+m} \right)_n . \tag{3.3}$$

Cette caractérisation permet de démontrer des propriétés sur les limites inférieure et supérieure à partir de résultats déjà démontrés sur les bornes de suites. Cette caractérisation fait intervenir des extrema de suites à valeurs dans $\mathbb{R}$ et dans $\overline{\mathbb{R}}$. J'ai donc choisi, contrairement aux ensembles, de ne définir qu'une seule version pour chacun des extrema : les bornes supérieures et inférieures de suites à valeurs dans $\overline{\mathbb{R}}$.

Même s'il est possible de définir la borne supérieure d'une suite comme la borne supérieure de l'ensemble de ses valeurs $\{y \in \overline{\mathbb{R}} \mid \exists n \in \mathbb{N},\ y = u_n\}$, j'ai choisi d'en donner une définition en "$\forall \varepsilon, ...$". J'ai noté que cette définition implique une meilleur compatibilité avec les définitions utilisées par la suite comme les limites. La définition utilisant la borne supérieure de l'ensemble des valeurs a tout de même été démontrée comme une caractérisation afin de démontrer l'existence de ces bornes supérieures de suites.

```
Definition is_sup_seq (u : nat -> Rbar) (l : Rbar) :=
  match l with
    | Finite l => forall (eps : posreal), (forall n, Rbar_lt (u n) (l + eps))
                                       /\ (exists n, Rbar_lt (l - eps) (u n))
    | p_infty => forall M : R, exists n, Rbar_lt M (u n)
    | m_infty => forall M : R, forall n, Rbar_lt (u n) M
  end.
```

Dans le cas où la borne supérieure est finie, il est courant de définir la borne supérieure par "$(\forall n \in \mathbb{N},\ u_n \leq_{\overline{\mathbb{R}}} l) \wedge (\forall k < l,\ \exists n \in \mathbb{N},\ k <_{\overline{\mathbb{R}}} u_n)$". J'ai cependant choisi la définition dont le code est présenté ci-dessus pour une meilleure compatibilité avec la limite supérieure d'une suite de réels. Pour la même raison, dans le cas où la borne supérieure est $-\infty$, j'ai choisi de passer par des inégalités plutôt que de la définir par "$\forall n,\ u_n = -\infty$".

La borne inférieure d'une suite a été définie de façon analogue, et l'existence de ces extrema a été démontrée en utilisant leur caractérisation comme extrema de l'ensemble de leurs valeurs. J'ai ainsi pu définir les deux premières fonctions totales calculant la borne supérieure et la borne inférieure d'une suite :

```
Sup_seq : (nat -> Rbar) -> Rbar   et   Inf_seq : (nat -> Rbar) -> Rbar.
```

Pour achever la construction des fonctions totales sur les suites de réels, j'ai défini la limite supérieure de façon usuelle :

```
Definition is_LimSup_seq (u : nat -> R) (l : Rbar) :=
  match l with
    | Finite l => forall eps : posreal,
```

```
      (forall N : nat, exists n : nat, (N <= n)%nat /\ l - eps < u n)
      /\ (exists N : nat, forall n : nat, (N <= n)%nat -> u n < l + eps)
  | p_infty => forall M : R,
      (forall N : nat, exists n : nat, (N <= n)%nat /\ M < u n)
  | m_infty => forall M : R,
      (exists N : nat, forall n : nat, (N <= n)%nat -> u n < M)
end.
```

L'existence a été démontrée à partir de la caractérisation des limites supérieures utilisant les extrema de suites (formule 3.3). Cette formule se traduit en Coq par

```
Lemma is_LimSup_infSup_seq (u : nat -> R) (l : Rbar) :
  is_LimSup_seq u l <-> is_inf_seq (fun m => Sup_seq (fun n => u (n + m)%nat)) l.
```

Ce théorème pour le passage entre la limite supérieure et les bornes est un premier exemple de l'importance des fonctions totales pour écrire les différentes quantités qui nous intéressent. Cet énoncé est en effet plus facile à lire que s'il avait fallu construire la suite $(\sup_{m\in\mathbb{N}}(u_{n+m})_n)_m$ "à la main" avant de pouvoir l'utiliser.

Plusieurs lemmes à propos des liens entre les limites supérieures et inférieures et les limites de suites ont été démontrés. En particulier, le lemme

```
Lemma ex_lim_LimSup_LimInf_seq (u : nat -> R) :
  ex_lim_seq u <-> LimSup_seq u = LimInf_seq u.
```

couplé à la décidabilité de l'égalité dans $\overline{\mathbb{R}}$ nous donne la décidabilité de la convergence des suites.

3.5.3 Rendre total ce qui ne l'est pas

Les fonctions totales qui permettent d'écrire les limites, dérivées et intégrales ont été construites de proche en proche en utilisant les théorèmes ci-dessous :

1. Si la suite (u_n) est convergente, alors $\lim\ u_n = \overline{\lim}\ u_n = \underline{\lim}\ u_n$.
2. Si la fonction f converge en $a \in \overline{\mathbb{R}}$, alors pour toute suite u_n de $\mathbb{R}\backslash\{a\}$, $\lim\ u_n = a \Rightarrow \lim_a f = \lim f(u_n)$.
3. Si la fonction f est dérivable en $a \in \mathbb{R}$, alors $f'(x) = \lim_{h\to 0} \frac{f(x+h)-f(x)}{h}$.
4. Si la fonction f est intégrable entre a et b, alors $\int_a^b f = \lim_{|x|\to 0} \sum_{k=0}^{n}(x_{k+1}-x_k)f(y_k)$, avec (x_k, y_k) des paires de suites telles que $\forall k, k \leq n \Rightarrow x_k \leq y_k \leq x_{k+1}$, $x_0 = a$ et $x_n = b$.

Pour toutes les fonctions totales présentées par la suite, l'unicité et la correction ont été démontrées suivant le modèle présenté dans la section 3.1. Ainsi, en cas de convergence, la valeur retournée est celle attendue, et sinon les fonctions totales renvoient une valeur arbitraire.

Limites de suites

En cas de convergence, les limites supérieure et inférieure étant égales à la limite, il aurait été possible de choisir l'une des deux comme valeur pour la fonction totale. J'ai cependant choisi de prendre une définition moins directe :

$$\texttt{Lim\_seq (u : nat-> R) : Rbar} \quad := \quad \frac{\overline{\lim}\ \mathrm{u} + \underline{\lim}\ \mathrm{u}}{2}.$$

Elle a pour avantage d'offrir quelques réécritures sans hypothèse. En effet, les limites supérieures et inférieures se comportant bien avec les multiplications par un scalaire strictement positif et étant échangées par l'opposé, j'ai pu démontrer le théorème suivant :

```
Lemma Lim_seq_scal_l (u : nat -> R) (a : R) :
  Lim_seq (fun n => a * u n) = Rbar_mult a (Lim_seq u).
```

Les séries et séries entières étant des limites de suites particulières, les fonctions totales `Series : (nat → R) → R` et `PSeries : (nat → R) → R → R` ont été définies à partir de la fonction totale `Lim_seq` et de `real` pour restreindre le type de retour aux réels.

Limites de fonctions

Concernant les limites de fonctions, le théorème "si la fonction f converge en $a \in \overline{\mathbb{R}}$, alors pour toute suite u_n de $\mathbb{R}\backslash\{a\}$, $\lim\ u_n = a \Rightarrow \lim_a f = \lim f(u_n)$" déjà évoqué est vrai pour toute suite convergeant vers le point réel ou infini qui nous intéresse. J'ai donc défini la famille de suites suivante :

$$\texttt{Rbar\_loc\_seq (x : Rbar) (n : nat) : R} \ := \begin{cases} \texttt{x} + 1/(n+1) & \text{si } \texttt{x} \in \mathbb{R} \\ n & \text{si } \texttt{x} = +\infty \\ -n & \text{si } \texttt{x} = -\infty \end{cases}$$

et j'ai démontré qu'elles convergent vers leur paramètre `x` :

```
Lemma filterlim_Rbar_loc_seq (x : Rbar) :
  filterlim (Rbar_loc_seq x) eventually (Rbar_locally' x).
```

J'ai ainsi pu construire une fonction totale simple pour les limites de fonctions :

```
Lim (f : R-> R) (x : Rbar) := Lim_seq (fun n => f (Rbar_loc_seq x n)).
```

ainsi que la fonction totale `Derive` qui correspond à la dérivée en utilisant la définition donnée plus haut.

Les deux fonctions totales ainsi définies héritent également des théorèmes de réécriture sans hypothèse de `Lim_seq` pour la multiplication par un scalaire.

Intégrale de Riemann

La définition présentée dans la section 3.3.2 pour l'intégrale de Riemann implique que lorsqu'une fonction f est intégrable entre a et b, alors pour toute suite de subdivisions pointées entre a et b (σ_n, ξ_n) dont le pas tend vers 0, la limite des sommes de Riemann associées $\sum(\sigma_{n,i+1} - \sigma_{n,i}) f(\xi_{n,i})$ est l'intégrale de la fonction f entre a et b. J'ai défini la fonction totale `RInt` correspondant à l'intégrale de Riemann en utilisant la subdivision uniforme entre a et b :

$$n \mapsto \sum_{k=0}^{n} (x_{n,k+1} - x_{n,k})\, f\left(\frac{x_{n,k} + x_{n,k+1}}{2}\right), \quad \text{avec } x_{n,k} = a + \frac{k \cdot (b-a)}{n+1}.$$

Il aurait été possible de factoriser les $(x_{n,k+1} - x_{n,k})$ qui valent tous $(b-a)/(n+1)$ pour obtenir une écriture plus simple, mais cela aurait impliqué la duplication de théorèmes plus généraux sur les sommes de Riemann et les fonctions étagées.

La construction de cette fonction totale a été l'une des motivations pour les modifications effectuées sur l'intégrale de Riemann qui ont été présentées dans la section 3.3. En effet, pour démontrer que cette limite est bien égale à l'intégrale, il a fallu commencer par exprimer également l'intégrale comme une limite.

En plus des réécritures pour la multiplication par un scalaire, la réécriture de `RInt` pour la composition par une fonction affine se fait également sans hypothèses :

```
Lemma RInt_comp_lin (f : R -> R) (u v a b : R) :
```

$$\int_a^b (u \cdot f(u \cdot x + v))\ dx = \int_{u \cdot a + v}^{u \cdot b + v} f(x)\, dx$$

3.5.4 Une application directe : les dérivées itérées

La première application de ces fonctions totales a été la possibilité d'écrire facilement les fonctions faisant intervenir plusieurs opérateurs comme les dérivées d'intégrales à paramètre et les dérivées itérées. Même si la définition de ces fonctions était possible avec les outils de la bibliothèque standard (voir [LM12] pour un exemple), leur manipulation était difficile en raison des nombreux termes de preuve à fournir. La possibilité d'écrire des fonctions dérivées sans fournir de terme de preuve offre plus de flexibilité.

La première étape concernant la notion de dérivée itérée a été de définir une fonction totale permettant d'écrire la dérivée n-ème d'une fonction :

```
Fixpoint Derive_n (f : R -> R) (n : nat) x :=
  match n with
    | O => f x
    | S n => Derive (Derive_n f n) x
  end.
```

Par définition de la dérivée, la dérivée $(n+1)$-ème est bien définie en x si la dérivée n-ème est bien définie localement autour de x et est dérivable en x. L'argument "bien défini localement" étant lourd à manipuler, la définition choisie pour `is_derive_n` est légèrement plus simple :

```
Definition is_derive_n f n x l :=
  match n with
  | 0 => f x = l
  | S n => is_derive (Derive_n f n) x l
  end.
```

La dérivabilité n-ème est définie de façon analogue :

```
Definition ex_derive_n f n x :=
  match n with
  | 0 => True
  | S n => ex_derive (Derive_n f n) x
  end.
```

Cela nous donne donc des définitions simples, mais ne possédant pas la propriété qu'être dérivable $(n+1)$ fois en un point implique être dérivable n fois en ce point. Cela impose de supposer l'existence de toute les dérivées itérées jusqu'au rang n comme cela a été fait dans le théorème de Taylor-Lagrange :

```
Theorem Taylor_Lagrange (f : R -> R) (n : nat) (x y : R) :
  x < y ->
  (forall t, x <= t <= y -> forall k, (k <= S n)%nat -> ex_derive_n f k t) ->
  exists zeta, x < zeta < y /\
    f y = sum_f_R0 (fun m => (y - x) ^ m / INR (fact m) * Derive_n f m x) n
                   + (y - x) ^ (S n) / INR (fact (S n)) * Derive_n f (S n) zeta.
```

Dans la pratique cela ne pose pas de problème. En effet, les fonctions considérées sont généralement infiniment dérivables sur un ensemble ouvert. La définition des dérivées itérées devra cependant être étudiée attentivement afin de pouvoir généraliser cette notion à des espaces ne possédant pas de fonctions totales permettant d'écrire les dérivées (voir chapitre 4 pour la généralisation de la différentiabilité).

3.6 Automatisation

Au cours de ma thèse, une tactique permettant de démontrer automatiquement (ou au moins simplifier les démonstrations) des énoncés de la forme `derivable_pt_lim f x l`, puis `is_derive f x l`, avec `f` une fonction réelle et `x` et `l` deux réels, a été écrite. C'est une tactique par réflexion ; elle est décrite dans la figure 3.2 : la fonction `f` est d'abord transformée par une tactique `reify` en une expression `e` du type inductif `expr`. Une fonction `D` est alors appliquée à l'expression `e` afin de calculer d'une part une expression représentant la dérivée `de` et d'autre part les conditions de dérivabilité. Pour finir, le lemme de correction, disant que l'expression `de` ainsi calculée représente bien une dérivée de la fonction représentée par `e`, renvoie trois buts : le premier but demande de vérifier que l'expression `e` représente bien la fonction `f`, le deuxième but demande de vérifier que l'expression `de` représente bien la dérivée `l` fournie par l'utilisateur et le troisième but correspond aux hypothèses nécessaires à la dérivabilité. Dans la majorité des cas, les deux premiers buts sont automatiquement déchargés par la tactique.

On peut résumer la construction de cette tactique de la façon suivante :

1. Construction des dérivées formelles :
   (a) Conception d'un type inductif `expr` permettant de représenter les fonctions.
   (b) Définition d'une fonction `D` retournant une expression de la dérivée formelle `e'` de l'expression `e` passée en paramètre.
2. Conception de la tactique `autoderive` :
   (a) Définition d'une fonction `interp` permettant de transformer une expression `e` en fonction réelle `f`.
   (b) Définition d'une tactique `reify` permettant de transformer une fonction `f` en expression `e`.
   (c) Démonstration du lemme de correction permettant de justifier que si une fonction `interp e` est dérivable, alors sa dérivée est `interp (D e)`.

Toutes les fonctions n'étant pas dérivables sur `R`, il a fallu trouver un moyen pour construire le domaine de dérivabilité. Dans un premier temps, le choix a été de les représenter par des fonctions partielles semblables à celles définies dans la bibliothèque C-CoRN évoquée dans la section 2.4.2. Dans la version actuelle de la tactique, le domaine de dérivabilité est représenté par un type inductif `domain`.

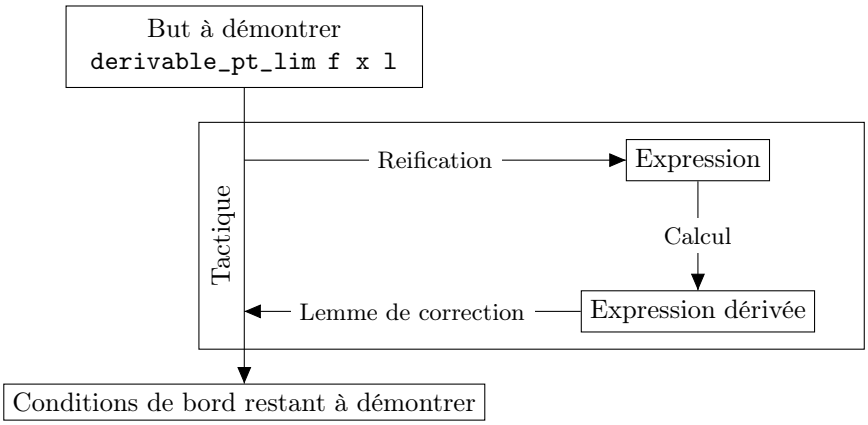


FIGURE 3.2 – Tactique par réflexion

3.6.1 Les expressions

Première version Les expressions de la première version permettaient d'exprimer les fonctions construites à partir d'une seule variable :

$$\texttt{e ::= } x \texttt{ | } cst \texttt{ | e + e | e } \times \texttt{ e | } - \texttt{ e | } \int_{\texttt{e}}^{\texttt{e}} f(t)\,dt \texttt{ | } f(\texttt{e}) \texttt{ | } f(\texttt{e},\texttt{e})$$

avec x la variable par rapport à laquelle on dérive et f un symbole de fonction arbitraire à un ou deux arguments dont le type sera développé dans la section suivante.

La construction d'une expression à partir d'une fonction est faite par une tactique `reify` programmée en `Ltac` [Del00] en étudiant récursivement le terme de tête de la fonction. Une optimisation est faite pour limiter le nombre de constantes, et donc le nombre de zéros lors de la dérivation. Concernant les intégrales, seules les fonctions à une variable continues sur $\mathbb{R}$ sont traduites par l'expression représentant l'intégrale. Cette expression est du type `Int : Cont_F -> expr -> expr -> expr` où `Cont_F` est le type des fonctions partielles continues sur $\mathbb{R}$ tout entier. Pour donner un exemple, tiré de l'application sur la formule de d'Alembert (voir section 5.3), une expression (simplifiée pour ne pas faire apparaître le passage entre les fonctions totales et les fonctions partielles) correspondant à la fonction

$$\beta(x,t) = (1/(2c)) \times \int_{x-ct}^{x+ct} u_1(s)\,ds$$

pour la dérivation par rapport à la variable x est :

```
(Binary Emult (Cst (1 / (2 * c)))
  (Int {| fct := u1; Cf_cond := C_u1 |}
    (Binary Eplus Var (Unary Eopp (Cst (c * t))))
    (Binary Eplus Var (Cst (c * t)))))
```

avec `C_u1` une preuve que la fonction `u1` est continue. La variable par laquelle on dérive est représentée par `Var`. Les opérations arithmétiques ont été regroupées par arité : l'addition et la multiplication sont représentées respectivement par `Binary Eplus` et `Binary Emult` et l'opposé est représenté par `Unary Eopp`. Cela permet d'optimiser l'opérateur de dérivation `D` dans le but de limiter le nombre de dérivation de constantes. Les intégrales des fonctions dont la preuve de continuité n'est pas disponible dans le contexte sont vues comme les fonctions à deux variables $(a,b) \mapsto \int_a^b f(t)\,dt$. C'est alors à l'utilisateur d'appliquer le théorème de différentiabilité pour continuer la démonstration.

Deuxième version Les expressions construites dans la seconde version de la tactique permettent de représenter un plus grand nombre de fonctions. En effet, elles permettent maintenant de représenter des fonctions avec un nombre arbitraire de variables :

$$\begin{aligned}\texttt{e := } x_i \texttt{ | } cst \texttt{ | e + e | e } \times \texttt{ e | } - \texttt{ e | e}^{-1} \texttt{ | } \int_{\texttt{e}}^{\texttt{e}} \texttt{ e}\\ \texttt{| } f|_{\mathbb{R}}(\texttt{e}) \texttt{ | } f|_{D}(\texttt{e}) \texttt{ | } f\,(\texttt{e},...,\texttt{e}) \texttt{ | } f'_j\,(\texttt{e},...,\texttt{e})\end{aligned}$$

Les fonctions disposent de différents constructeurs en fonction des informations disponibles et de leur signification :

- $f|_{\mathbb{R}}(\mathtt{e})$ et $f|_D(\mathtt{e})$ représentent l'application des fonctions dont la dérivabilité a été démontrée. Cette preuve est enregistrée dans la classe `UnaryDiff` quand la fonction est dérivable sur $\mathbb{R}$ entier, ou dans la classe `UnaryDiff'` quand la fonction est dérivable sur un sous-domaine D seulement. Les instances de `UnaryDiff` et `UnaryDiff'` correspondant aux fonctions de la bibliothèque standard ont été définies.
- $f\,(\mathtt{e}, ..., \mathtt{e})$ représente l'application des symboles de fonction non interprétés.
- $f'_j\,(\mathtt{e}, ..., \mathtt{e})$ représente l'application des dérivées des symboles de fonction non interprétés par rapport à l'une des variables.

La construction des expressions par la tactique `reify` de cette version est semblable à la version précédente. Il n'y a cependant plus besoin de garder dans les expressions d'information concernant les fonctions de départ. Cela permet de traduire toutes les intégrales en utilisant le constructeur dédié. Le dernier constructeur pour les fonctions décrit ci-dessus n'est pas produit par cette tactique. Il permet juste d'exprimer plus facilement l'expression de la dérivée.

Pour reprendre l'exemple de la fonction β, l'expression (non simplifiée cette fois-ci) retournée par la tactique `reify` est maintenant :

```
(Binary Emult (Cst (1 / (2 * c)))
  (Int (AppExt 1 u1 [:: Var 0 ])
    (Binary Eplus (Var 0) (Unary Eopp (Cst (c * t))))
    (Binary Eplus (Var 0) (Cst (c * t)))))
```

Pour cette expression, les principales différences par rapport à la version précédente sont les représentations de la fonction unaire u_1 et des variables. Les variables sont numérotées par leur indice de De Bruijn. Ainsi, la variable 0 des bornes de l'intégrale `(Binary Eplus (Var 0) ...)` correspond à la variable de dérivation x et celle de `(AppExt 1 u1 [:: Var 0 ])` correspond à la variable d'intégration s.

Il est important de noter que pour la dérivée de la fonction

$$\gamma(x,t) = 1/(2c) \times \int_0^t \int_{x-c(t-\tau)}^{x+c(t-\tau)} f(\xi,\tau)\,d\xi\,d\tau,$$

par rapport à la variable x, la nouvelle tactique `reify` prend également en compte les variables sous le signe intégral :

```
(Binary Emult (Cst (1 / (2 * c)))
  (Int
    (Int (AppExt 2 f [:: Var 1; Var 0])
      (Binary Eplus (Var 1)
        (Unary Eopp (Binary Emult (Cst c)
                                  (Binary Eplus (Cst t) (Unary Eopp (Var 0))))))
      (Binary Eplus (Var 1)
        (Binary Emult (Cst c) (Binary Eplus (Cst t) (Unary Eopp (Var 0))))))
    (Cst 0) (Cst t)))
```

Il y a dans cet exemple plusieurs intégrales. Dans l'expression de la fonction f `(AppExt 2 f [:: Var 1; Var 0])` la variable 0 représente la variable d'intégration ξ et la variable 1 correspond au paramètre τ. Dans les bornes de cette première intégrale, la variable τ est maintenant représentée par la variable 0 et la variable de dérivation x est représentée par la variable 1.

3.6.2 L'opérateur de dérivation et le lemme de correction

Les expressions ainsi construites nous permettent de représenter les fonctions à dériver. Pour les utiliser dans des preuves de dérivabilité, il faut non seulement déterminer la valeur, mais également les conditions de dérivabilité. Le tableau suivant donne quelques exemples de dérivation en $x \in \mathbb{R}$:

Fonction	**Dérivée**	**Condition**
x_i	1 si x est la variable x_i 0 sinon	
$f \cdot g$	$f'(x) \cdot g(x) + f(x) \cdot g'(x)$	$x \in D_f \cap D_g$
$f \circ g$	$g'(x) \cdot f'(g(x))$	$x \in D_g \wedge g(x) \in D_f$
$x \mapsto \int_{a(x)}^{b(x)} f(x,t)\,dt$	$b'(x) \cdot f(x,b(x)) - a'(x) \cdot f(x,a(x))$ $+ \int_{a(x)}^{b(x)} \frac{\partial f}{\partial x}(x,t)\,dt$	$x \in D_a \cap D_b$ $\wedge$ f continue en $a(x)$ et $b(x)$ $\wedge$ conditions pour $\int$

avec D_h le domaine de dérivabilité de la fonction h. Pour résumer, les conditions de dérivabilité portant sur les fonctions représentées par les `expr`, à l'exception des intégrales, correspondent exactement aux conditions d'existence de la fonction dérivée. Pour les dérivées sur les intégrales et intégrales à paramètre, des conditions supplémentaires de continuité de la fonction intégrée interviennent également. Les conditions de dérivabilité sont calculées par la fonction `D` en même temps que la valeur de la dérivée.

En utilisant des fonctions partielles

La première version de la tactique utilisant les dérivées et intégrales de la bibliothèque standard, il était nécessaire de fournir des termes de preuve pour pouvoir les écrire. Nous avons choisi de définir les fonctions partielles sous la forme `R (-> R) -> partial_val` avec :

```
Record partial_val := Pval {
  pdom : Type;
  pval : pdom -> R;
  pHeq : forall d1 d2, pval d1 = pval d2
}.
```

Les opérations arithmétiques ont été définies sur les `partial_val` afin de couvrir les opérations traitées par les expressions.

Ces valeurs partielles permettent d'exprimer facilement les fonctions partielles usuelles ainsi que les dérivées et les intégrales :

```
Definition PartialRinv (x : R) : partial_val :=                    (* inverse *)
  Pval (x <> 0) (fun _ => / x) (fun _ _ => eq_refl).
Definition PartialDerive (f : R -> R) (x : R) : partial_val :=     (* dérivée *)
  Pval (derivable_pt f x) (derive_pt f x) (pr_nu f x).
Definition PartialRint (f : R -> R) (a b : R) : partial_val :=    (* intégrale *)
  Pval (Riemann_integrable f a b) (RiemannInt (b:=b)) (RiemannInt_P5 (b:=b)).
```

Cette forme ne permet cependant pas de traiter le cas des dérivées et intégrales de fonctions partielles car il faudrait transformer les fonctions partielles, sur des domaines qui ne sont pas forcément décidables, en fonctions totales. Nous avons donc dupliqué les définitions et théorèmes d'analyse réelle utiles à cette tactique.

La fonction `interp : expr -> partial_val` a été définie afin de transformer les expressions en valeurs partielles. Le lemme de correction de la fonction `D` a été démontré afin de revenir dans l'espace des fonctions partielles :

```
Lemma D_correct : forall (e : expr) (x : R),
  let vd := interp (D e) x in
  forall d : pdom vd, Pfun_derivable_pt_lim (interp e) x (pval vd d).
```

Ce lemme ne s'intéressant qu'aux fonctions partielles, c'est en fait un corollaire qui est utilisé dans la tactique.

```
Lemma AutoDerive_helper :
  forall e f x l,
  (forall x, let v := interp e x in forall d : pdom v, pval v d = f x) ->
  (let vd := interp (D e) x in { d : pdom vd | pval vd d = l }) ->
  derivable_pt_lim f x l.
```

Autrement dit, la tactique `AutoDerive` transforme un but de la forme `derivable_pt_lim f x l` en deux sous-buts. Le premier demande de vérifier que l'interprétation de l'expression `e` prend bien les mêmes valeurs que la fonction `f` sur le domaine de définition de `e`. Le second demande de vérifier que l'interprétation de `D e` a un domaine non vide et que sa valeur est égale à `l`.

Dans une approche par réflexion classique, le premier sous-but serait traité immédiatement par β-conversion, à supposer que la tactique `reify` qui produit l'expression `e` à partir de la fonction `f` n'a pas fait d'erreur [5]. Dans notre cas, ce n'est pas tout à fait vrai, à cause des termes de preuve présents dans les intégrales. En effet, les termes de preuve générés par `interp` ne sont pas ceux de l'utilisateur et il y a donc potentiellement une phase de réécriture qui est gérée par la tactique pour unifier les deux fonctions. Quoi qu'il en soit, cette étape est complètement transparente et l'utilisateur ne verra jamais ce sous-but.

L'autre sous-but est plus intéressant. Dans un premier temps, la tactique demande démontrer l'existence de la dérivée calculée. Dans les cas où la fonction à dériver contient des symboles de fonction non

5. L'étape de réification se fait nécessairement en Ltac et ne peut donc pas être prouvée correcte a priori.

interprétés, l'utilisateur devra démontrer la dérivabilité de ces fonctions aux points considérés. La tactique cherche cependant à en éliminer le plus possible en s'aidant du contexte. L'autre partie de ce sous-but demande de prouver l'égalité entre la valeur de la dérivée et la dérivée utilisateur. Ce but est laissé à l'utilisateur, la tactique essayant seulement d'appliquer les tactiques de la bibliothèque standard `ring` et `field` au cas où les deux expressions seraient facilement identifiables, ce qui a toujours été le cas pour nos exemples, dont la vérification de la formule de d'Alembert.

En utilisant des fonctions totales

L'ajout des fonctions totales pour représenter les dérivées et intégrales a permis de séparer les écritures de leur preuve d'existence. Dans la seconde version de la tactique, la fonction de dérivation formelle `D` renvoie un couple composé d'une expression représentant la dérivée et d'un terme de type `domain` permettant de représenter les conditions de dérivabilité. Le lemme de correction est alors :

```
Lemma D_correct :
  forall (e : expr) (l : seq R) (n : nat),
       let '(a, b) := D e n in
            interp_domain l b ->
            is_derive (fun x : R => interp (set_nth 0 l n x) e)
              (nth 0 l n) (interp l a).
```

avec `l` la liste des valeurs des variables et `n` le numéro de la variable à dériver. Contrairement à la version précédente, la fonction `interp` renvoie une fonction à valeurs dans `R`. La fonction `fun x : R => interp (set_nth 0 l n x) e` est donc convertible en la fonction ayant servi à la fabriquer. La tactique `auto_derive` vérifie alors que la dérivée fournie par l'utilisateur est égale à la valeur calculée `interp l a` sous les conditions `interp_domain l b`. Il faut noter que le domaine est simplifié avant d'être transmis à l'utilisateur à l'aide de la fonction `simplify_domain : domain -> domain` qui simplifie les conjonctions et certains domaines. Par exemple, une propriété interprétée initialement par `locally a (fun _ => True)` sera interprétée, après simplification du domaine, par `True`.

3.6.3 Commentaires sur les deux versions

Les deux versions ci-dessus permettent donc de simplifier et de résoudre les preuves de différentiabilité. Elles s'appliquent toutes les deux sur les buts de la forme `derivable_pt_lim f x l`. La seconde version s'applique également sur `is_derive f x l` qui est l'équivalent du but précédent dans la bibliothèque Coquelicot, ainsi que sur `derivable_pt` et `ex_derive` qui permettent d'exprimer la dérivabilité respectivement dans la bibliothèque standard et dans la bibliothèque Coquelicot. Même si cela n'a pas été fait, il serait possible d'ajouter à ces tactiques, avec peu de travail, la possibilité de réécrire des expressions contenant
`derive_pt f x pr` ou `Derive f x`.

Les fonctions pouvant être dérivées grâce à cette tactique sont construites à partir de constantes, de variables, d'intégrales et de fonctions arbitraires à une ou deux variables. La seconde version permet en plus de traiter les intégrales à paramètres. La présence des intégrales et des fonctions arbitraires permet de traiter plus de fonctions que la tactique `reg` de la bibliothèque standard.

Il est important de noter que la première tactique `AutoDerive` est indépendante de la bibliothèque Coquelicot [6]. L'intégrale utilisée étant celle de la bibliothèque standard, seules les intégrales de fonctions continues sur $\mathbb{R}$ sont effectivement dérivées ; les autres intégrales sont traitées comme des fonctions à deux variables (les deux bornes). Cette tactique passant par le type auxiliaire `partial_val`, l'égalité entre la fonction fournie par l'utilisateur et celle effectivement dérivée doit être vérifiée. Dans la plupart des cas, cette égalité est automatiquement prouvée par la tactique. La définition des fonctions totales dans la bibliothèque Coquelicot a permis de gérer plus facilement les conditions de dérivabilité. Le développement de cette version basée sur la bibliothèque standard n'a donc pas été poursuivi.

En plus de traiter plus de fonctions que la première version `AutoDerive` développée auparavant, la seconde version `auto_derive` offre la possibilité à l'utilisateur d'ajouter lui-même, et de façon simple, des fonctions pouvant être reconnues et dérivées automatiquement. Il lui suffit pour cela de définir une nouvelle instance de la classe `UnaryDiff` ou `UnaryDiff'` qui sera reconnue par la tactique. Cette deuxième version de la tactique est intégrée à la bibliothèque Coquelicot et a été testée tout d'abord sur la formule de d'Alembert, puis sur les fonctions de Bessel pour simplifier la vérification de l'équation différentielle. L'extension de cette tactique à l'étude de la différentiabilité est maintenant possible car la plupart des

6. Une version mise à jour pour Coq version 8.4pl2 est disponible sur ma page web `https://www.lri.fr/~lelay/Stage/`.

opérations traitées par cette tactique ont été généralisées (voir chapitre 4). Il est donc envisageable de l'utiliser pour dériver des fonctions complexes à une variable (c'est-à-dire de $\mathbb{C}$ dans $\mathbb{C}$), voire pour étudier la différentiabilité de fonctions à plusieurs variables. Une structure supplémentaire pourra être ajoutée à la hiérarchie algébrique afin de traiter la différentiabilité de la multiplication qui n'est possible que dans le cas où celle-ci est commutative.

Chapitre 4

Vers une bibliothèque plus générale

Les outils présentés dans le chapitre 3 avaient pour principal objectif de faciliter la démontration de théorèmes d'analyse réelle en Coq. Ainsi, les réels étendus ont été introduits afin de pouvoir étudier les limites infinies de suites et de fonctions ; des fonctions totales ont été formalisées afin de faciliter l'écriture et la manipulation des limites, dérivées et intégrales. Quelques éléments d'analyse dans $\mathbb{R}^2$ ont également été formalisés pour pouvoir traiter les intégrales à paramètre.

L'analyse n'est cependant pas limitée à l'analyse réelle. Par exemple, de nombreux théorèmes portant sur les limites, dérivées et intégrales sont semblables, autant dans leur énoncé que dans leur démonstration, en analyse réelle et en analyse complexe. En utilisant le polymorphisme de Coq, il est possible de démontrer des théorèmes s'appliquant aussi bien aux nombres réels qu'aux nombres complexes. Nous allons généraliser toutes les notions définies auparavant dans le cadre des nombres réels, à savoir les limites, les séries et séries entières, les dérivées et l'intégrale de Riemann. L'objectif est de pouvoir démontrer de la même façon, aux notations près, des théorèmes comme `forall x, is_derive (fun x => x + 1) x 1`, que la variable `x` soit réelle ou complexe.

Plutôt que de construire une hiérarchie exhaustive, comme dans C-CoRN/MathClasses [CFGW04] ou Isabelle/HOL, nous avons choisi de nous restreindre à quelques structures algébriques et topologiques permettant de représenter aussi bien des corps commutatifs comme les nombres réels et complexes que des modules sur des anneaux non commutatifs comme les matrices. La généralisation de la notion de limite, développée dans la section 4.1, nous a conduits à formaliser les notions d'espace uniforme et d'espace uniforme complet, ainsi que la notion de filtre afin de gérer aussi facilement les limites au sens topologique usuel (l'image réciproque d'un voisinage est également un voisinage) que des limites plus "exotiques" comme les limites à droite et à gauche dans l'ensemble des nombres réels ou la convergence de subdivisions pointées pour l'intégrale de Riemann. La généralisation de l'intégrale de Riemann et des séries entières, développée dans la section 4.2, nous a conduits à formaliser des anneaux munis d'une valeur absolue et des modules normés sur ces anneaux qui soient compatibles avec les espaces uniformes précédemment construits. Pour finir, la généralisation de la différentiabilité, développée dans la section 4.3, nous a conduits à formaliser les notions d'équivalence et de prédominance (notation o) pour les fonctions.

4.1 Généraliser les limites

La généralisation de la notion de limite a déjà été abordée dans le chapitre 3. En effet, le prédicat `filterlim` généralisant cette notion a été utilisé pour définir les limites généralisées dans la section 3.2.2 et donner une nouvelle définition de l'intégrale de Riemann dans la section 3.3. Cette nouvelle définition de la limite est cependant beaucoup plus générale que les applications déjà présentées. Elle permet en effet de représenter toute notion reposant sur la convergence.

Il est possible de définir la notion de limite dès que l'on dispose d'une structure d'espace topologique. Ces espaces topologiques sont représentés dans la bibliothèque Coquelicot par la notion de filtre. Plutôt que de décrire tous les ouverts de l'espace considéré, comme c'est le cas lorsqu'on décrit un espace topologique, les filtres permettent en fait de représenter facilement les voisinages d'un point donné.

4.1.1 Limites et filtres

La notion de limite pour une fonction $f : \mathtt{U} \to \mathtt{V}$ a été généralisée en utilisant le prédicat `filterlim` défini par

$$\mathtt{filterlim}(f, \mathtt{F}, \mathtt{G}) \quad \Leftrightarrow \quad \forall P \in \mathtt{G},\ f^{-1}(P) \in \mathtt{F},$$

où `F` : (`U` → `Prop`) → `Prop` et `G` : (`V` → `Prop`) → `Prop` sont des collections de sous-ensembles des espaces de départ et d'arrivée de la fonction f. Cette approche est inspirée des limites définies dans Isabelle/HOL [HIH13]. J'ai présenté dans la section 3.2.2 un certain nombre de collections de sous-ensembles utilisées pour étudier les fonctions réelles. Dans le cas des limites de suites et de fonctions, ces collections sont usuellement des familles de voisinages comme `Rbar_locally x` dans $\overline{\mathbb{R}}$ ou `eventually` dans $\mathbb{N}$. J'ai également présenté `at_right x` et `at_left x` qui permettent d'exprimer les limites à droite et à gauche, ainsi que `Riemann_fine a b` qui permet d'exprimer l'intégrale de Riemann. Ces familles d'ensembles, qui peuvent également être vues comme des voisinages dans des espaces moins simples que $\overline{\mathbb{R}}$ ou $\mathbb{N}$, sont des filtres. Elles sont caractérisées dans la bibliothèque Coquelicot par le prédicat `Filter` et vérifient les trois propriétés suivantes :

- L'espace tout entier appartient à `G` : $\mathtt{V} \in \mathtt{G}$;
- `G` est stable par intersection : $\forall P, Q \subseteq \mathtt{V}, P \in \mathtt{G} \wedge Q \in \mathtt{G} \Rightarrow P \wedge Q \in \mathtt{G}$;
- `G` est stable par sur-ensemble : $\forall P, Q \subseteq \mathtt{V}, P \subseteq Q \wedge P \in \mathtt{G} \Rightarrow Q \in \mathtt{G}$.

Dans certaines démonstrations, il est également utile de pouvoir extraire des témoins, c'est-à-dire récupérer un élément de tout sous-ensemble appartenant au filtre. Pour ces situations, le concept de filtre propre a été formalisé : ce sont les filtres ne contenant pas l'ensemble vide.

Dans la bibliothèque Coquelicot, les filtres ont été définis en utilisant les classes de types [CS12] :

```
Class Filter {T : Type} (F : (T -> Prop) -> Prop)
```

Cette structure permet à Coq de déduire que certaines collections d'ensembles sont des filtres, que ce soit par démonstration ou par construction. À partir des `Global Instance` disponibles dans la bibliothèque Coquelicot et de ceux définis par l'utilisateur, Coq peut instancier automatiquement les hypothèses du type `Filter F`. Par exemple, l'instance `Rbar_locally_filter` permet d'utiliser des lemmes concernant `filterlim` pour démontrer des lemmes sur les limites de fonctions réelles sans avoir besoin de fournir la preuve que `Rbar_locally` est un filtre.

Pour certains théorèmes, il fallait de plus supposer que les filtres utilisés ne contenait pas l'ensemble vide. Ces filtres particuliers sont appelés filtres propres. La bibliothèque Coquelicot dispose de deux formalisations de ces filtres propres :

- dans la première, nommée `ProperFilter`, la propriété de ne pas contenir l'ensemble vide est exprimée par le fait que tous les ensembles contenus dans le filtre contiennent au moins un élément,
- dans la deuxième, nommée `ProperFilter'`, cette propriété a été définie directement.

La seconde définition a été introduite en raison de la logique intuitionniste de Coq qui ne permettait pas de démontrer l'existence d'un élément dans certains cas.

4.1.2 Les filtres pré-définis

Des filtres ont été définis afin d'exprimer facilement les propriétés usuelles de l'analyse réelle. La majorité de ces filtres ont déjà été évoqués dans le chapitre 3 pour parler des limites de suites et de fonctions, et même d'intégrales de Riemann. Cette section les détaille ainsi que des moyens pour en engendrer de nouveaux.

Les voisinages pour les limites dans $\overline{\mathbb{R}}$

Les deux filtres `Rbar_locally` et `eventually` présentés dans cette section ont été définis avant de formaliser la notion de filtre. Ils n'étaient à ce moment-là que des notations pour définir les voisinages des points dans $\overline{\mathbb{R}}$ et de l'infini dans $\mathbb{N}$. Ces notations permettaient certes d'améliorer la lisibilité des théorèmes portant à la fois sur les réels finis et sur les infinis, mais de nombreux éléments de démonstration, comme le fait que l'intersection de deux voisinages d'un point a est également un voisinage du point a, étaient redémontrés régulièrement. L'introduction des filtres a donc permis de factoriser un certain nombre de démonstrations.

Étant donné $\mathtt{a} \in \overline{\mathbb{R}}$, `Rbar_locally a` et `Rbar_locally' a` sont deux familles de sous-ensembles de $\mathbb{R}$ permettant d'exprimer respectivement les voisinages de `a` (c'est-à-dire les sous-ensembles contenant un intervalle de la forme $]\mathtt{a}-\varepsilon;\mathtt{a}+\varepsilon[$) et les voisinages pointés de `a` (c'est-à-dire les sous-ensembles contenant un intervalle de la forme $]\mathtt{a}-\varepsilon;\mathtt{a}+\varepsilon[\backslash\{\mathtt{a}\}$). La définition de `Rbar_locally a` a été donnée dans la

section 3.2.2. Nous allons ici généraliser les cas finis de `Rbar_locally a` et `Rbar_locally' a` en utilisant respectivement les familles de voisinages `locally a` et `locally' a` qui caractérisent les voisinages dans un espace uniforme (voir section 4.1.3) :

```
Definition Rbar_locally (a : Rbar) (P : R -> Prop) :=
  match a with
    | Finite a => locally a P
    | p_infty => exists M : R, forall x : R, M < x -> P x
    | m_infty => exists M : R, forall x : R, x < M -> P x
  end.
```

Un traitement homogène de $\overline{\mathbb{R}}$ et des autres espaces demanderait d'éliminer complètement `Rbar_locally` et de ne garder que `locally`. Pour cela, il faudrait munir $\overline{\mathbb{R}}$ d'une structure d'espace uniforme [1], en utilisant par exemple les fonctions tangente et arc-tangente pour "transporter" la structure d'espace uniforme de l'intervalle $[-\pi/2; \pi/2]$ dans $\overline{\mathbb{R}}$, ou toute autre fonction bijective d'un intervalle fermé de $\mathbb{R}$ dans $\overline{\mathbb{R}}$. Cela permettrait de ramener de nombreuses démonstrations sur les limites à des démonstrations de continuité. Par exemple, le théorème $\lim(f+g) = \lim f +_{\overline{\mathbb{R}}} \lim g$ découle du fait que la fonction $(x, y) \mapsto x +_{\overline{\mathbb{R}}} y$ est continue sur son domaine de définition [2]. Cette solution n'est cependant pas satisfaisante car elle impose de travailler sur des fonctions de $\overline{\mathbb{R}}$ dans $\overline{\mathbb{R}}$. Les fonctions doivent donc être composées avec la fonction `real` dans le cas où les limites en l'infini n'interviennent pas dans la démonstration, soit être prolongées par continuité en $\pm\infty$, ce qui revient à démontrer la limite de cette fonction en l'infini.

Le filtre `eventually` permet pour sa part d'exprimer les voisinages de $+\infty$ dans les entiers naturels $\mathbb{N}$, c'est-à-dire les sous-ensembles de $\mathbb{N}$ contenant tous les entiers à partir d'un certain rang :

```
Definition eventually (P : nat -> Prop) :=
  exists N : nat, forall n, (N <= n)%nat -> P n.
```

Comme pour `Rbar_locally`, même s'il est possible de définir une structure d'espace uniforme sur les entiers naturels étendus avec un élément infini, le type des voisinages obtenus ne serait pas celui souhaité. De plus, le seul voisinage intéressant sur les entiers naturels étant celui de l'infini (les voisinages d'un entier $n \in \mathbb{N}$ sont les ensembles d'entiers contenant ce point), définir un tel espace uniforme nécessiterait beaucoup de travail sans pour autant simplifier les démonstrations.

Tous les filtres présentés ci-dessus sont des filtres propres.

Les modificateurs

Les filtres décrits ci-dessus permettent de définir les limites de suites et de fonctions usuelles. Ils ne sont cependant pas suffisants pour exprimer d'autres types de limites comme les limites à droite et à gauche ou les limites dans $\mathbb{R}^n$. Comme fabriquer entièrement un filtre à partir de zéro peut s'avérer lourd, nous avons formalisé des opérateurs permettant de fabriquer de nouveaux filtres à partir de ceux déjà définis.

Image d'un filtre Tout d'abord, pour exprimer facilement $f^{-1}(P) \in \mathtt{F}$ dans la définition de `filterlim`, un premier opérateur `filtermap` a été défini de la façon suivante :

```
Definition filtermap {T U : Type} (f : T -> U) (F : (T -> Prop) -> Prop) :=
  fun (P : U -> Prop) => F (fun x => P (f x)).
```

Les ensembles du nouveau filtre sont ceux dont la pré-image par `f` appartient au filtre de départ. En l'utilisant, la définition Coq de `filterlim` est alors

```
Definition filterlim {T U : Type} (f : T -> U) F G :=
  filter_le (filtermap f F) G.
```

avec `filter_le F G` une relation disant que `G` est inclus dans `F` (`F` est plus petit que `G` dans le sens où les conditions d'appartenance sont moins restrictives).

L'opérateur `filtermap` transforme des filtres propres en filtres propres.

1. généralisation des espaces métriques présentée dans la section 4.1.3
2. Pour tout $x, y \in \overline{\mathbb{R}}$, si $x +_{\overline{\mathbb{R}}} y$ existe, alors $\lim_{(u,v)\to(x,y)}(u +_{\overline{\mathbb{R}}} v) = x +_{\overline{\mathbb{R}}} y$.

Restreindre les filtres Dans le cas des limites à gauche et à droite d'un point $a \in \mathbb{R}$, nous avons besoin de considérer les ensembles contenant respectivement $]a-\varepsilon; a[=]a-\varepsilon; a+\varepsilon[\cap]-\infty; a[$ et $]a; a+\varepsilon[=]a-\varepsilon; a+\varepsilon[\cap]a; +\infty[$. Il s'agit donc des ensembles vérifiant les mêmes conditions que ceux contenus dans `locally a` augmentés de ceux contenant $]-\infty; a[$, respectivement $]a; +\infty[$. Cela a donc conduit à définir `within` qui permet de relâcher des conditions sur un filtre donné afin d'accepter plus d'ensembles :

```
Definition within {T : Type} D (F : (T -> Prop) -> Prop) (P : T -> Prop) :=
  F (fun x => D x -> P x).
```

Ainsi, le filtre correspondant aux limites à droite a été défini de la façon suivante :

```
Definition at_right (x : R) : (R -> Prop) -> Prop :=
  within (fun u : R => Rlt x u) (locally x).
```

avec `locally x` l'ensemble des voisinages du point $\mathtt{x} \in \mathbb{R}$. Le filtre `at_right x` est ainsi la collection des ensembles de $\mathbb{R}$ contenant $]\mathtt{x}-\varepsilon; \mathtt{x}+\varepsilon[\cap]\mathtt{x}; +\infty[=]\mathtt{x}; \mathtt{x}+\varepsilon[$.

La fonction `within` permet de transformer des filtres en filtres, mais ne permet pas de conserver la propriété de filtre propre. Ainsi, sur l'exemple `at_right`, comme la propriété de filtre propre n'a pu être héritée directement de `locally x`, il a fallu la redémontrer.

Un autre filtre intéressant construit avec `within` est celui permettant de définir les intégrales de Riemann comme des limites :

```
Definition Riemann_fine (a b : R) :=
  within (fun ptd : SF_seq => pointed_subdiv ptd /\
                      SF_h ptd = Rmin a b /\
                      last (SF_h ptd) (SF_lx ptd) = Rmax a b)
         (locally_dist (fun ptd : SF_seq => seq_step (SF_lx ptd))).
```

avec `locally_dist (fun ptd => seq_step (SF_lx ptd))` le filtre sur les `SF_seq` (voir section 3.3.1) permettant d'exprimer la convergence du pas vers 0. Plus précisément, la fonction `locally_dist` prend en argument une fonction `d : R -> T` et renvoie le filtre des ensembles contenant $\mathtt{d}^{-1}(]-\infty; \varepsilon[)$ avec $\varepsilon > 0$. Dans le cas où la fonction `d` est une norme ou plus généralement une fonction positive, comme c'est le cas ici avec la fonction `(fun ptd => seq_step (SF_lx ptd))`, cela correspond à la convergence vers 0.

La fonction `within` permet donc de simuler des filtres sur des sous-ensembles de l'espace de départ. Dans certains cas, ce n'est cependant pas suffisant. Par exemple, plutôt que d'exprimer la propriété "est un sous-ensemble de $\mathbb{R}$ contenant un intervalle de la forme $]\mathtt{a}; \mathtt{a}+\varepsilon[$" par `at_right a`, il peut être utile d'exprimer la propriété "est un voisinage de `a` dans l'ensemble $]\mathtt{a}; +\infty[$". Cela permet en particulier d'adapter des théorèmes, comme le théorème d'échange de limites `filterlim_switch` qui sera développé dans la prochaine section, au cas où seul un sous-ensemble de l'espace considéré vérifie les hypothèses du théorème. J'ai donc été amenée à définir un opérateur semblable à `within`, mais dont le filtre résultant a pour ensemble de base le domaine fourni :

```
Definition subset_filter {T} (F : (T -> Prop) -> Prop) (dom : T -> Prop)
  (P : { x | dom x} -> Prop) : Prop :=
    F (fun x => forall H : dom x, P (existT dom x H)).
```

Même si cela n'apparaît par explicitement dans son énoncé, le théorème d'échange de limite (voir section suivante) a ainsi pu être utilisé pour démontrer le théorème d'échange limite-intégrale alors que les propriétés de convergence attendues ne sont vérifiées que sur l'ensemble des subdivisions pointées, c'est-à-dire un sous-ensemble de `SF_seq`.

Produit de filtre Pour finir, afin de démontrer la continuité de fonctions à deux variables comme l'addition et la multiplication, la notion de filtre produit a été introduite. Du point de vue des voisinages, un voisinage de (u, v) dans $U \times V$ contient le produit d'un voisinage de u dans U et d'un voisinage de v dans V. Le même raisonnement a été utilisé pour construire les filtres produits. Ils ont ainsi été construits comme le produit cartésien des filtres de départ. C'est-à-dire que si on a deux filtres `F` et `G`, alors pour tous ensembles $\mathtt{P} \in \mathtt{F}$ et $\mathtt{Q} \in \mathtt{G}$, l'ensemble $\mathtt{P} \times \mathtt{Q}$ appartient au filtre produit `filter_prod F G`. Cet opérateur conserve la propriété de filtre propre.

4.1.3 Voisinages et espaces uniformes

Comme cela a été évoqué dans la section précédente, les voisinages généralisés n'ont pas été définis dans les espaces métriques, mais dans les espaces uniformes. C'est-à-dire que au lieu de définir la topologie utilisée par les ouverts associés à une distance $d : U \times U \to \mathbb{R}^+$, nous avons choisi de la définir par les ouverts associés à un écart $e : U \times U \to \mathbb{R}^+ \cup \{+\infty\}$. Pour donner un exemple, l'écart sur les espaces de fonctions $e(f,g) = \sup_{x \in X} d(f(x), g(x))$ sert usuellement à définir une topologie pour la convergence uniforme. Si l'on pose $d = \min\{1, e\}$, l'une des distances associées à e, on obtient un espace métrique ayant la même topologie, mais il peut être difficile à manipuler en raison de l'usage de la fonction minimum. Cette difficulté a été rencontrée lors de la démonstration du théorème d'échange de limite :

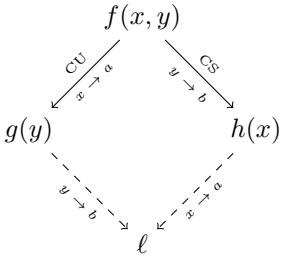
Théorème (`filterlim_switch`)**.** *Soient $f : A \times B \to F$, avec F un espace complet, a un point adhérent à A et b un point adhérent à B.*
Si on suppose que

CU *f converge uniformément vers la fonction $g : B \to F$ quand sa première variable tend vers a,*

CS *f converge simplement vers la fonction $h : A \to F$ quand sa seconde variable tend vers b,*

alors, il existe ℓ dans l'adhérence de F tel que

$$g(y) \underset{y \to b}{\longrightarrow} \ell \quad et \quad h(x) \underset{x \to a}{\longrightarrow} \ell$$



Dans ce théorème, l'hypothèse de convergence uniforme peut être exprimée facilement en utilisant `filterlim` à condition de pouvoir définir des voisinages sur un espace de fonction : `filterlim f (locally a) (locally g)`. Comme je l'ai fait remarquer, utiliser la distance associée à l'écart de la convergence uniforme $d_{\mathbb{R}\to\mathbb{R}}(f,g) = \min_{\overline{\mathbb{R}}}(1; \sup(d_{\mathbb{R}}(f(x), g(x))))$ n'est pas souhaitable car cette distance est difficile à manipuler. De même, utiliser directement l'écart $e(f,g) = \sup(d_{\mathbb{R}}(f(x), g(x)))$ est difficile car il est à valeurs dans $\overline{\mathbb{R}}$. J'ai donc choisi de définir les espaces uniformes par les boules engendrées par l'écart choisi. Ce choix a été guidé par un élément revenant régulièrement dans la démonstration du théorème d'échange de limite : les nombreux énoncés de la forme $d_{\mathbb{R}\to\mathbb{R}}(f,g) < \varepsilon$ étaient systématiquement transformés en $\forall x,\ d_{\mathbb{R}}(f(x), g(x)) < \varepsilon$, avec $\varepsilon \leq 1$, avant de poursuivre la démonstration.

Définition des espaces uniformes

Les espaces uniformes ont donc été définis par la donnée d'un prédicat `ball : T -> R -> T -> Prop` permettant de représenter les boules, ainsi que la démonstration qu'elles représentent les boules engendrées par un écart :

`ball_center :` $\forall x \in \mathtt{T},\ \forall \varepsilon > 0,\ x \in \mathtt{ball}(x, \varepsilon)$

`ball_sym :` $\forall x, y \in \mathtt{T},\ \forall \varepsilon \in \mathbb{R},\ y \in \mathtt{ball}(x, \varepsilon) \Rightarrow x \in \mathtt{ball}(y, \varepsilon)$

`ball_triangle :` $\forall x, y, z \in \mathtt{T},\ \forall \varepsilon_1, \varepsilon_2 \in \mathbb{R},\ y \in \mathtt{ball}(x, \varepsilon_1) \wedge z \in \mathtt{ball}(y, \varepsilon_2) \Rightarrow z \in \mathtt{ball}(x, \varepsilon_1 + \varepsilon_2)$

On peut faire un parallèle entre ces énoncés et certaines propriétés sur les distances : la distance d'un point à lui-même ($\forall x,\ d(x,x) = 0$), la symétrie ($\forall x, y,\ d(x,y) = d(y,x)$) et l'inégalité triangulaire ($\forall x, y, z,\ d(x,z) \leq d(x,y) + d(y,z)$). Il est important de noter que la propriété de séparation ($\forall x, y,\ d(x,y) = 0 \Rightarrow x = y$) n'est pas représentée. Cette propriété n'est en effet pas utile pour démontrer les théorèmes et peut être remplacée, comme en C-CoRN, par la notion de "infiniment proche" ($\forall \varepsilon > 0,\ y \in \mathtt{ball}(x, \varepsilon)$). Même si cela peut entraîner quelques difficultés dans les démonstrations, cela ne pose pas de problème dans les ensembles usuels ($\mathbb{R}$, $\mathbb{C}$, $\mathbb{R}^n$, ...) car cette propriété est alors équivalente à l'égalité.

Pour donner quelques exemples d'espaces uniformes, on peut citer

- l'espace uniforme sur les réels : $\mathtt{ball}_{\mathbb{R}}(x, \varepsilon, y)$ `:=` $|y - x| < \varepsilon$ qui reprend la distance usuelle sur les nombres réels,
- les produits d'espaces uniformes : $\mathtt{ball}_{U \times V}((x,y), \varepsilon, (x', y'))$ `:=` $\mathtt{ball}_U(x, \varepsilon, x') \wedge \mathtt{ball}_V(y, \varepsilon, y')$ avec U et V deux espaces uniformes,
- l'espace uniforme sur les fonctions : $\mathtt{ball}_{U \to V}(f, \varepsilon, g)$ `:=` $\forall x \in U, \mathtt{ball}_V(f(x), \varepsilon, g(x))$ où l'espace d'arrivée V est également muni d'une structure d'espace uniforme.

Sur ces quelques exemples, on peut remarquer que les `ball` ne sont ni des boules fermées, ni des boules ouvertes. En effet, dans $\mathbb{R}$, les `ball` correspondent aux boules ouvertes usuelles de la forme $]x - \varepsilon; x + \varepsilon[$, mais si on regarde l'espace uniforme des fonctions à valeurs dans $\mathbb{R}$, les boules engendrées ne sont ni

ouvertes ni fermées. On a en effet les inclusions strictes $\mathcal{B}(f,\varepsilon) \subset \texttt{ball}(f,\varepsilon) \subset \overline{\mathcal{B}}(f,\varepsilon)$ avec $\mathcal{B}(f,\varepsilon) = \{g | e(f,g) < \varepsilon\}$ et $\overline{\mathcal{B}}(f,\varepsilon) = \{g | e(f,g) \leq \varepsilon\}$. Pour donner un exemple concernant la première inclusion, sur l'intervalle $[0,\varepsilon[$, la fonction identité est dans $\texttt{ball}(x \mapsto 0, \varepsilon)$ car on a bien $\forall x \in [0,\varepsilon[, |x| < \varepsilon$, mais elle n'est pas dans la boule ouverte $\mathcal{B}(x \mapsto 0, \varepsilon)$ car $\sup\{\ |x| \text{ tel que } x \in [0,\varepsilon[\ \} = \varepsilon$. De même, pour la seconde inclusion la fonction constante $x \mapsto \varepsilon$ est dans la boule fermée $\overline{\mathcal{B}}(x \mapsto 0, \varepsilon)$ car $\sup |\varepsilon| = \varepsilon \leq \varepsilon$, mais pas dans $\texttt{ball}(x \mapsto 0, \varepsilon)$ car $|\varepsilon| = \varepsilon \not< \varepsilon$. Cette particularité ne pose cependant pas de problème en pratique car la topologie engendrée par les boules fermées ou ouvertes de rayon strictement positif est la même. La topologie engendrée par les `ball` est également la même en raison des inclusions citées ci-dessus.

Le prédicat `ball` ainsi défini permet d'exprimer les voisinages d'un point :

```
Definition locally {U : UniformSpace} (x : U) (P : U -> Prop) :=
  exists eps : posreal, forall y : U, ball x eps y -> P y.
```

Dans le cas où l'espace uniforme `U` est celui des nombres réels `R`, on retrouve la caractérisation usuelle des voisinages. En effet, dans ce cas le prédicat `ball x eps y` est défini par $|\texttt{y} - \texttt{x}| < \texttt{eps}$. On a alors la propriété qu'un ensemble P est un voisinage de $x \in \mathbb{R}$ si $\exists \varepsilon > 0,\ \forall y \in \mathbb{R},\ |y - x| < \varepsilon \Rightarrow y \in P$.

Comme pour les filtres, nous avons souhaité utiliser un mécanisme permettant de reconnaître automatiquement les espaces uniformes, ainsi que les structures présentées dans la suite de ce chapitre. L'usage des classes de types s'est avéré inadapté pour des raisons de priorités dans la recherche qui entraînent des boucles infinies. Nous nous sommes donc tournés vers les structures canoniques [Sai99]. Ainsi, pour désigner un élément `x` de l'ensemble `U` possédant une structure d'espace uniforme, l'utilisateur peut écrire indifféremment "`x : U`" ou "`x : My_UniformSpace`" où

```
Canonical My_UniformSpace := UniformSpace.Pack U My_UniformSpace_mixin U.
```

avec `My_UniformSpace_mixin` une preuve que l'ensemble `U` possède bien les propriétés d'espace uniforme. L'utilisateur pourra alors utiliser tous les théorèmes démontrés dans le cadre des espaces uniformes. Il est à noter que ce système ne supporte pas la surcharge : un seul espace uniforme peut être associé de manière canonique à un ensemble `U`. Pour définir un second espace uniforme sur le même espace de base `U`, il conviendra alors de remplacer **Canonical** par **Definition** et d'utiliser explicitement `my_UniformSpace` pour désigner l'espace uniforme choisi.

Continuité

Afin d'alléger l'écriture des théorèmes concernant la continuité, la définition suivante a été introduite :

```
Definition continuous {T U : UniformSpace} (f : T -> U) (x : T) : Prop :=
  filterlim f (locally x) (locally (f x)).
```

ainsi que sa version sur un domaine :

```
Definition continuous_on {T U : UniformSpace}
  (D : T -> Prop) (f : T -> U) : Prop :=
    forall x : T, D x -> filterlim f (within D (locally x)) (locally (f x)).
```

Cette définition permet d'exprimer la continuité de fonctions comme arcsin qui est définie et continue sur l'intervalle $[-1;1]$ sans avoir à distinguer le cas de la continuité en -1 et en 1.

La continuité d'un certain nombre de fonctions a été démontrée avant l'introduction de cette définition. Pour les fonctions à une seule variable, cela ne pose pas de problème car l'énoncé utilisant `filterlim` et celui utilisant `continuous` sont interchangeables. Ce n'est cependant pas le cas pour les fonctions à plusieurs variables. En effet, le filtre choisi pour le point vers lequel tend la variable dans les théorèmes déjà définis est `filter_prod (locally x) (locally y)`, alors qu'il s'agit de `locally (x,y)` dans le cas des théorèmes utilisant le prédicat `continuous`. Même si les deux versions sont équivalentes, elles ne sont pas convertibles.

Les espaces uniformes complets

Les espaces uniformes décrits dans la section ci-dessus permettent d'exprimer aussi facilement en Coq l'hypothèse de convergence uniforme "$y \mapsto f(x,y)$ converge uniformément vers g quand x tend vers a", qui s'écrit

$$\texttt{filterlim}\ f\ \texttt{(locally}\ a\texttt{)}\ \texttt{(locally}\ g\texttt{)},$$

que l'hypothèse de convergence simple "$f(x, y)$ converge simplement vers $h(x)$ quand y tend vers b", qui s'écrit

`forall` x`, filterlim (`$f(x)$`) (locally` b`) (locally (`$h(x)$`)).`

Pour démontrer le théorème d'échange de limite présenté au début de la section 4.1.3, il faut de plus pouvoir exprimer la complétude d'un espace uniforme. La caractérisation des espaces complets par la convergence des suites de Cauchy n'a pas été choisie comme définition en raison de son manque de généralité. En effet, avec les définitions présentées jusqu'ici, il est possible d'exprimer l'énoncé "la suite u est une suite de Cauchy" par

```
Definition is_cauchy (u : nat -> U) :=
  forall eps : posreal,
    filter_prod eventually eventually (fun n => ball (u (snd n)) eps (u (fst n))).
```

Cependant cette approche ne permet de caractériser que les espaces séquentiellement complets et n'assure que la convergence des suites. Il est possible de généraliser cette définition "être de Cauchy" afin de caractériser effectivement les espaces de Cauchy :

```
Definition is_cauchy (f : T -> U) (F : (T -> Prop) -> Prop) :=
  forall eps : posreal,
    filter_prod F F (fun n => ball (f (snd n)) eps (f (fst n))).
```

mais cette définition est lourde à utiliser en raison de la présence de `filter_prod`.

La définition "être de Cauchy" retenue dans la bibliothèque Coquelicot est la notion de filtre de Cauchy telle que définie par Bourbaki [Bou71]. Il s'agit des filtres d'un espace uniforme `U` contenant des boules de rayon arbitrairement petit :

```
Definition cauchy (F : (U -> Prop) -> Prop) :=
  forall eps : posreal, exists x : T, F (ball x eps).
```

Après plusieurs pistes étudiées pour caractériser la complétude, la solution retenue est la suivante :

- une fonction totale `lim : ((U -> Prop) -> Prop) -> U` prenant en argument une collection de sous-ensembles d'un espace uniforme `U`, usuellement un filtre, et renvoyant un élément de `U`,
- et un prédicat

  ```
  complete_cauchy : forall F : (T -> Prop) -> Prop,
    ProperFilter F -> cauchy F ->
      forall eps : posreal, F (ball (lim F) eps).
  ```

  qui assure que cet élément est la limite du filtre si le filtre fournit en argument est un filtre de Cauchy.

Cette approche permet de construire, pour tout espace complet, des fonctions totales comme celles décrites dans la section 3.5. En effet, pour tout prédicat `P` vrai pour au plus une valeur, la fonction

```
Definition iota (P : T -> Prop) := lim (fun A => (forall x, P x -> A x)).
```

renvoie, si elle existe, cette unique valeur et une valeur arbitraire sinon.

Notre définition d'un espace complet étant assez éloignée de la formulation usuelle, le lemme suivant a été démontré :

```
Lemma filterlim_locally_cauchy {T : Type} {U : CompleteSpace} :
  forall {F : (T -> Prop) -> Prop} {FF : ProperFilter F} (f : T -> U),
      (forall eps : posreal, exists P : T -> Prop,
         F P /\ forall u v : T, P u -> P v -> ball (f u) eps (f v))
  <-> exists y, filterlim f F (locally y).
```

Dans le cas où `F` est le filtre sur les entiers `eventually`, ce lemme correspond exactement à la caractérisation de la complétude (séquentielle) par les suites de Cauchy. En d'autres termes, toute suite de Cauchy admet une limite. La réciproque est vraie dans tout espace uniforme.

L'espace uniforme des fonctions

Comme cela a été évoqué tout au long de cette section, le choix des espaces uniformes vient de la nécessité d'exprimer la convergence dans les espaces de fonction. L'espace uniforme des fonctions à valeurs dans un espace uniforme a été défini avec le prédicat `ball` suivant :

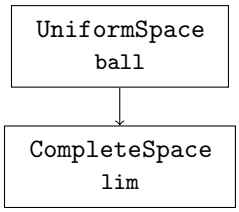


FIGURE 4.1 – Les espaces uniformes dans la bibliothèque Coquelicot. Les boîtes de cette figure représentent les différents espaces formalisés, avec leur nom et les éléments à fournir par l'utilisateur. La flèche $\longrightarrow$ signifie "fait partie de la définition de".

```
Definition fct_ball (T : Type) (U : UniformSpace)
  (x : T -> U) (eps : R) (y : T -> U) :=
    forall t : T, ball (x t) eps (y t).
```

L'ensemble des fonctions à valeurs dans un espace complet définit également un espace complet. La limite d'un filtre `F` sur les fonctions est définie par la fonction qui associe à chaque point `t` de l'espace de départ la limite de la projection du filtre `F` par rapport à `t` :

```
Definition lim_fct {T : Type} {U : CompleteSpace}
  (F : ((T -> U) -> Prop) -> Prop) (t : T) :=
    lim (fun P => F (fun g => P (g t))).
```

Résumé des espaces uniformes

La construction des espaces permettant d'exprimer la convergence dans la bibliothèque Coquelicot est résumée dans la figure 4.1, extraite de la figure 4.3 page 73. Les espaces uniformes `U` sont définis à partir d'un prédicat `ball : U -> R -> U -> Prop` permettant de représenter les boules. Les espaces complets sont quant à eux définis à partir d'un espace uniforme `U` et d'une fonction totale `lim : ((U -> Prop) -> Prop) -> U` renvoyant la limite des filtres de Cauchy. Ces espaces permettent de construire facilement la famille de filtres `locally : U -> (U -> Prop) -> Prop` qui représente les voisinages.

4.2 Généraliser les espaces de nombres

Les espaces uniformes et espaces complets uniformes, présentés dans la section 4.1.3, permettent de construire et d'utiliser facilement des topologies sur des espaces variés, et donc d'y définir la notion de limite. Dans les sections 3.2.3 et 3.3, nous avons vu que les séries et l'intégrale de Riemann peuvent être définies à l'aide du prédicat `filterlim`. Pour généraliser ces deux notions, nous avons besoin de définir des structures algébriques suffisamment générales pour couvrir aussi bien l'ensemble des nombres réels que l'ensemble des matrices complexes. Qui plus est, elles doivent être compatibles avec les espaces uniformes et complets précédemment définis afin de pouvoir exprimer la convergence.

4.2.1 Sommes partielles et modules

Dans un premier temps, il a fallu représenter les sommes intervenant dans la définition des séries, séries entières et intégrales de Riemann :

$$\sum_{k=0}^{n} a_k, \qquad \sum_{k=0}^{n} x^k \cdot a_k \quad \text{et} \quad \texttt{Riemann\_sum}(f,a,b,(\sigma,\xi)) \ \texttt{:=} \ \sum_{i} (\sigma_{i+1}-\sigma_i)\cdot f(\xi_i)$$

Pour les exprimer, nous avons besoin d'une addition et d'un produit. Ces notions sont généralement définies sur un corps ou un espace vectoriel, mais ces deux structures ne permettent pas de prendre en compte les matrices de façon satisfaisante. En effet, pour les séries entières, il est préférable de disposer d'une seule structure permettant de couvrir le cas $\sum a_n \cdot X^n$, avec $a_n \in \mathbb{R}$ et $X \in M_k(\mathbb{R})$, servant à définir les exponentielles de matrices, de la même façon que le cas $\sum z_n \cdot (x_{n+1} - x_n)$, avec $z_n \in \mathbb{C}$ et $x_n \in \mathbb{R}$, servant à définir les intégrales complexes. J'ai donc choisi de définir ces sommes sur des anneaux non commutatifs et des modules. Comme pour les espaces uniformes, des structures canoniques ont été utilisées.

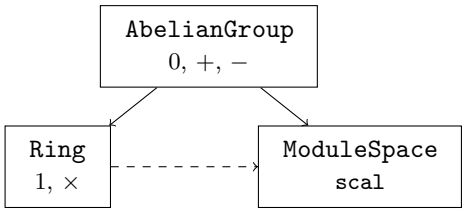


FIGURE 4.2 – Les structures algébriques de la bibliothèque Coquelicot. Les boîtes de cette figure représentent les différents espaces formalisés, avec leur nom et les éléments à fournir par l'utilisateur. Les flèches ⟶ signifient "fait partie de la définition de" et la flèche ⇢ signifie "est paramétré par".

J'ai commencé par la structure de groupe abélien qui intervient à la fois dans la définition d'anneau et dans la définition de module. Un groupe abélien `AbelianGroup` est la donnée d'un élément `zero`, d'une fonction binaire `plus`, d'une fonction unaire `opp`, ainsi que des preuves que ces opérations décrivent bien un groupe abélien :

```
plus_comm : forall x y, plus x y = plus y x ;
plus_assoc : forall x y z, plus x (plus y z) = plus (plus x y) z ;
plus_zero_r : forall x, plus x zero = x ;
plus_opp_r : forall x, plus x (opp x) = zero ;
```

Afin d'obtenir des écritures plus naturelles, la fonction `minus x y := plus x (opp y)` a été définie pour représenter la soustraction. Il est alors possible de définir les sommes partielles pour les séries en généralisant la fonction `sum_n` présentée dans la section 3.2.3 afin qu'elle prenne en argument une suite à valeurs dans un `AbelianGroup`.

Les groupes abéliens ainsi définis sont ensuite utilisés pour construire des anneaux non commutatifs `Ring` en ajoutant un élément `one`, une fonction binaire `mult` et des preuves justifiant que ces éléments définissent bien un anneau non commutatif :

```
mult_assoc : forall x y z, mult x (mult y z) = mult (mult x y) z ;
mult_one_r : forall x, mult x one = x ;
mult_one_l : forall x, mult one x = x ;
mult_distr_r : forall x y z, mult (plus x y) z = plus (mult x z) (mult y z) ;
mult_distr_l : forall x y z, mult x (plus y z) = plus (mult x y) (mult x z) ;
```

On remarque que, comme l'anneau n'est pas commutatif, il faut supposer que `one` est un élément neutre à droite et à gauche et que l'addition `plus` est distributive à droite et à gauche.

Pour finir, un module `ModuleSpace`[3] est la donnée d'un groupe abélien `K`, d'un anneau non commutatif `V`, d'une opération binaire `scal : K -> V -> V` et de la preuve que cela forme bien un module :

```
scal_assoc : forall x y u, scal x (scal y u) = scal (mult x y) u ;
scal_one : forall u, scal one u = u ;
scal_distr_l : forall x u v, scal x (plus u v) = plus (scal x u) (scal x v) ;
scal_distr_r : forall x y u, scal (plus x y) u = plus (scal x u) (scal y u) ;
```

Les structures algébriques ainsi construites ont permis d'exprimer les sommes partielles et sommes de Riemann nécessaires à la généralisation des séries et intégrales de Riemann. En reprenant les définitions de la section 3.2, il a suffi de remplacer respectivement l'addition `Rplus` de `R` par l'addition `plus` des groupes abéliens et la multiplication `Rmult` de `R` par la multiplication par un scalaire `scal` des modules.

La construction des espaces permettant d'exprimer la convergence dans la bibliothèque Coquelicot est résumée dans la figure 4.2, extraite de la figure 4.3 page 73. La base de cette bibliothèque est la notion de groupe abélien `G : AbelianGroup` définie à partir d'une opération binaire `plus : G -> G -> G` correspondant à l'addition, d'une opération unaire `opp : G -> G` correspondant à l'opposé et d'un élément neutre `zero : G` pour l'opération binaire. Les groupes abéliens ainsi définis sont utilisés pour définir les anneaux non commutatifs `K : Ring` en ajoutant une opération binaire `mult : K -> K -> K` représentant la multiplication et un élément neutre `one` pour cette opération binaire. Les modules `V : ModuleSpace K` sont définis comme des groupes abéliens munis d'une multiplication `scal : K -> V -> V` par un scalaire à valeur dans un anneau `K : Ring`. Il est important de noter qu'un anneau vérifie les propriétés de module avec `scal := mult`. Ainsi tous les théorèmes démontrés sur les modules peuvent être appliqués au cas

3. Le mot `Module` étant réservé en Coq, la structure définie ci-dessus n'a pas pu prendre ce nom.

particulier des anneaux. Afin de simplifier la définition des instances pour $\mathbb{R}$ et $\mathbb{C}$, il a été démontré que les anneaux non commutatifs `Ring` sont également des `ModuleSpace`.

4.2.2 Convergence dans un module

Afin d'achever la généralisation des séries et de l'intégrale de Riemann, il a fallu munir les structures algébriques précédemment construites d'une structure d'espace uniforme. Cela permet en effet de définir facilement des filtres sur ces structures algébriques.

La première approche consistant à ajouter des propriétés au prédicat `ball` a rapidement été abandonnée. En effet des propriétés comme

```
ball_plus_r : forall x y z eps,
  ball x eps y -> ball (plus x z) eps (plus y z)
ball_opp : forall x y eps,
  ball x eps y -> ball (opp x) eps (opp y)
```

n'auraient pas été faciles à utiliser en pratique. Nous nous sommes donc tournés vers les anneaux munis d'une valeur absolue `AbsRing` et les modules normés `NormedModule`.

La caractérisation choisie pour la valeur absolue `abs` sur les anneaux est inspirée de la semi-valeur absolue de Bourbaki [Bou74] :

```
abs_zero : abs zero = 0.
abs_opp_one : abs (opp one) = 1.
abs_triangle : forall x y : K, abs (plus x y) <= abs x + abs y.
abs_mult : forall x y : K, abs (mult x y) <= abs x * abs y.
```

La seule condition supplémentaire par rapport à la définition de Bourbaki est `abs_opp_one` qui implique que l'anneau n'est pas dégénéré (`zero` $\neq$ `one`). La norme `norm` sur les modules a été caractérisée de façon similaire :

```
norm_triangle : forall (x y : V), norm (plus x y) <= norm x + norm y.
norm_scal : forall (l : K) (x : V), norm (scal l x) <= abs l * norm x.
```

Nous avons tenté dans un premier temps d'utiliser la valeur absolue des anneaux et la norme des modules pour définir les `ball` de la même façon que pour les nombres réels : `ball`(x, ε, y) `:=` $|y - x| < \varepsilon$. Cette définition est facile à mettre en place et convient bien aux anneaux munis d'une valeur absolue, mais lors de la définition des nombres complexes (`C := R * R`, voir section 4.2.3), la valeur absolue obtenue ne correspondait pas à celle usuelle. En effet, `C` était automatiquement reconnu comme un module sur l'anneau `R` et donc la norme permettant de respecter la définition de `ball` était $|x+i\cdot y|_{\mathtt{C}} = \max\{|x|_{\mathtt{R}}; |y|_{\mathtt{R}}\}$ au lieu du module $\sqrt{x^2+y^2}$ attendu.

Nous avons donc choisi de séparer la construction de la norme de celle de l'espace uniforme. Les modules normés ont été définis comme un ensemble `T` possédant à la fois une structure de module sur un anneau muni d'une valeur absolue `K : AbsRing` et une structure d'espace uniforme. La compatibilité entre l'espace uniforme déduit de la norme et celui fourni en paramètre est assurée par le réel `norm_factor : R` et les deux lemmes suivant :

```
norm_compat1 : forall (x y : V) (eps : R), norm (minus y x) < eps -> ball x eps y.
norm_compat2 : forall (x y : V) (eps : posreal),
  ball x eps y -> norm (minus y x) < norm_factor * eps.
```

La norme choisie par défaut pour définir le produit de deux `K`-modules normés est $\|(x,y)\|_{U\times V} = \sqrt{\|x\|_U^2 + \|y\|_V^2}$. Cela nous permet d'obtenir dans le cas du `R`-module normé `C`, la norme correspondant au module usuel $|x + i\cdot y|_{\mathtt{C}} = \sqrt{|x|_{\mathtt{R}}^2 + |y|_{\mathtt{R}}^2}$.

Pour finir, les modules complets normés `CompleteNormedModule` ont été définis afin de démontrer des théorèmes comme l'échange de limite et d'intégrale. Ils sont définis de la même façon que les espaces uniformes complets, c'est-à-dire comme des modules normés possédant les propriétés d'espace complet.

La continuité des opérations arithmétiques `plus`, `opp`, `mult` et `scal` a été démontrée en utilisant le prédicat `filterlim`. Par exemple, pour l'opposé et l'addition, on a

```
Lemma filterlim_opp {K : AbsRing} {V : NormedModule K} :
  forall x : V,
  filterlim opp (locally x) (locally (opp x)).
Lemma filterlim_plus {K : AbsRing} {V : NormedModule K} :
  forall x y : V,
  filterlim (fun z : V * V => plus (fst z) (snd z))
```

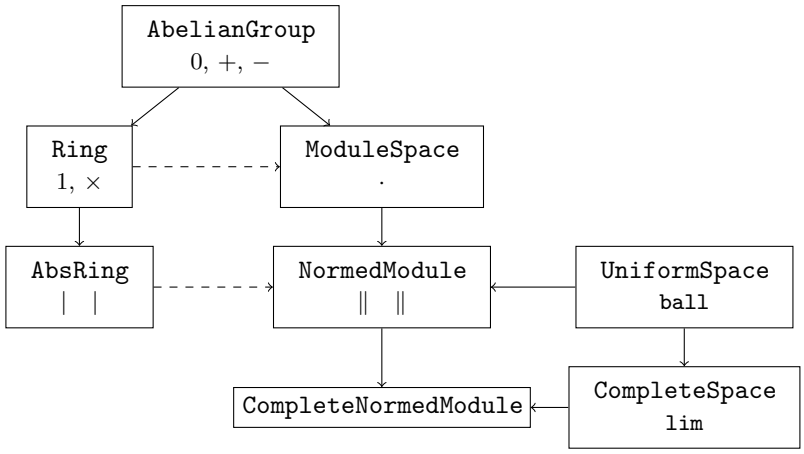


FIGURE 4.3 – La hiérarchie algébrique de la bibliothèque Coquelicot. Les boîtes de cette figure représentent les différents espaces formalisés, avec leur nom et les éléments à fournir par l'utilisateur. Les flèches $\longrightarrow$ signifient "fait partie de la définition de" et les flèches $\dashrightarrow$ signifient "est paramétré par".

```
(filter_prod (locally x) (locally y))
(locally (plus x y)).
```

Les anneaux munis d'une valeur absolue étant également des modules normés, ces théorèmes s'appliquent aussi à eux. De la même façon, le théorème de continuité de la fonction `mult` est un cas particulier de la continuité de la fonction `scal`. La continuité de la norme et de la valeur absolue ont également été démontrées.

Ces théorèmes sont plus simples à démontrer que ceux de la forme "si $\lim_a f = \ell_f$ et $\lim_a g = \ell_g$, alors $\lim_a (f + g) = \ell_f + \ell_g$" et permettent de déduire ces derniers en quelques lignes (moins de 5 lignes dans la plupart des cas) par application d'un théorème de composition.

4.2.3 La hiérarchie algébrique

La hiérarchie algébrique formalisée dans la bibliothèque Coquelicot est résumée dans la figure 4.3. Pour achever la construction des structures algébriques simples, une valeur absolue `abs : K -> R` est ajoutée aux anneaux pour définir `AbsRing`. Grâce à l'usage des structures canoniques, il est possible d'utiliser les théorèmes concernant les groupes abéliens `AbelianGroup` pour démontrer des résultats sur les anneaux `Ring` et les anneaux munis d'une valeur absolue `AbsRing`. Il est également possible d'utiliser les théorèmes concernant les anneaux `Ring` pour démontrer des théorèmes sur les anneaux munis d'une valeur absolue `AbsRing`.

Les modules normés `V : NormedModule` sont définis comme la donnée d'un module sur un anneau muni d'une valeur absolue avec une norme `norm : V -> R` et d'un espace uniforme définissant la même topologie. Ils sont donc à la fois des modules et des espaces uniformes. Les anneaux munis d'une valeur absolue sont également des modules normés avec `norm := abs`. Les anneaux munis d'une valeur absolue définissent donc aussi un espace uniforme dont les boules sont définies par défaut par `ball`(x, ε, y)`:= abs`$(y - x) < \varepsilon$. Pour finir, les modules normés complets ont été définis en ajoutant la propriété de complétude à un module normé.

Des instances concernant les réels `R`, les complexes `C` et les matrices `matrix` sont fournies afin de pouvoir utiliser les versions générales des théorèmes sur ces nombres.

Les nombres réels

Ainsi, les nombres réels `R` vérifient les propriétés d'un anneau muni d'une valeur absolue `AbsRing`, avec les opérations arithmétiques et la valeur absolue de la bibliothèque standard. La structure de module normé complet `CompleteNormedModule` sur `R` a également été définie en utilisant la fonction totale `Lub_Rbar`, présentée dans la section 3.5, qui renvoie la borne supérieure d'un ensemble de réels.

```
Definition R_complete_lim (F : (R -> Prop) -> Prop) : R :=
  Lub_Rbar (fun x : R => F (ball (x + 1) 1)).
```

Même si cette fonction respecte les propriétés attendues pour la fonction `lim`, elle ne permet pas pour autant de démontrer les lemmes de réécriture sans hypothèses de la section 3.5.

Les nombres complexes

J'ai défini l'ensemble des nombres complexes comme l'ensemble des paires de réels `R * R`. Les opérations arithmétiques ont été définies de façon usuelle :

```
Definition Cplus (x y : C) : C := (fst x + fst y, snd x + snd y).
Definition Copp (x : C) : C := (-fst x, -snd x).
Definition Cmult (x y : C) : C :=
  (fst x * fst y - snd x * snd y, fst x * snd y + snd x * fst y).
Definition Cinv (x : C) : C :=
  (fst x / (fst x ^ 2 + snd x ^ 2), - snd x / (fst x ^ 2 + snd x ^ 2)).
```

Pour en faciliter l'usage, les notations `+`, `-`, `*` et `/` ont été introduites pour représenter les différentes opérations arithmétiques. Une coercion des nombres réels dans les nombres complexes a également été définie.

J'ai démontré que les nombres complexes `C` sont à la fois un anneau muni d'une valeur absolue (et donc un `C`-module normé) et un `R`-module normé. Il faut noter que, en tant que `C`-module normé, l'espace uniforme n'est pas celui par défaut pour les anneaux, mais celui de $\mathbb{R}^2$:

$$\texttt{ball}(x + i \cdot y, \varepsilon, x' + i \cdot y') \texttt{ := ball}(x, \varepsilon, x') \wedge \texttt{ball}(y, \varepsilon, y').$$

Ce choix a été fait afin d'obtenir la même valeur absolue et le même espace uniforme pour le `C`-module normé et le `R`-module normé. La complétude de `C` n'est actuellement pas démontrée, mais elle peut être déduite de celle de `R`.

Les matrices

Une formalisation de l'ensemble des matrices a été initiée dans la bibliothèque Coquelicot. Elles ont été définies comme des vecteurs de vecteurs :

```
Definition matrix {T : Type} (m n : nat) := Tn m (Tn n T).
```

avec `Tn : nat -> Type -> Type` le type des vecteurs. Les constantes et opérations de base ont été définies sur les matrices. J'ai démontré les propriétés suivantes :

- Pour tout $n, m \in \mathbb{N}$, l'ensemble des matrices de n lignes et m colonnes sur un groupe abélien forme également un groupe abélien.
- Pour tout $n \in \mathbb{N}$, l'ensemble des matrices carrées de taille n sur un anneau non commutatif forme un anneau non commutatif.
- Pour tout $n, m \in \mathbb{N}$, l'ensemble des matrices de n lignes et m colonnes sur un anneau est un module sur l'anneau des matrices carrées de taille m.

4.3 Généraliser la notion de dérivée

Les notions généralisées jusqu'ici reposent sur la notion de limite, et ont donc pu être généralisées grâce au prédicat `filterlim`. Pour la différentiabilité, cela n'est cependant possible que dans le cadre de l'analyse sur un corps muni d'une valeur absolue. La différentiabilité se confond alors avec la dérivabilité et la définition est comme dans l'analyse réelle : une fonction f est dérivable/différentiable en x s'il existe ℓ tel que

$$\lim_{y \to x} \frac{f(y) - f(x)}{y - x} = \ell.$$

La différentielle est alors $x \mapsto \ell \cdot x$. Cette définition n'est cependant pas suffisante pour traiter la différentiabilité dans un espace vectoriel, et encore moins dans les anneaux et les modules que nous avons formalisés car ils ne disposent pas d'une division.

Une autre définition de la différentiabilité utilisant la notion de limite est : une fonction f est différentiable en x s'il existe une application linéaire continue L telle que

$$\lim_{y \to x} \frac{\|f(y) - f(x) - L(y - x)\|}{\|y - x\|} = 0.$$

Cette définition est formalisable en utilisant ce qui a été défini dans les sections précédentes, mais elle impose de gérer l'hypothèse $\|y - x\| \neq 0$. Cela est facilement exprimable grâce au modificateur de filtre `within` mais peut s'avérer très lourd dans les démonstrations.

La définition finalement choisie est : une fonction f est différentiable en x s'il existe une application linéaire continue L telle que

$$f(y) - f(x) - L(y - x) \in o_{y \to x}\,(y - x)\,.$$

L'ensemble $o_{y \to x}\,(y - x)$ des fonctions négligeables devant $y \mapsto y - x$ au voisinage de x étant un espace vectoriel, cette approche permet de simplifier de nombreuses démonstrations sur la différentiabilité.

J'ai donc formalisé la notion de prépondérance, ainsi que la notion d'application linéaire.

4.3.1 La relation de prépondérance

Dans le cadre de l'analyse réelle, la relation de prépondérance est usuellement définie de la façon suivante : une fonction g est négligeable devant f au voisinage d'un point a, noté $g \in o_a(f)$ ou $g = o_a(f)$, si

$$\forall \varepsilon > 0,\ \exists \delta > 0,\ |x - a| < \delta \Rightarrow |g(x)| \leq \varepsilon \cdot |f(x)|\,,$$

ce qui correspond exactement à $\forall \varepsilon > 0,\ \left\{x \in \mathbb{R} \;\middle|\; |g(x)| \leq \varepsilon \cdot |f(x)|\right\} \in \texttt{locally}(a)$. La généralisation des équivalents aux modules normés a donc été traduite par

```
Definition is_domin {T : Type} {Ku Kv : AbsRing}
  {U : NormedModule Ku} {V : NormedModule Kv}
    (F : (T -> Prop) -> Prop) (f : T -> U) (g : T -> V) :=
      forall eps : posreal, F (fun x => norm (g x) <= eps * norm (f x)).
```

J'ai démontré que cette relation est une relation d'ordre strict sur l'ensemble des fonctions, ainsi que quelques théorèmes concernant son comportement par rapport aux opérations usuelles afin de simplifier les démonstrations de différentiabilité.

Dans l'idée de poursuivre ultérieurement le développement de cette théorie, la relation d'équivalence $f \sim_{\texttt{F}} g$ a été définie par $g - f \in o_{\texttt{F}}(g)$. J'ai démontré que c'est une relation d'équivalence, ainsi que quelques théorèmes concernant les opérations de base et l'influence sur les limites.

4.3.2 Fonctions linéaires

Les fonctions linéaires continues ont été définies par le prédicat `is_linear` permettant de les caractériser :

```
Record is_linear {K : AbsRing} {U V : NormedModule K} (l : U -> V) :=
  { linear_plus : forall x y : U, l (plus x y) = plus (l x) (l y);
    linear_scal : forall (k : K) (x : U), l (scal k x) = scal k (l x);
    linear_norm : exists M : R,
                    0 < M /\ (forall x : U, norm (l x) <= M * norm x) }.
```

En plus des hypothèses usuelles de linéarité, cette caractérisation donne également une propriété sur la norme afin d'assurer la continuité de ces fonctions linéaires. Cette propriété est indispensable pour démontrer que toute fonction différentiable est continue. Dans le cas où le module normé est de dimension finie, cette hypothèse est superflue car elle peut être déduite de la linéarité. Cette approche permet de traiter avec la même définition le cas des espaces de dimension infinie.

La linéarité des fonctions intervenant dans la différentiabilité des fonctions de base a été démontrée. Il faut cependant remarquer que la fonction `scal` n'est linéaire par rapport à la seconde variable que dans le cas où l'anneau est commutatif. Cela entraîne pour le moment des hypothèses supplémentaires dans les théorèmes traitant de la différentiabilité de `scal` et de `mult`. Une solution pourrait être d'ajouter la structure d'anneau commutatif à la hiérarchie existante.

4.3.3 Différentiabilité et dérivabilité

Pour finir, j'ai défini le prédicat caractérisant le fait qu'une fonction `f` admet pour différentielle la fonction `l` sur un filtre `F` :

```
Definition filterdiff {K : AbsRing} {U V : NormedModule K}
  (f : U -> V) (F : (U -> Prop) -> Prop) (l : U -> V) :=
is_linear l /\ (forall x : U, is_filter_lim F x ->
 is_domin F (fun y : U => minus y x)
   (fun y : U => minus (minus (f y) (f x)) (l (minus y x)))).
```

avec `is_filter_lim F x` un prédicat signifiant que `x` est une limite du filtre `F`. Cette notion n'avait pas été utile jusqu'à présent, mais elle est indispensable pour définir la différentiabilité afin d'exprimer avec la même définition les dérivées usuelles, c'est-à-dire avec `F := locally x`, et les dérivées à droite et à gauche.

Le cas particulier de la dérivabilité a également été formalisé :

```
Definition is_derive {K : AbsRing} {V : NormedModule K}
  (f : K -> V) (x : K) (l : V) :=
    filterdiff f (locally x) (fun y => scal y l).
```

Peu de théorèmes concernant la différentiabilité et la dérivabilité ont été démontrés. On peut cependant citer la différentiabilité des fonctions linéaires et des opérations de base, découlant pour la plupart de la linéarité. Le théorème disant qu'une fonction différentiable est continue a également été démontré.

Chapitre 5

Applications

J'ai présenté dans les chapitres 3 et 4 une formalisation de l'analyse réelle ainsi que sa généralisation à des modules normés. À chaque étape, cette formalisation a été testée sur des exemples simples, mais aussi sur des applications plus complexes présentées dans ce chapitre, afin de vérifier son utilisabilité. Les limites généralisées, dérivées et intégrales ont été testées sur les exercices d'analyse du Baccalauréat de Mathématiques en 2013 (section 5.1). Les séries entières ont été utilisées pour vérifier des relations sur les fonctions de Bessel (section 5.2). Et pour finir, les dérivées, intégrales et la tactique permettant d'automatiser les preuve de dérivabilité ont été testées sur la formule de d'Alembert (section 5.3).

5.1 Coq passe le Bac

Le programme d'analyse de terminale S couvre les bases de l'analyse réelle. En effet, un élève de fin de terminale est censé connaître les notions de limites de suites et de fonctions, de dérivées et d'intégrales, ainsi qu'un certain nombre de théorèmes usuels concernant ces notions comme le théorème des valeurs intermédiaires. Les lycéens ayant choisi l'option mathématique disposent également de résultats sur les matrices réelles. J'ai résumé les principaux points de ce programme dans la première colonne du tableau 5.1. L'état de la bibliothèque au moment de l'expérience correspond à la colonne Coquelicot 1.0. À ce moment-là, une bonne partie du programme de terminale était déjà traitée avec les limites généralisées à tout voisinage de $\overline{\mathbb{R}}$ développées dans la section 3.2.2. Les parties manquantes ont été ajoutées à Coquelicot 2.0 en même temps que la généralisation des limites aux filtres présentés dans la section 4.1.2 et les démonstrations correspondant au nouveau formalisme sont disponibles dans le dossier "examples" de Coquelicot.

Le Baccalauréat de Mathématiques visant à évaluer les connaissances de ces élèves, il constitue également un bon moyen de vérifier qu'une bibliothèque d'analyse réelle comme Coquelicot contient une base raisonnable de théorèmes.

Un autre avantage à tester la bibliothèque Coquelicot sur le Baccalauréat est de vérifier sa facilité d'utilisation. En effet, les exercices proposés étant relativement simples, les démonstrations doivent également être simples et faciles à lire. Une façon de valider la simplicité des démonstrations est de les présenter à des personnes qui ne sont pas familières avec Coq.

J'ai donc contacté M. Michalak, l'un des inspecteurs principaux d'éducation de l'académie de Versailles, pour lui proposer l'expérience. Il a arrangé mon accueil au Lycée Parc de Vilgénis à Massy pour passer le Baccalauréat le matin du 20 juin 2013, ainsi qu'un entretien avec des enseignants de mathématiques l'après-midi pour présenter mes résultats [1].

5.1.1 Méthodologie

Le Baccalauréat de Mathématiques 2013 [Bac13] était constitué de 3 exercices communs à tous les élèves et un quatrième exercice différent suivant qu'ils ont suivi l'option mathématique ou non :

- Exercice 1 : Probabilités.
- Exercice 2 : Étude de fonction.
- Exercice 3 : Géométrie.
- Exercice 4 - autres spécialités : Étude de suite de réels.
- Exercice 4 - spécialité mathématiques : Matrices et étude de suite.

1. Remerciement : M. Michalak et Mme Roudneff pour l'organisation de cette journée, Mme Guérin et M. Biset pour leur accueil, et M. Taillet, Mme Levi, Mme Marquier, M. Jospin et M. Gabilly pour leur attention et leurs remarques.

Programme de terminale	`Reals`	**Coquelicot**	
		1.0	**2.0**
Limites de suites	$\mathbb{R} \cup \{+\infty\}$	$\overline{\mathbb{R}}$	$\overline{\mathbb{R}}$
Opérations	$\mathbb{R}$	$\overline{\mathbb{R}}$	$\overline{\mathbb{R}}$
Suites géométriques	Non	Oui	Oui
Limites de fonctions	$\mathbb{R}$	$\overline{\mathbb{R}}$	$\overline{\mathbb{R}}$
Opérations	$\mathbb{R}$	$\overline{\mathbb{R}}$	$\overline{\mathbb{R}}$
Valeurs intermédiaires	$\mathbb{R}$	$\overline{\mathbb{R}}+ \nearrow\nearrow$	$\overline{\mathbb{R}}$
Dérivées	Oui	Oui	Oui
Opérations	Oui	Oui	Oui
Sens de variation	Oui	Oui	Oui
Fonctions de référence	Oui	Oui	Oui
Dérivées	Oui	Oui	Oui
Limites	$\mathbb{R}$	$\overline{\mathbb{R}}$	$\overline{\mathbb{R}} \cup \{a^{\pm}\}$
$\lim_{x\to+\infty} e^x/x$, $\lim_{x\to0^+} x\ln x$, ...	Non	$\overline{\mathbb{R}}$	$\overline{\mathbb{R}} \cup \{a^{\pm}\}$
Intégration	Riemann	Riemann	Riemann
Linéarité, positivité, Chasles	Oui	Oui	Oui
Primitives de $u'e^u$, $u'u^n$, ...	Non	Oui	Oui
Matrices			
Matrices carrées et vecteurs	Non	Non	Oui
Opérations arithmétiques	Non	Non	Oui

TABLEAU 5.1 – Coq et le programme d'analyse de terminale S 2013

Lors de l'expérience en juin 2013, la bibliothèque Coquelicot ne permettait de traiter que les exercices 2 et 4 - autres spécialités. L'ajout des matrices à la bibliothèque Coquelicot m'a permis depuis de traiter l'exercice 4 - spécialité mathématiques.

En ce qui concerne la résolution des exercices, on peut classer les questions en deux catégories :

1. les questions du type "démontrer que" pour lesquelles l'énoncé à démontrer est donné par l'exercice,
2. les questions du type "déterminer" pour lesquelles il faut d'abord résoudre l'exercice afin de pouvoir établir l'énoncé à démontrer.

Qui plus est, pour certaines questions, il a été indispensable d'adapter l'énoncé pour pouvoir appliquer les théorèmes de la bibliothèque. En effet, l'objectif étant de présenter des démonstrations convaincantes à des non-spécialistes l'après-midi même, je n'avais pas le temps de démontrer des théorèmes trop compliqués mais je ne pouvais pas non plus me permettre de présenter un travail incomplet. J'ai donc démontré des versions équivalentes aux énoncés de l'épreuve quand ils ne pouvaient pas être exprimés avec la version 1.0 de la bibliothèque Coquelicot.

5.1.2 Exercice 2

Cet exercice avait pour objet l'étude de la fonction

$$f(x) = \frac{a + b\ln x}{x}.$$

Il a été traité en environ 2 heures et 280 lignes de preuve. Cet exercice couvrant une bonne partie du programme sur l'étude de fonctions, il a permis de mettre en évidence les forces et les faiblesses de la bibliothèque Coquelicot à ce moment-là.

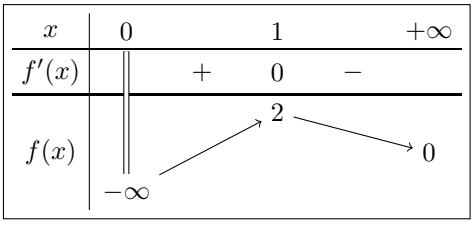
La détermination des valeurs de a et b à partir d'une lecture graphique faisait l'objet de la première série de questions. J'ai donc défini la fonction `fab (a b : R) : R -> R` permettant de représenter la fonction f avec `a` et `b` arbitraires. La lecture graphique nous permettant de déduire que $f(1) = 2$ et $f'(1) = 0$. Après avoir démontré la valeur de la dérivée sur $\mathbb{R}_+^*$, j'ai démontré la valeur des deux réels `a` et `b` :

```
Lemma Val_a_b (a b : R) : fab a b 1 = 2 -> Derive (fab a b) 1 = 0
  -> a = 2 /\ b = 2.
```

Même si la démonstration de problèmes de ce type n'est pas forcément immédiate en Coq, la bibliothèque standard contient les théorèmes d'arithmétique réelle suffisants pour cette démonstration de 17 lignes. Qui

plus est, la démonstration de la différentiabilité de f s'est faite sans problème (13 lignes) par application de théorèmes de la bibliothèque standard sur la dérivabilité. J'ai fait le choix de ne pas utiliser la tactique `auto_derive` (voir la section 3.6 pour les détails) afin de souligner le potentiel pédagogique de Coq pour l'apprentissage de la dérivabilité. Pour la suite de l'exercice, j'ai défini la fonction `f : R -> R := fab 2 2` correspondant à la fonction représentée sur le graphique.

La deuxième série de questions avait pour but d'établir le tableau de variation complet de la fonction f :



La bibliothèque Coquelicot était bien adaptée pour la limite en $+\infty$ et pour faire le lien entre croissance et dérivée positive. Cette partie a cependant mis en évidence des concepts manquants. Tout d'abord, la bibliothèque 1.0 ne disposait pas encore de limites à droite. J'ai donc dû calculer la limite en 0 de la fonction $g : x \mapsto f(|x|)$ qui coïncide avec la fonction f sur $\mathbb{R}^+$ et qui admet une limite en 0. Ce point a été corrigé par l'introduction après le Baccalauréat de `filterlim` qui généralise la notion de limite et de la famille de filtres `at_right` permettant d'exprimer la notion de limites à droite (voir section 3.2.2). Ensuite, l'utilisation de la fonction `sign` disponible dès Coquelicot 1.0, pour faire le lien entre le signe de la dérivée et celui de $-\ln x$, a permis d'écrire des théorèmes plus lisibles, mais a conduit à résoudre de nombreuses inégalités. Cela a donné une démonstration longue et assez indigeste. La démonstration de théorèmes supplémentaires suite au Baccalauréat, comme le fait que le signe d'un produit est le produit des signes, m'a permis de fortement simplifier les inégalités à résoudre, passant de 36 à 17 lignes pour démontrer l'égalité des signes. Des théorèmes sur les opérations arithmétiques dans $\overline{\mathbb{R}}$, comme $(+\infty) \times x = (+\infty)$ si $x > 0$, ont également été améliorés pour limiter le nombre d'inégalités générées.

La troisième série de questions était consacrée à établir l'existence et le nombre de solutions de l'équation $f(x) = 1$. Seule l'existence a été démontrée pour chacun des intervalles $]0; 1]$ et $[1; +\infty[$ en utilisant le théorème des valeurs intermédiaires. Comme celui-ci ne concernait que les fonctions croissantes admettant une limite à chacune des bornes de l'intervalle, il a fallu, comme pour la deuxième série de questions, faire appel à la fonction g pour la limite en 0 et appliquer le théorème de façon judicieuse à $x \mapsto -f(x)$ pour l'intervalle $[1; +\infty[$. L'unicité n'a pas été traitée pour des raisons de temps. Il est cependant possible de la démontrer en utilisant la monotonie stricte de la fonction f sur les deux intervalles.

La quatrième série de questions n'entrait pas dans le cadre de l'expérience : elle étudiait un algorithme d'approximation de l'une des solutions de $f(x) = 1$. Je ne l'ai donc pas traitée pour me concentrer sur les questions d'analyse réelle.

Pour finir, la cinquième série de questions concernait le calcul de l'intégrale de f entre e^{-1} et 1. Cette question, en dehors des nombreux problèmes de positivité générés par les théorèmes concernant la fonction logarithme, a été facilement traitée en utilisant la remarque donnée par l'énoncé qui permettait de déduire que $2 \ln y + \ln^2 y$ est une primitive de f.

5.1.3 Exercice 4 - autres spécialités

Cet exercice avait pour but l'étude de la suite (u_n) définie par

$$u_0 = 2 \quad \text{et} \quad u_{n+1} = \frac{2}{3} u_n + \frac{1}{3} n + 1.$$

Les démonstrations m'ont demandé une heure et ont nécessité 141 lignes de preuve.

La première série de questions de cet exercice concernait le calcul numérique des premières valeurs de la suite (u_n) ainsi qu'une conjecture sur le sens de variation de cette suite. Comme pour l'algorithme de l'exercice 2, cette question n'a pas été traitée pour des raisons de temps.

La deuxième série de questions avait pour but de confirmer la conjecture en démontrant que la suite était croissante. Cette démonstration se fait à l'aide de deux questions préliminaires : d'une part montrer que $\forall n,\ u_n \leq n + 3$, et d'autre part que $\forall n,\ u_{n+1} - u_n = (n + 3 - u_n)/3$. Le premier point est démontré par récurrence en une dizaine de lignes sans difficulté. Le second est juste une résolution d'égalité à l'aide de la tactique `field`.

La troisième série de questions avait pour but de déterminer la limite de la suite (u_n) :

```
Lemma Q3c : is_lim_seq u p_infty.
```

Cela a été fait en utilisant la suite auxiliaire $v_n = u_n - n$ donnée par le sujet. Cette suite étant géométrique, cela permet d'obtenir une expression en fonction de n de la suite (u_n) et ainsi de calculer sa limite en appliquant les théorèmes sur les limites disponibles dans la bibliothèque Coquelicot. Le nombre d'opérations intervenant dans ce calcul de limite étant faible, les conditions de bord générées par les théorèmes sur les limites pour les opérations arithmétiques, comme `is_lim_seq_plus`, sont peu nombreuses.

La dernière série de questions avait pour but d'étudier les suites

$$S_n = \sum_{k=0}^{n} u_k \quad \text{et} \quad T_n = \frac{S_n}{n^2}.$$

Comme pour l'étude de la suite (u_n), il était demandé de trouver une forme explicite de la suite (S_n) :

```
Lemma Q4a : forall n, Su n = 6 - 4 * (2/3)^n + INR n * (INR n + 1) / 2.
```

puis de s'en servir pour calculer la limite de la suite (T_n). Le calcul des S_n s'est fait assez facilement à l'aide de la bibliothèque standard. Concernant la limite de la suite (T_n), celle-ci étant moins évidente que les autres limites demandées jusqu'ici, je me suis aidée des capacités de calcul de Coq pour la déterminer. J'ai donc commencé par écrire l'énoncé avec une valeur arbitraire, puis j'ai fait la démonstration pour permettre à Coq de "calculer" la limite à partir des différents théorèmes sur les limites. Il restait ensuite à simplifier la limite calculée à l'aide de la tactique `field_simplify` et reporter le résultat dans l'énoncé pour terminer la démonstration. Les théorèmes de la bibliothèque Coquelicot étaient suffisants, mais les conditions de bord malheureusement lourdes et répétitives.

5.1.4 Exercice 4 - spécialité mathématiques

Cet exercice avait pour but de déterminer la limite de deux suites v et c mutuellement récursives représentant respectivement la population d'une ville et de sa campagne. Il n'avait pas été traité le jour du Baccalauréat en raison de l'usage de matrices. La généralisation de la bibliothèque Coquelicot a conduit à formaliser les matrices et à définir les opérations arithmétiques sur celles-ci pour en faire un anneau non commutatif (voir section 4.2). Même si l'analyse sur l'anneau des matrices n'est pas encore très développée, l'arithmétique est maintenant suffisante pour pouvoir résoudre cet exercice.

Les deux premières questions avaient pour but de déterminer la relation de récurrence liant les suites v et c, et de l'exprimer sous forme matricielle :

$$X_{n+1} = AX_n, \quad \text{avec } X_n = \begin{pmatrix} v_n \\ c_n \end{pmatrix} \text{ et } A = \begin{pmatrix} \frac{95}{100} & \frac{1}{100} \\ \frac{5}{100} & \frac{99}{100} \end{pmatrix}.$$

La matrice A étant donnée, la principale difficulté de ces deux questions était l'écriture des matrices considérées. En effet, la définition du type `matrix` (vecteurs de vecteurs) conduit à une représentation lourde :

```
Definition A : matrix 2 2 :=
  ((95/100, (1/100, tt)) ,
   ((5/100, (99/100, tt)),tt)).
```

J'ai résolu ce problème en introduisant une notation pour les vecteurs :

```
Definition A : matrix 2 2 :=
  [[ 95/100,  1/100 ] ,
   [  5/100, 99/100 ]].
```

Pour plus de clarté, j'ai précisé le type `matrix 2 2` de la matrice A. En effet, sans cette précision, Coq aurait interprété le type de cet objet par `R * (R * unit) * (R * (R * unit) * unit)`.

La troisième question avait pour but de calculer la matrice A^n. Pour cela, deux matrices P et Q étaient données pour permettre la diagonalisation de la matrice A. Le plus difficile dans cette question était la manipulation des différentes opérations arithmétiques pour vérifier les différentes égalités de matrice. La démonstration de l'inversibilité des matrices de passage a demandé 25 lignes de Coq, la diagonalisation a demandé 7 lignes de Coq, et pour finir la valeur de A^n a demandé 4 lignes.

Pour finir, la dernière question avait pour but le calcul des limites des suites v et c. La forme explicite de la suite v étant donnée, la difficulté de cette question était similaire au calcul de la suite T des élèves n'ayant pas suivi l'option mathématiques. Même si ce n'était pas demandé dans l'énoncé, j'ai démontré que la forme explicite donnée est correcte en 20 lignes de Coq pour un travail plus rigoureux. Le calcul des limites a demandé quant à lui 39 lignes de Coq

5.1.5 L'entretien avec les enseignants

L'après-midi a été consacrée à un entretien avec cinq enseignants de mathématiques provenant de différents lycées de l'académie de Versailles, ainsi que Mme Roudneff, inspectrice principale d'académie.

L'entretien s'est déroulé en trois temps :

1. Présentation de Coq et de la bibliothèque Coquelicot.
2. Détail des exercices 2 et 4 des élèves n'ayant pas suivi la spécialité mathématique.
3. Discussion autour des impressions des enseignants et des améliorations à apporter afin d'envisager une séance de travaux pratiques en lycée.

Réaction à l'outil Coq

Malgré un emploi du temps chargé lié au Baccalauréat, l'une des enseignantes avait tenté d'installer et de tester Coq en se servant de tutoriels disponibles sur internet. Elle n'était pas parvenue à installer CoqIde sous Mac et l'usage dans un terminal lui avait donc semblé très austère.

Concernant l'utilisation des tactiques, la signification de "`move => x Hx`" au début de chaque démonstration semble avoir été difficile à comprendre. S'étendre plus longuement sur la présentation de CoqIde, et en particulier sur les notions de "(script de) preuve", "but (à prouver)" et "contexte" pourrait faciliter la compréhension de la tactique `move`.

La différence entre `apply` et `rewrite` n'a pas semblé évidente au premier abord. Les deux servant à modifier le but à l'aide de théorèmes déjà démontrés, il semblait étrange d'utiliser deux commandes différentes. Dans mon exposé préliminaire, préparé avant la découverte du sujet, j'avais présenté les tactiques `apply`, `left`, `right` et `split`. Une présentation de la tactique `rewrite` aurait permis une meilleur compréhension des scripts Coq réalisés.

Avis sur la bibliothèque

La bibliothèque a été dans l'ensemble jugée assez complète. Les solutions choisies à ce moment-là concernant les limites de $\frac{1}{x}$ et $\ln x$ en 0 ont été suffisamment convaincantes pour que cela ne pose pas de problèmes à ces enseignants. Les filtres `at_left` et `at_right` permettent désormais de traiter ces limites sans avoir recours à de tels artifices.

Les principales objections étaient le nommage parfois un peu obscur des théorèmes. Cette remarque portait en particulier sur les théorèmes utilisés pour la résolution des inégalités. L'utilisation de tactiques comme `lra` peut dans certains cas éviter le recours à ces théorèmes. Comme je ne connaissais pas cette tactique à ce moment-là, je leur ai présenté la tactique `admit` qui permet d'admettre des buts. Je leur ai également présenté `Print Assumptions` qui permet d'afficher les axiomes d'un lemme, et en particulier les points de la démonstration ayant été admis. Cette solution leur a semblé parfaitement acceptable pour permettre aux élèves de se concentrer sur les points importants de l'exercice.

Conseils pour une utilisation avec des élèves

Le principe de devoir tout justifier a été assez bien accueilli. La principale question à l'issue de l'après-midi était de savoir quel crédit des élèves pourraient accorder à de telles démonstrations sur ordinateur. La réponse ne pourra être apportée que le jour où des TP de preuve formelle seront testés avec des lycéens.

En raison du très grand nombre de théorèmes proposés, l'idée d'une interface "à la GéoGébra" a été évoquée. Le système de menus de ce logiciel est en effet très bien conçu et facile à utiliser avec des élèves pour des travaux pratiques de géométrie.

Un minimum d'automatisation pour passer les phases d'application de théorèmes sans hypothèses particulières (opposé, addition, multiplication de fonctions) aurait été apprécié pour pouvoir se concentrer sur les théorèmes plus complexes (division par exemple).

5.1.6 Bilan de l'expérience

Cette expérience a été globalement un succès. Elle m'a en effet permis

1. de vérifier que la bibliothèque Coquelicot a atteint une maturité suffisante pour traiter les exercices du Baccalauréat, ce qui n'est pas son objet initial, et
2. de recevoir des a priori positifs d'enseignants en activité sur l'utilisation d'un tel système avec des élèves.

Même si je ne disposais pas de tous les théorèmes nécessaires à la démonstration des résultats des différents exercices, je suis parvenue en 3 heures à obtenir des énoncés et des démonstrations propres et compréhensibles. Les difficultés rencontrées lors de cette épreuve ont permis d'améliorer la bibliothèque Coquelicot en ajoutant des lemmes pour les surpasser.

5.2 Fonctions de Bessel

La seconde application concerne les solutions développables en série entière de l'équation différentielle de Bessel :

$$\forall x \in \mathbb{R},\ x^2 \cdot J_n''(x) + x \cdot J_n'(x) + \left(x^2 - n^2\right) J_n(x) = 0. \tag{5.1}$$

J'ai dans un premier temps démontré que les séries entières

$$J_n(x) = \left(\frac{x}{2}\right)^n \sum_{p=0}^{+\infty} b_{n,p} \left(\left(\frac{x}{2}\right)^2\right)^p \quad \text{avec } b_{n,p} = \frac{(-1)^p}{p!(n+p)!}$$

sont bien définies et sont solutions de l'équation différentielle (5.1) pour tout $n \in \mathbb{N}$. L'unicité de cette solution, aux valeurs initiales près, a été démontrée plus récemment. J'ai également démontré un certain nombre de relations sur les fonctions J_n tirées de la page Wikipédia francophone sur les fonctions de Bessel [2].

5.2.1 Existence et dérivées

L'étude de ces fonctions a été grandement facilitée par l'usage du rayon de convergence dans les théorèmes sur les séries entières. Comme toutes les séries de terme général b_n que nous considérons ici admettent un rayon de convergence infini, les hypothèses de ces théorèmes deviennent triviales.

La démonstration de la valeur du rayon

```
Lemma CV_Bessel1 (n : nat) : CV_radius (Bessel1_seq n) = p_infty.
```

est démontrée à l'aide du théorème de d'Alembert (voir section 3.2.3) en 35 lignes de preuve, dont environ la moitié pour des égalités et inégalités faisant intervenir la fonction `INR : nat -> R` plongeant les entiers dans les réels.

Les preuves des dérivées première et seconde des fonctions J_n ont été faites en grande partie en utilisant la tactique `auto_derive` présentée dans la section 3.6. La série entière de terme général b_n s'écrivant pour tout réel `x`, `PSeries (Bessel1_seq n) x`, la partie "`PSeries (Bessel1_seq n)`" est interprétée par la tactique `auto_derive` comme une fonction arbitraire. Il reste ensuite à justifier que cette fonction est dérivable et à remplacer la dérivée formelle de la série $\sum_{p\in\mathbb{N}} b_{n,p}x^p$ par la série dérivée $\sum_{p\in\mathbb{N}} b'_{n,p}x^p$ avec $b'_{n,p} = (p+1)b_{n,p}$.

En utilisant cette méthode, la démonstration de la dérivée première

$$J_n'(x) = \left(\frac{x}{2}\right)^{n+1} \sum_{p=0}^{+\infty} b'_{n,p} \left(\left(\frac{x}{2}\right)^2\right)^p + \frac{n}{2}\left(\frac{x}{2}\right)^{n-1} \sum_{p=0}^{+\infty} b_{n,p} \left(\left(\frac{x}{2}\right)^2\right)^p$$

nécessite 7 lignes, et la dérivée seconde 15 lignes. Dans ces démonstrations, le fait que la dérivée d'une série entière reste une série entière permet de traiter ces deux démonstrations de la même façon, sans réécriture intermédiaire. De même que pour le rayon de convergence, les principales difficultés de cette seconde démonstration viennent de la manipulation de `INR`.

5.2.2 Différentes égalités

Afin de s'assurer de la facilité d'utilisation des séries entières de la bibliothèque, j'ai démontré l'équation de Bessel (5.1), ainsi que les trois égalités sur les fonctions de Bessel présentes sur la page Wikipédia citée précédemment.

$$\forall n \in \mathbb{N}^*,\ \forall x \in \mathbb{R}^*, \qquad J_{n+1}(x) + J_{n-1}(x) = \frac{2n}{x} J_n(x), \tag{5.2}$$

$$\forall n \in \mathbb{N},\ \forall x \in \mathbb{R}^*, \qquad J_{n+1}(x) = \frac{n}{x} J_n(x) - J_n'(x), \tag{5.3}$$

$$\forall n \in \mathbb{N}^*,\ \forall x \in \mathbb{R}, \qquad J_{n+1}(x) - J_{n-1}(x) = -2J_n'(x). \tag{5.4}$$

2. `http://fr.wikipedia.org/wiki/Fonction_de_Bessel`, consultée en janvier 2015.

Le rayon de convergence des séries de terme général b_n étant infini, la question de savoir si ces égalités sont bien définies ou non ne se pose pas. De plus, l'usage des fonctions totales `PSeries` offre un certain nombre de réécritures sans hypothèses pour les multiplications par des constantes ou des variables.

Pour chacune de ces égalités, la méthode est la suivante :

1. mise sous la forme $\sum a_n x^n = \sum b_n x^n$,
2. application du théorème d'extensionnalité,
3. démonstration de $\forall n \in \mathbb{N},\ a_n = b_n$.

L'égalité (5.2) ne contient que des opérations arithmétiques. Elle a donc pu être traitée rapidement après avoir défini les premières opérations sur les séries entières. La première étape du plan ci-dessus a demandé 18 lignes, dont 3 pour les réécritures sur les séries entières, et la troisième étape 13 lignes.

L'équation différentielle (5.1), ainsi que les deux égalités (5.3) et (5.4), ont dû attendre les théorèmes sur la différentiabilité des séries entières. Comme pour l'égalité (5.2), la première étape s'est faite à chaque fois en une vingtaine de lignes. Pour la troisième étape, les premiers indices ont été traités séparément en raison de leur forme particulière héritée des dérivées de J_n. Pour chacun des indices à traiter, l'ordre de grandeur des démonstrations est, comme pour l'équation (5.2), d'une vingtaine de lignes.

5.2.3 Unicité

Les parties précédentes ont donc permis de démontrer que les séries de terme général b_n sont bien solutions de l'équation différentielle (5.1). La bibliothèque Coquelicot permet d'aller encore plus loin en montrant que ce sont les seules solutions, à une constante multiplicative près, développables en séries entières.

Pour cela, j'ai commencé par démontrer que pour un n fixé, le terme général d'une telle série est forcément de la forme

$$\begin{aligned}
\forall k < n, &\qquad a_k &&= 0,\\
\forall p \in \mathbb{N}, &\quad a_{n+2p+1} &&= 0,\\
\forall p \in \mathbb{N}, &\qquad a_{n+2p} &&= b_{n,p} \cdot \frac{1}{2^{2p}} \cdot n! \cdot a_n.
\end{aligned}$$

avec a_n choisi arbitrairement.

Dans cette démonstration, j'ai commencé par établir les conditions initiales, ainsi que la relation de récurrence vérifiée par les solutions développables en séries entières :

$$\begin{gathered}
(a_0 = 0 \vee n = 0),\ (a_1 = 0 \vee n = 1), \text{ et}\\
\forall k \in \mathbb{N},\ \left((k+2)^2 - n^2\right) a_{k+2} + a_k = 0.
\end{gathered}$$

La clé de cette démonstration est le coefficient $(k+2)^2 - n^2$. En effet, s'il est nul ($k+2 = n$), toute valeur pour a_{k+2} est valide. Les valeurs initiales nous permettent alors d'en déduire la forme explicite ci-dessus.

Comme pour démontrer les égalités de la section précédente, l'étape importante est le passage entre $\forall n,\ a_n = b_n$ et $\sum a_n x^n = \sum b_n x^n$. Le théorème utilisé ici est la réciproque du théorème d'extensionnalité. Au moment où les égalités de la section précédente ont été démontrées, une version de ce théorème existait déjà dans la bibliothèque. Il n'était malheureusement pas utilisable en raison d'hypothèses trop contraignantes. La preuve d'unicité (à une constante multiplicative près) de la fonction de Bessel comme solution de l'équation de Bessel développable en série entière a donc été faite plus tard.

Pour finir cette preuve d'unicité, j'ai démontré que toute solution développable en série entière de l'équation différentielle (5.1) est de la forme αJ_n, avec $\alpha \in \mathbb{R}$. Cette démonstration a nécessité environ 300 lignes de preuve au total.

5.3 L'équation des ondes et la formule de d'Alembert

L'équation des ondes acoustiques dans un milieu homogène est l'une des modélisations les plus simples du phénomène physique de propagation d'onde. Son expression mathématique est un système d'équations aux dérivées partielles, c'est-à-dire des équations portant sur les dérivées par rapport aux coordonnées spatiales et temporelle d'une certaine valeur, ici l'amplitude d'une onde. La résolution de ce système a de nombreuses applications dans le domaine industriel, de l'acoustique d'une salle de concert aux tests non destructifs sur les matériaux.

En dimension 1, le système d'équations s'obtient par l'étude de la position $(x, u(x,t))$ d'un point sur une corde finie ou infinie :

$$\forall (x,t) \in \mathbb{R}^2, \quad \begin{cases} \dfrac{\partial^2 u}{\partial t^2}(x,t) - c^2 \dfrac{\partial^2 u}{\partial x^2}(x,t) = f(x,t) \\ u(x,0) = u_0(x) \\ \dfrac{\partial u}{\partial t}(x,0) = u_1(x) \end{cases} \tag{5.5}$$

avec $c \in \mathbb{R}^*$, $u_0 \in \mathcal{C}^2(\mathbb{R},\mathbb{R})$, $u_1 \in \mathcal{C}^1(\mathbb{R},\mathbb{R})$ et $f \in \mathcal{C}^1(\mathbb{R}\times\mathbb{R}_+,\mathbb{R})$, où $\mathcal{C}^n$ désigne les fonctions n fois différentiable dont la différentielle d'ordre n est continue.

Ce système et sa résolution par un schéma numérique à trois points ont fait l'objet d'une formalisation en Coq [BCF+14]. Cette formalisation a cependant suivi une approche plutôt algébrique et n'a donc mis à contribution qu'une petite partie de la bibliothèque standard de Coq sur l'analyse. En particulier, elle est construite sur l'hypothèse que l'équation des ondes admet une solution suffisamment régulière.

En dimension 1, la formule de d'Alembert

$$u(x,t) = \underbrace{\frac{1}{2}\left(u_0(x+ct) + u_0(x-ct)\right)}_{\alpha(x,t)} + \underbrace{\frac{1}{2c}\int_{x-ct}^{x+ct} u_1(\xi)\,d\xi}_{\beta(x,t)} + \underbrace{\frac{1}{2c}\int_0^t \int_{x-c(t-\tau)}^{x+c(t-\tau)} f(\xi,\tau)\,d\xi\,d\tau}_{\gamma(x,t)} \tag{5.6}$$

est justement solution de l'équation des ondes [lRd47]. Cette fonction, et la démonstration qu'elle est bien solution, a été prise comme cas d'étude pour deux approches différentes [LM12, BLM12] de l'écriture de dérivées et d'intégrales, ainsi que de la démonstration automatique de différentiabilité. Pour faciliter son étude, je l'ai découpée en trois fonctions α, β et γ de difficulté croissante tant au niveau de la définition que de la différentiabilité.

5.3.1 Expression du problème

La première difficulté de cette application est de traduire le problème en Coq. En effet, si la fonction α ne pose pas de problème particulier, la fonction β et plus encore la fonction γ font intervenir des intégrales. Comme cela a été expliqué dans la section 2.1, les intégrales `RiemannInt` de la bibliothèque standard ne sont pas faciles à utiliser.

Les théorèmes `derivable_continuous` et `continuity_implies_RiemannInt` de la bibliothèque standard permettent de passer, comme leur nom l'indique, de la dérivabilité à la continuité et de la continuité à l'intégrabilité. Il sont suffisants pour démontrer que la fonction u_1 est intégrable, et donc que la fonction β est bien définie. Concernant la fonction γ, la bibliothèque standard n'est pas suffisamment riche. En effet, cette bibliothèque ne traitant pas de l'analyse pour les fonctions à deux variables, il n'est pas possible de démontrer directement que la fonction

$$\gamma_0 : \tau \mapsto \int_{x-c(t-\tau)}^{x+c(t-\tau)} f(\xi,\tau)\,d\xi$$

est intégrable.

La formalisation présentée aux JFLA [LM12] utilisant les intégrales de la bibliothèque standard, il a fallu démontrer un certain nombre de résultats concernant les intégrales à paramètre. Parmi ces résultats, on peut remarquer la différentiabilité des intégrales à paramètre qui permet de démontrer l'intégrabilité de la fonction γ_0 :

Théorème. *Soient $f : \mathbb{R}^2 \to \mathbb{R}$ et $a, b \in \mathbb{R}$. Si*
- *$\forall x \in \mathbb{R}$, $t \mapsto f(t,x)$ est intégrable entre a et b,*
- *$\forall t \in \mathbb{R}$, $x \mapsto f(t,x)$ est dérivable, et*
- *$\dfrac{\partial f}{\partial x}$ est continue sur $\mathbb{R}^2$,*

alors la fonction $x \mapsto \int_a^b f(t,x)\,dx$ est dérivable sur $\mathbb{R}$ et

$$\forall x \in \mathbb{R}, \quad \left(y \mapsto \int_a^b f(t,y)\,dt\right)'(x) = \int_a^b \frac{\partial f}{\partial x}(t,x)\,dt.$$

Les intégrales de la bibliothèque standard étant difficiles à utiliser, les résultats démontrés supposent souvent la continuité globale des fonctions considérées.

La bibliothèque Coquelicot dispose de fonctions totales permettant d'écrire les intégrales (voir section 3.5). Les difficultés liées aux intégrales à paramètres n'apparaissent donc plus au moment de l'écriture des lemmes permettant d'établir que la formule de d'Alembert est bien solution de l'équation des ondes. Les fonctions α, β et γ ont donc pu être définies de façon plus naturelle :

```
Definition alpha x t :=
  1 / 2 * (u0 (x + c * t) + u0 (x - c * t)).
Definition beta (x t : R) :=
  1 / (2 * c) * RInt (fun u => u1 u) (x - c * t) (x + c * t).
Definition gamma x t :=
  1 / (2 * c) * RInt (fun tau => RInt (fun xi => f xi tau)
                           (x - c * (t - tau)) (x + c * (t - tau))) 0 t.
```

Comme les fonctions `u0`, `u1` et `f` sont arbitraires, le fait que α, β et γ sont bien définies intervient forcément dans les preuves de différentiabilité permettant de vérifier la formule de d'Alembert, mais pas dans l'écriture du but à prouver.

5.3.2 Vérification de la formule de d'Alembert

La vérification a été faite en démontrant la valeur des différentes dérivées des fonctions α, β et γ dans un premier temps manuellement en Coq pour nous assurer que nous avions tous les lemmes d'analyse nécessaires à la démonstration, puis de façon automatique en utilisant les différentes versions de la tactique développée au cours de ma thèse. Ces résultats sont ensuite utilisés pour démontrer que la formule de D'Alembert vérifie l'équation des ondes. Il est possible de démontrer dans un lemme unique la valeur des dérivées première et seconde de la fonction u. Cependant l'étude séparée des fonctions α, β et γ permet de présenter de façon plus lisible l'efficacité des tactiques pour chacun des type de fonction : simple, avec une intégrale simple et avec une intégrale à paramètre.

Dérivabilité "manuelle"

La fonction α étant formée à partir des opérations arithmétiques de base et de composition de fonctions, la bibliothèque standard est suffisante pour calculer les dérivées premières par rapport à chacune des variables. Pour les dérivées secondes, il a fallu remplacer les dérivées par rapport à chacune des variables de α par leur valeur démontrée précédemment. Pour ces démonstrations, le seul théorème manquant dans la bibliothèque standard était le théorème d'extensionnalité qui a donc été démontré.

Il était a priori possible de démontrer la dérivabilité de la fonction β par rapport à chacune de ses variables uniquement en utilisant les théorèmes de la bibliothèque standard. Pour cela, il suffisait de réécrire cette fonction sous la forme

$$\frac{1}{2c}\left(\int_0^{x+ct} u_1(\xi)\,d\xi - \int_0^{x-ct} u_1(\xi)\,d\xi\right).$$

Malheureusement, les différentes versions du théorème fondamental de l'analyse disponibles dans la bibliothèque standard sont assez difficiles à utiliser pour la première intégrale $\int_0^{x+ct} u_1(\xi)\,d\xi$ (si on suppose $c > 0$), et elles le sont encore plus pour la seconde intégrale. En effet, la version la plus facile à utiliser de la bibliothèque standard n'a été démontrée que pour des fonctions continues sur l'intervalle $[a;b]$, avec $a \leq b$, et pour la fonction $x \mapsto \int_a^x f(t)\,dt$ pour $x \in]a;b[$. Les autres versions imposaient le passage par le type `C1_fun` des fonctions dérivables et de dérivée continue, et donc l'application d'un théorème d'extensionnalité. J'ai donc choisi de démontrer une version plus générale et surtout plus facile à utiliser de ce théorème : la fonction à deux variables $(x, y) \mapsto \int_x^y f(t)\,dt$ est différentiable, et admet pour différentielle $(-f(x); f(y))$. La dérivation de la fonction β par rapport à chacune des variables utilise un corollaire permettant d'éviter le passage par la différentiabilité :

$$(x \mapsto \int_{b(x)}^{a(x)} f(t)\,dt)' = a'(x) \cdot f(a(x)) - b'(x) \cdot f(b(x)).$$

La dérivée première par rapport à chacune des variables étant, comme la fonction α, formée à partir d'opérations de base et de composition, la dérivée seconde n'a pas posé de problème particulier.

Pour finir, les dérivées de la fonction γ par rapport à x se calculent par le théorème de dérivation d'une intégrale à paramètre. Le traitement des dérivées par rapport à t est un peu plus compliqué puisque

t apparaît à la fois comme borne de l'intégrale extérieure et à l'intérieur de celle-ci. Il faut donc passer par la fonction suivante et ses dérivées partielles.

$$(t_1, t_2) \mapsto \frac{1}{2c} \int_0^{t_1} \int_{x-c(t_2-\tau)}^{x+c(t_2-\tau)} f(\xi, \tau)\, d\xi\, d\tau.$$

La bibliothèque standard de Coq n'est d'aucune utilité pour traiter la fonction γ. Elle ne traite ni des intégrales à paramètre ni des fonctions à plusieurs variables en général et de leur différentiabilité. J'ai donc démontré en Coq les théorèmes concernant les intégrales à paramètres, ainsi que ceux concernant la continuité et la différentiabilité des fonctions à deux variables qui ont été présentés dans la section 3.4.

Dérivabilité automatique

Comme je l'ai expliqué dans la section 3.6, deux versions d'une tactique permettant de démontrer automatiquement la dérivabilité d'une fonction ont été développées au cours de ma thèse. Elles ont toutes les deux été testées sur la formule de d'Alembert.

La première version a permis de démontrer automatiquement les dérivées partielles des fonctions α et β. Les dérivées partielles ainsi vérifiées ont ensuite été utilisées pour démontrer, également automatiquement, les dérivées partielles secondes utilisées pour vérifier la formule de D'Alembert. Pour ces deux fonctions, le nombre de lignes de Coq pour calculer les différentes dérivées a été divisé par 4. Cette première tactique ne permet cependant pas de dériver la fonction γ car elle ne prend pas en compte les intégrales à paramètre.

La seconde version couvre les intégrales à paramètre en plus des fonctions déjà traitées par la première tactique. Les valeurs des dérivées partielles des trois fonctions ont pu être démontrées automatiquement. Pour cette application, une tactique ad-hoc `auto_derive_2` a été définie afin de traiter en une seule fois les dérivées secondes. Cette tactique commence par calculer une dérivée première `f'`, ainsi que ses conditions de dérivabilité. Elle vérifie ensuite que cette nouvelle fonction est bien dérivable et admet pour dérivée la valeur fournie par l'utilisateur.

Chapitre 6

Conclusion

La bibliothèque Coquelicot [BLM14a] est une extension de la bibliothèque standard d'analyse réelle de Coq. Elle contient environ 430 définitions, 1 675 théorèmes et 30 000 lignes de preuve Coq. Elle dispose de définitions homogènes et des lemmes concernant les limites, différentielles, intégrales, séries et séries entières. Chaque fois que cela est possible, les définitions de la bibliothèque Coquelicot ont été démontrées équivalentes à celles de la bibliothèque standard afin que l'utilisateur puisse aisément combiner les deux bibliothèques.

Pour toutes ces notions d'analyse réelle, des fonctions totales ont été formalisées. Ces fonctions renvoient la valeur attendue en cas de convergence et une valeur arbitraire dans le cas contraire. L'absence des types dépendants dans la bibliothèque Coquelicot rend les énoncés des théorèmes plus faciles à lire et à écrire qu'en utilisant les outils de la bibliothèque standard. En effet, les hypothèses de convergence, différentiabilité et autres peuvent maintenant être traitées indépendemment de l'écriture des limites, dérivées, etc. Sans une telle fonctionnalité, il serait presque impossible d'énoncer des théorèmes comme celui de Taylor-Lagrange. Cela simplifie également le processus de preuve car les règles de réécriture sont maintenant utilisables en pratique, et même sans hypothèses de convergence pour certaines. Cette approche basée sur les fonctions totales est similaire à ce que d'autres assistants de preuve peuvent faire avec l'opérateur ε de Hilbert, mais sans avoir à supposer l'existence de cet opérateur en Coq. Pour les propriétés topologiques, la bibliothèque repose sur l'utilisation généralisée de filtres. L'arithmétique et l'ordre sur les nombrées réels étendus permettent d'exprimer facilement les théorèmes sur les limites et les séries entières sans avoir à distinguer les cas finis et infinis.

Afin de permettre une utilisation plus générale de cette bibliothèque, une hiérarchie algébrique a été formalisée. Bien que partielle, elle permet d'exprimer limites, dérivées et intégrales pour des espaces aussi variés que les nombres réels et complexes, les matrices, ou tout espace disposant d'une structure de module muni d'une valeur absolue. Cette approche permet de disposer d'une base commune de définitions et de théorèmes usuels pour tous les espaces considérés. Il est alors possible de se concentrer sur des notions plus spécifiques à chaque espace comme les limites généralisées pour les nombres réels ou les intégrales de chemin pour les nombres complexes. En plus de ces notions de base, plusieurs extensions ont également été développées : la différentiabilité pour les fonctions à deux variables réelles, les intégrales à paramètres, les comportements asymptotiques et une tactique pour l'automatisation des preuves de différentiabilité. Enfin, un grand soin a été pris pour nommer et énoncer les théorèmes, de manière à éviter certains des problèmes les plus déconcertants de la bibliothèque standard.

Dans ce document, j'ai aussi présenté un aperçu des différentes bibliothèque d'analyse réelle qui ont servi d'inspiration à la bibliothèque Coquelicot [BLM14b]. J'ai également présenté les applications qui ont motivé cette bibliothèque, comme la formule de d'Alembert, ou qui ont servi à la tester, comme le Baccalauréat de Mathématiques. Les notions couvertes par ces applications sont résumées dans le tableau 6.1. En effet, je considère qu'il est important, lors de la conception d'une bibliothèque pour les mathématiques, de garder à l'esprit les applications et de s'assurer qu'elle sera utilisable en pratique. Ces applications sont intégrées aux exemples de la bibliothèque Coquelicot afin de s'assurer que les différentes améliorations apportées à celle-ci permettent toujours de traiter les problèmes étudiés précédemment.

La bibliothèque Coquelicot est une amélioration notable par rapport à la bibliothèque standard de Coq et elle est adaptée pour nos applications actuelles. La version 2.1, ainsi que les versions précédentes, sont disponibles depuis la page du projet `http://coquelicot.saclay.inria.fr/`.

La bibliothèque Coquelicot est actuellement utilisée par des personnes extérieures au projet. Ainsi, elle a été utilisée par G.C. Hulette et son équipe pour formaliser un phénomène en thermodynamique [HAMR14].

Notions	Applications
Normalisation du nommage	Bac
Réels étendus	Bac, Bessel
Séries et séries entières	Bessel
Intégrale de Riemann	D'Alembert
$\mathbb{R}^2$	D'Alembert
Fonctions totales	Bac, Bessel, D'Alembert
Automatisation de la dérivée	D'Alembert, Bessel
Généralisation de l'analyse	Analyse complexe
Généralisation des limites	Intégrale de Riemann

TABLEAU 6.1 – Les notions de Coquelicot et les applications qui les utilisent

Elle a aussi été utilisée par Y. Bertot dans la démonstration d'un algorithme permettant de calculer les décimales du nombre π [Ber14]. La preuve utilisée s'appuie sur quelques notions non élémentaires, comme le raisonnement sur les intégrales impropres ou la dérivation sous le signe d'intégration. Les intégrales généralisées étaient absentes de la bibliothèque Coquelicot à ce moment-là. Les fonctions totales ont cependant permis de simplifier les démonstrations. Ce travail a été une motivation pour formaliser les intégrales de Riemann impropres actuellement présentes dans la bibliothèque Coquelicot.

La formalisation actuelle des nombres complexes permet d'utiliser les limites et les dérivées de fonctions complexes, ainsi que les intégrales de fonctions à variable réelle. De nombreuses notions spécifiques aux fonctions complexes, comme les fonctions analytiques et les intégrales de chemin, sont encore à formaliser. L'objectif serait de prouver qu'une équation différentielle linéaire d'ordre 1 à coefficients matriciels admet une unique solution holomorphe sur un ouvert étoilé.

Il serait également intéressant d'augmenter le nombre d'intégrales disponibles dans la bibliothèque Coquelicot. En effet, même si l'intégrale de Riemann est assez couramment utilisée, ce n'est pas la seule. L'intégrale de Kurzweil-Henstock (ou intégrale KH, ou intégrale de jauge) a été formalisée pendant l'été 2014 par X. Onfroy. Cette intégrale est une généralisation de l'intégrale de Riemann. Une autre intégrale intéressante à formaliser pourrait être l'intégrale de Lebesgue pour aborder la théorie de la mesure et la formalisation de la méthode des éléments finis.

Un autre point à étudier serait de rattacher des nombres autres que les nombres réels ou complexes à la bibliothèque Coquelicot comme les nombres p-adiques. Ces nombres ont déjà été implémentés en Coq par Á. Pelayo, V. Voevodsky et M.A. Warren [PVW15] ; il serait intéressant de voir dans quelle mesure il est possible de faire de l'analyse avec ces nombres. Comme il s'agit d'un anneau commutatif valué, il est a priori possible d'étendre la bibliothèque Coquelicot afin d'obtenir gratuitement des limites et des dérivées.

Pour finir, la bibliothèque Coquelicot est encore très dépendante des nombres réels de la bibliothèque standard. En particulier, les espaces uniformes en dépendent par le prédicat `ball : U -> R -> U` et les intégrales de Riemann sont définies pour les fonctions à variable réelle et à valeur dans un $\mathbb{R}$-module. Il serait donc intéressant de remplacer ces dernières traces de la bibliothèque standard afin de pouvoir également utiliser cette bibliothèque avec les réels constructifs de C-CoRN.

Bibliographie

[AD04] Jeremy Avigad and Kevin Donnelly. Formalizing O notation in Isabelle/HOL. In David A. Basin and Michaël Rusinowitch, editors, *Proceedings of Automated Reasoning - Second International Joint Conference (IJCAR 2004)*, volume 3097 of *Lecture Notes in Computer Science*, pages 357–371, Cork, Ireland, July 2004.

[AK96] Rob D. Arthan and Dave King. Development of practical verification tools. *ICL Systems Journal*, 11(1):106–122, 1996.

[Art01] Rob D. Arthan. An irrational construction of $\mathbb{R}$ from $\mathbb{Z}$. In Richard J. Boulton and Paul B. Jackson, editors, *Proceedings of the 14th International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2001)*, volume 2152 of *Lecture Notes in Computer Science*, pages 43–58, Edinburgh, UK, 2001.

[Art12a] Rob D. Arthan. *Mathematical Case Studies: Basic Analysis*. Lemma 1 Ltd., 2012.

[Art12b] Rob D. Arthan. Mathematical case studies: the complex numbers. Technical report, Lemma 1 Ltd., August 2012.

[Bac13] Baccalauréat général, Série S, Mathématiques, Session 2013, June 2013. `http://eduscol.education.fr/prep-exam/sujets/13MASCOMLR1.pdf`.

[BC04] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer, 2004.

[BCF+13] Sylvie Boldo, François Clément, Jean-Christophe Filliâtre, Micaela Mayero, Guillaume Melquiond, and Pierre Weis. Wave equation numerical resolution: a comprehensive mechanized proof of a C program. *Journal of Automated Reasoning*, 50(4):423–456, 2013.

[BCF+14] Sylvie Boldo, François Clément, Jean-Christophe Filliâtre, Micaela Mayero, Guillaume Melquiond, and Pierre Weis. Trusting computations: A mechanized proof from partial differential equations to actual program. *Computers & Mathematics with Applications*, 68(3):325 – 352, 2014.

[Ber14] Yves Bertot. PI, Arithmetic geometric means, and formal verification in Coq using Coquelicot. In *Mathematical Structures of Computation*, February 2014.

[BGOBP08] Yves Bertot, Georges Gonthier, Sidi Ould Biha, and Ioana Pasca. Canonical Big Operators. In *Theorem Proving in Higher Order Logics*, volume 5170/2008 of *LNCS*, Montreal, Canada, August 2008.

[Bis67] Errett Bishop. *Foundations of constructive analysis*. University of California, San Diego, 1967.

[BLM12] Sylvie Boldo, Catherine Lelay, and Guillaume Melquiond. Improving real analysis in Coq: a user-friendly approach to integrals and derivatives. In Chris Hawblitzel and Dale Miller, editors, *Proceedings of the 2nd International Conference on Certified Programs and Proofs (CPP)*, volume 7679 of *Lecture Notes in Computer Science*, pages 289–304, Kyoto, Japan, January 2012.

[BLM14a] Sylvie Boldo, Catherine Lelay, and Guillaume Melquiond. Coquelicot: A user-friendly library of real analysis for coq. *Mathematics in Computer Science*, pages 1–22, 2014.

[BLM14b] Sylvie Boldo, Catherine Lelay, and Guillaume Melquiond. Formalization of real analysis: A survey of proof assistants and libraries. *Mathematical Structures in Computer Science*, 2014. Accepted for publication.

[BM11] Sylvie Boldo and Guillaume Melquiond. Flocq: A unified library for proving floating-point algorithms in Coq. In Elisardo Antelo, David Hough, and Paolo Ienne, editors, *Proceedings of the 20th IEEE Symposium on Computer Arithmetic (ARITH 20)*, pages 243–252, Tübingen, Germany, July 2011.

[Bou71] N. Bourbaki. *Topologie Générale: Chapitres 1 à 4.* Éléments de mathématiques. Springer-Verlag Berlin and Heidelberg GmbH & Co. K, 1971.

[Bou74] N. Bourbaki. *Topologie Générale: Chapitres 5 à 10.* Éléments de mathématiques. Springer-Verlag Berlin and Heidelberg GmbH & Co. K, 1974.

[But09] Ricky W. Butler. Formalization of the integral calculus in the PVS theorem prover. *Journal of Formalized Reasoning*, 2(1):1–26, 2009.

[Byl90a] Czesław Byliński. The complex numbers. *Journal of Formalized Mathematics*, 1(3):507–513, 1990.

[Byl90b] Czesław Byliński. Functions and their basic properties. *Journal of Formalized Mathematics*, 1(1):55–65, 1990.

[CAB+85] Robert L. Constable, Stuart F. Allen, Mark Bromley, Walter R. Cleaveland, James F. Cremer, Robert W. Harper, Douglas J. Howe, Todd B. Knoblock, Nax P. Mendler, Prakash Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing Mathematics with The Nuprl Proof Development System.* CreateSpace Independent Publishing Platform, Cornell University, Ithaca, NY, 1985.

[CDG06] Alberto Ciaffaglione and Pietro Di-Gianantonio. A certified, corecursive implementation of exact real numbers. *Theoretical Computer Science*, 351(1):39–51, 2006.

[CF03] Luís Cruz-Filipe. A constructive formalization of the fundamental theorem of calculus. In *Proceedings of the International Conference on Types for Proofs and Programs (TYPES'02)*, volume 2646 of *Lecture Notes in Computer Science*, pages 108–126, Berg en Dal, The Netherlands, 2003.

[CFGW04] Luís Cruz-Filipe, Herman Geuvers, and Freek Wiedijk. C-CoRN: the constructive Coq repository at Nijmegen. In Andrea Asperti, Grzegorz Bancerek, and Andrzej Trybulec, editors, *Proceedings of the 3rd International Conference of Mathematical Knowledge Management (MKM)*, volume 3119 of *Lecture Notes in Computer Science*, pages 88–103, Białowieża, Poland, September 2004.

[CG10] John R. Cowles and Ruben Gamboa. Using a first order logic to verify that some set of reals has no Lesbegue measure. In Matt Kaufmann and Lawrence C. Paulson, editors, *Proceeding of the 1st International Conference of Interactive Theorem Proving (ITP 2010)*, volume 6172 of *Lecture Notes in Computer Science*, pages 25–34, Edinburgh, UK, July 2010.

[Coh12] Cyril Cohen. Reasoning about big enough numbers in Coq. In *Proceedings of the 4th Coq Workshop*, Princeton, NJ, USA, 2012.

[CS12] Pierre Castéran and Matthieu Sozeau. A gentle introduction to type classes and relations in Coq. Technical Report hal-00702455, version 1, 2012.

[Del00] David Delahaye. A tactic language for the system Coq. In Michel Parigot and Andrei Voronkov, editors, *Proceedings of the 7th International Conference of Logic for Programming and Automated Reasoning*, volume 1955 of *Lecture Notes in Artificial Intelligence*, pages 85–95, Reunion Island, France, November 2000.

[Des02] Olivier Desmettre, 2002. Coq standard library – RiemannInt.

[Dut96] Bruno Dutertre. Elements of mathematical analysis in PVS. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Joakim von Wright, Jim Grundy, and John Harrison, editors, *Proceedings of the 9th International Conference Theorem Proving in Higher Order Logics (TPHOLs)*, volume 1125 of *Lecture Notes in Computer Science*, pages 141–156, August 1996.

[EK99] Noboru Endou and Artur Korniłowicz. The definition of the Riemann definite integral and some related lemmas. *Journal of Formalized Mathematics*, 8(1):3–102, 1999.

[EWS01] Noboru Endou, Katsumi Wasaki, and Yasunari Shidama. Definition of integrability for partial functions from $\mathbb{R}$ to $\mathbb{R}$ and integrability for continuous functions. *Journal of Formalized Mathematics*, 9(2):281–284, 2001.

[Fle00] Jacques D. Fleuriot. On the mechanization of real analysis in Isabelle/HOL. In Mark Aagaard and John Harrison, editors, *Proceeding of the 13th International Conference of Theorem Proving in Higher Order Logics (TPHOLs)*, volume 1869 of *Lecture Notes in Computer Science*, pages 145–161, Portland, OR, USA, August 2000.

[Gam00] Ruben Gamboa. *Computer-Aided Reasoning: ACL2 Case Studies*, volume 4 of *Advances in Formal Methods*, chapter Continuity and differentiability, pages 145–161. Springer, 2000.

[GC09] Ruben Gamboa and John R. Cowles. Inverse functions in ACL2(r). In Sandip Ray and David Russinoff, editors, *Proceedings of the 8th International Workshop on the ACL2 Theorem Prover and its Applications*, pages 57–61, Boston, MA, USA, May 2009.

[GCK04] Ruben Gamboa, John R. Cowles, and Nadya Kuzmina. Axiomatic events in ACL2(r): A story of defun, defun-std, and encapsulate. In *Proceedings of the 5th International Workshop on the ACL2 Theorem Prover and its Applications*, Austin, TX, USA, 2004.

[GK01] Ruben Gamboa and Matt Kaufmann. Nonstandard analysis in ACL2. *Journal of Automated Reasoning*, 27(4):323–351, November 2001.

[GM93] Michael J. C. Gordon and Thomas F. Melham. *Introduction to HOL: a theorem proving environment for higher order logic*. Cambridge University Press, 1993.

[GN02] Herman Geuvers and Milad Niqui. Constructive reals in Coq: Axioms and categoricity. In Paul Callaghan, Zhaohui Luo, James McKinna, and Robert Pollack, editors, *Proceedings of the International Workshop on Types for Proofs and Programs (TYPES'00)*, volume 2277 of *Lecture Notes in Computer Science*, pages 79–95, Durham, UK, December 2002.

[GOBS69] James R. Guard, Francis C. Oglesby, James H. Bennett, and Larry G. Settle. Semi-automated mathematics. *Journal of the ACM*, 16:49–62, 1969.

[Got00] Hanne Gottliebsen. Transcendental functions and continuity checking in PVS. In Mark Aagaard and John Harrison, editors, *Proceedings of the 13th International Conference Theorem Proving in Higher Order Logics (TPHOLs 2000)*, volume 1869 of *Lecture Notes in Computer Science*, pages 197–214, Portland, OR, USA, 2000.

[GPWZ02] Herman Geuvers, Randy Pollack, Freek Wiedijk, and Jan Zwanenburg. A constructive algebraic hierarchy in Coq. *Journal of Symbolic Computation*, 34(4):271–286, 2002. Special Issue on the Integration of Automated Reasoning and Computer Algebra Systems.

[HAMR14] Geoffrey C. Hulette, Robert C. Armstrong, Jackson R. Mayo, and Joseph R. Ruthruff. Theorem-proving analysis of digital control logic interacting with continuous dynamics. In *NSV 2014, 7th International Workshop on Numerical Software Verification*, Vienne, Autriche, 2014.

[Har94] John Harrison. Constructing the real numbers in HOL. *Formal Methods in System Design*, 5(1–2):35–59, July 1994.

[Har96] John Harrison. Proof style. In Eduardo Giménez and Christine Paulin-Mohring, editors, *Proceedings of the International Workshop on Types for Proofs and Programs (TYPES'96)*, volume 1512 of *Lecture Notes in Computer Science*, pages 154–172, Aussois, France, December 1996.

[Har97] John Harrison. Floating point verification in HOL Light: the exponential function. Technical Report 428, University of Cambridge Computer Laboratory, 1997.

[Har98] John Harrison. *Theorem Proving with the Real Numbers*. Springer-Verlag, 1998.

[Har06] John Harrison. Towards self-verification of HOL Light. In Ulrich Furbach and Natarajan Shankar, editors, *Proceedings of the 3rd International Joint Conference (IJCAR 2006)*, volume 4130 of *Lecture Notes in Computer Science*, pages 177–191, Seattle, WA, USA, 2006.

[Har07] John Harrison. Formalizing basic complex analysis. *From Insight to Proof: Festschrift in Honour of Andrzej Trybulec. Studies in Logic, Grammar and Rhetoric*, 10(23):151–165, 2007.

[Har09] John Harrison. HOL Light: An overview. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, volume 5674 of *Lecture Notes in Computer Science*, pages 60–66, Munich, Germany, 2009.

[Har13] John Harrison. The HOL Light theory of Euclidean space. *Journal of Automated Reasoning*, 50:173–190, 2013.

[HH11] Johannes Hölzl and Armin Heller. Three chapters of measure theory in Isabelle/HOL. In Marko van Eekelen, Herman Geuvers, Julien Schmaltz, and Freek Wiedijk, editors, *Proceedings of the 2nd International Conference of Interactive Theorem Proving*, volume 6898 of *Lecture Notes in Computer Science*, pages 135–151, Berg en Dal, The Netherlands, August 2011.

[HIH13] Johannes Hölzl, Fabian Immler, and Brian Huffman. Type classes and filters for mathematical analysis in Isabelle/HOL. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Proceedings of the 4th International Conference on Interactive Theorem Proving (ITP)*, volume 7998 of *Lecture Notes in Computer Science*, pages 279–294, Rennes, France, 2013.

[HOL12] HOL4 development team. The HOL System LOGIC. Technical report, University of Cambridge, NICTA and University of Utah, 2012.

[Hur11] Joe Hurd. The OpenTheory standard theory library. In Mihaela Bobaru, Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi, editors, *Proceedings of the 3rd International Symposium on NASA Formal Methods (NFM 2011)*, volume 6617 of *Lecture Notes in Computer Science*, pages 177–191, Pasadena, CA, USA, April 2011.

[IH12] Fabian Immler and Johannes Hölzl. Numerical analysis of ordinary differential equations in Isabelle/HOL. In Lennart Beringer and Amy Felty, editors, *Proceedings of the 3rd International Conference on Interactive Theorem Proving (ITP 2012)*, volume 7406 of *Lecture Notes in Computer Science*, pages 377–392, Princeton, NJ, USA, 2012.

[Jon93] Claire Jones. Completing the rationals and metric spaces in LEGO. In Gérard Huet and Gordon Plotkin, editors, *Papers presented at the second annual Workshop on Logical environments*, pages 297–316, Edinburgh, UK, 1993. Cambridge University Press.

[JP09] Nicolas Julien and Ioana Paşca. Formal verification of exact computations using Newton's method. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2009)*, volume 5674 of *Lecture Notes in Computer Science*, pages 408–423, München, Germany, 2009.

[Jut77] L. S. van Bentham Jutting. *Checking Landau's "Grundlagen" in the AUTOMATH System.* PhD thesis, Eindhoven University of Technology, 1977.

[KMM00] Matt Kaufmann, J. Strother Moore, and Panagiotis Manolios. *Computer-Aided Reasoning: An Approach.* Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[KO09] Cezary Kaliszyk and Russell O'Connor. Computing with classical real numbers. *Journal of Automated Reasoning*, 2(1):27–29, 2009.

[Kot90a] Jarosław Kotowicz. Convergent sequences and the limit of sequences. *Journal of Formalized Mathematics*, 1(2):273–275, 1990.

[Kot90b] Jarosław Kotowicz. Real sequences and basic operations on them. *Journal of Formalized Mathematics*, 1(2):269–272, 1990.

[Kot91a] Jarosław Kotowicz. The limit of a real function at a point. *Journal of Formalized Mathematics*, 2(1):71–80, 1991.

[Kot91b] Jarosław Kotowicz. The limit of a real function at infinity. *Journal of Formalized Mathematics*, 2:17–28, 1991.

[Kot91c] Jarosław Kotowicz. One-side limits of a real function at a point. *Journal of Formalized Mathematics*, 2(1):29–40, 1991.

[KS13] Robbert Krebbers and Bas Spitters. Type classes for efficient exact real arithmetic in Coq. *Logical Methods in Computer Science*, 9(1:1):1–27, 2013.

[Lec35] Maurice Lecat. *Erreurs de mathématiciens des origines à nos jours.* Castaigne, 1935.

[Les07] David R. Lester. Topology in PVS: continuous mathematics with applications. In John Rushby and Natarajan Shankar, editors, *Proceedings of the 2nd workshop on Automated Formal Methods (AFM '07)*, pages 11–20, Atlanta, GA, USA, November 2007.

[Les08] David R. Lester. Real number calculations and theorem proving. In Otmane Ait Mohamed, César A. Muñoz, and Sofiène Tahar, editors, *Proceedings of the 21st International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2008)*, volume 5170 of *Lecture Notes in Computer Science*, pages 215–229, August 2008.

[Let04] Pierre Letouzey. *Programmation fonctionnelle certifiée – L'extraction de programmes dans l'assistant Coq*. PhD thesis, Université Paris-Sud, July 2004.

[LM12] Catherine Lelay and Guillaume Melquiond. Différentiabilité et intégrabilité en Coq. Application à la formule de d'Alembert. In *23èmes Journées Francophones des Langages Applicatifs*, pages 119–133, Carnac, France, 2012.

[lRd47] Jean le Rond d'Alembert. *Histoire de l'académie royale des sciences et belles lettres de Berlin*, volume 3, chapter Recherches sur la courbe que forme une corde tenduë mise en vibration, pages 214–219. 1747.

[May01] Micaela Mayero. *Formalisation et automatisation de preuves en analyses réelle et numérique*. PhD thesis, Université Paris VI, décembre 2001.

[May12] Micaela Mayero. *Problèmes critiques et preuves formelles*. Habilitation à diriger des recherches, Université Paris 13, novembre 2012.

[MHT10] Tarek Mhamdi, Osman Hasan, and Sofiène Tahar. On the formalization of the Lebesgue integration theory in HOL. In Matt Kaufmann and Lawrence C. Paulson, editors, *Proceedings of the 1st International Conference of Interactive Theorem Proving (ITP 2010)*, volume 6172 of *Lecture Notes in Computer Science*, pages 387–402, Edinburgh, UK, July 2010.

[MLK98] J. Strother Moore, Tom Lynch, and Matt Kaufmann. A mechanically checked proof of the correctness of the kernel of the AMD5K86 floating point division algorithm. *IEEE Transactions on Computers*, 47(9):913–926, September 1998.

[MM03] Panagiotis Manolios and J. Strother Moore. Partial functions in ACL2. *Journal of Automated Reasoning*, 31(2):107–127, December 2003.

[MS13] Evgeny Makarov and Bas Spitters. The Picard algorithm for ordinary differential equations in Coq. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Proceeding of the 4th International Conference of Interactive Theorem Proving (ITP 2013)*, volume 7998 of *Lecture Notes in Computer Science*, pages 463–468, Rennes, France, July 2013.

[MWS01] Takashi Mitsuishi, Katsumi Wasaki, and Yasunari Shidama. Property of complex sequence and continuity of complex function. *Journal of Formalized Mathematics*, 9(1):185–190, 2001.

[NES08] Keiko Narita, Noboru Endou, and Yasunari Shidama. Integral of complex-valued measurable function. *Journal of Formalized Mathematics*, 16(4):319–324, 2008.

[NK09] Adam Naumowicz and Artur Korniłowicz. A brief overview of Mizar. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Proceedings of the 22th International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2009)*, volume 5674 of *Lecture Notes in Computer Science*, pages 67–72, Munich, Germany, August 2009.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer-Verlag Berlin, 2002.

[O'C07] Russell O'Connor. A monadic, functional implementation of real numbers. *Mathematical Structures in Computer Science*, 17:129–159, 1 2007.

[O'C08] Russell O'Connor. Certified exact transcendental real number computation in Coq. In Otmane Ait Mohamed, César Muñoz, and Sofiène Tahar, editors, *Proceedings of the 21th International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2008)*, volume 5170 of *Lecture Notes in Computer Science*, pages 246–261, Montreal, Canada, August 2008.

[ORS92] Sam Owre, John Rushby, and Natarajan Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, jun 1992.

[OS03] Sam Owre and Nataraja Shankar. The PVS Prelude Library. Technical Report SRI-CSL-03-01, SRI International, 2003.

[OS10] Russell O'Connor and Bas Spitters. A computer-verified monadic functional implementation of the integral. *Theoretical Computer Science*, 411(37):3386–3402, 2010.

[Pau87] Lawrence C. Paulson. *Logic and Computation: Interactive Proof with Cambridge LCF*. Cambridge University Press, New York, NY, USA, 1987.

[Per92] Beata Perkowska. Functional sequence from a domain to a domain. *Journal of Formalized Mathematics*, 3(1):17–21, 1992.

[Pro06] ProofPower development team. Proofpower - HOL reference manual. Technical report, Lemma 1 Ltd., 2006.

[PVW15] Álvaro Pelayo, Vladimir Voevodsky, and Michael A. Warren. A univalent formalization of the p-adic numbers. *Mathematical Structures in Computer Science*, FirstView:1–25, March 2015.

[PYSN09] Chanapat Pacharapokin, Hiroshi Yamazaki, Yasunari Shidama, and Yatsuka Nakamura. Complex function differentiability. *Journal of Formalized Mathematics*, 17(1-4):67–72, 2009.

[Rac91] Konrad Raczkowski. Integer and rational exponents. *Journal of Formalized Mathematics*, 2(1):125–130, 1991.

[RG11] Peter Reid and Ruben Gamboa. Automatic differentiation in ACL2. In Marko Eekelen, Herman Geuvers, Julien Schmaltz, and Freek Wiedijk, editors, *Proceeding of the 2nd International Conference of Interactive Theorem Proving (ITP 2011)*, volume 6898 of *Lecture Notes in Computer Science*, pages 312–324, Berg en Dal, The Netherlands, August 2011.

[Ric04] Stefan Richter. Formalizing integration theory with an application to probabilistic algorithms. In Konrad Slind, Annette Bunker, and Ganesh Gopalakrishnan, editors, *Proceedings of the 17th International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2004)*, volume 3223 of *Lecture Notes in Computer Science*, pages 271–286, Park City, UT, USA, September 2004.

[RN91a] Konrad Raczkowski and Andrzej Nedzusiak. Real exponents and logarithms. *Journal of Formalized Mathematics*, 2(2):213–216, 1991.

[RN91b] Konrad Raczkowski and Andrzej Nedzusiak. Series. *Journal of Formalized Mathematics*, 2(4):449–452, 1991.

[ROS98] John Rushby, Sam Owre, and Natarajan Shankar. Subtypes for specifications: Predicate subtyping in PVS. *IEEE Transactions on Software Engineering*, 24(9):709–720, sep 1998.

[RS90a] Konrad Raczkowski and Paweł Sadowski. Real function continuity. *Journal of Formalized Mathematics*, 2, 1990.

[RS90b] Konrad Raczkowski and Paweł Sadowski. Real function differentiability. *Journal of Formalized Mathematics*, 1(4):797–801, 1990.

[RZ04] Krzysztof Retel and Anna Zalewska. Mizar as a tool for teaching mathematics. In *Proceedings of Mizar 30 workshop*, 2004.

[Sai99] Amokrane Saibi. *Formalization of Mathematics in Type Theory. Generic tools of Modelisation and Demonstration. Application to Category Theory.* PhD thesis, Université Pierre et Marie Curie - Paris VI, March 1999.

[SGL+13] Zhiping Shi, Weiqing Gu, Xiaojuan Li, Yong Guan, Shiwei Ye, Jie Zhang, and Hongxing Wei. The gauge integral theory in HOL4. *Journal of Applied Mathematics*, 2013, Special Issue:1–7, 2013.

[SK97] Yasunari Shidama and Artur Korniłowicz. Convergence and the limit of complex sequences. series. *Journal of Formalized Mathematics*, 6(3):403–410, 1997.

[SLG+13] Zhiping Shi, Liming Li, Yong Guan, Xiaoyu Song, Minhua Wu, and Jie Zhang. Formalization of the complex number theory in HOL4. *Applied Mathematics & Information*, 7(1):279–286, January 2013.

[SN08] Konrad Slind and Michael Norrish. A brief overview of HOL4. In *Theorem Proving in Higher Order Logics*, pages 28–32. Springer, 2008.

[SvdW11] Bas Spitters and Eelis van der Weegen. Type classes for mathematics in type theory. *Mathematical Structures of Computer Science*, 21:1–31, 2011.

[Try93] Andrzej Trybulec. Some features of the Mizar language. In *Proceedings of the ESPRIT Workshop*, Torino, Italy, 1993.

[Wen02] Markus Wenzel. *Isabelle/Isar – a versatile environment for human-readable formal proof documents.* PhD thesis, Institut für Informatik, Technische Universität München, 2002.

[WW02] Markus Wenzel and Freek Wiedijk. A comparison of Mizar and Isar. *Journal of Automated Reasoning*, 29(3-4):389–411, 2002.

Repenser la bibliothèque réelle de Coq : vers une formalisation de l'analyse classique mieux adaptée

L'analyse réelle a de nombreuses applications car c'est un outil approprié pour modéliser de nombreux phénomènes physiques et socio-économiques. En tant que tel, sa formalisation dans des systèmes de preuve formelle est justifié pour permettre aux utilisateurs de vérifier formellement des théorèmes mathématiques et l'exactitude de systèmes critiques. La bibliothèque standard de Coq dispose d'une axiomatisation des nombres réels et d'une bibliothèque de théorèmes d'analyse réelle. Malheureusement, cette bibliothèque souffre de nombreuses lacunes. Par exemple, les définitions des intégrales et des dérivées sont basées sur les types dépendants, ce qui les rend difficiles à utiliser dans la pratique. Cette thèse décrit d'abord l'état de l'art des différentes bibliothèques d'analyse réelle disponibles dans les assistants de preuve. Pour pallier les insuffisances de la bibliothèque standard de Coq, nous avons conçu une bibliothèque facile à utiliser : Coquelicot. Une façon plus facile d'écrire les formules et les théorèmes a été mise en place en utilisant des fonctions totales à la place des types dépendants pour écrire les limites, dérivées, intégrales et séries entières. Pour faciliter l'utilisation, la bibliothèque dispose d'un ensemble complet de théorèmes couvrant ces notions, mais aussi quelques extensions comme les intégrales à paramètres et les comportements asymptotiques. En plus, une hiérarchie algébrique permet d'appliquer certains théorèmes dans un cadre plus générique comme les nombres complexes pour les matrices. Coquelicot est une extension conservative de l'analyse classique de la bibliothèque standard de Coq et nous avons démontré les théorèmes de correspondance entre les deux formalisations. Nous avons testé la bibliothèque sur plusieurs cas d'utilisation : sur une épreuve du Baccalauréat, pour les définitions et les propriétés des fonctions de Bessel ainsi que pour la solution de l'équation des ondes en dimension 1.