Master recherche PLMT

Rapport de stage

présenté en juin 2005

par

Nicolas JULIEN

Vérification formelle d'arithmétique réelle exacte et fonctions analytiques

Responsable du stage : Yves BERTOT
Rapporteur du stage : Enrico FORMENTI

Table des matières

1 Introduction

L'objectif de ce stage est de faire la vérification formelle de preuves mathématiques concernant la correction d'une bibliothèque de calculs d'arithmétique réelle exacte. Ces calculs sont fondés sur une représentation des réels par des listes potentiellement infinies de chiffres signés d'une base quelconque, à l'aide du principe de co-induction.

Les outils usuels de calcul manipulent les nombres réels par une représentation finie : les nombres à virgule flottante. Cette représentation, même si elle est parfois paramétrable, a une précision qui est fixée avant les calculs. Les nombres manipulés ne sont en réalité qu'un sous ensemble des nombres rationnels. Les calculs effectués sur une telle représentation manipulent donc en général des approximations, ce qui entraîne des erreurs d'arrondi, dont des compositions répétées peuvent aboutir à des résultats complètement différents de la réalité.

L'arithmétique réelle exacte fournit des calculs que l'on peut effectuer avec une précision arbitraire ; le résultat est un intervalle aussi fin que l'on veut, dont on a la garantie qu'il contient le résultat escompté. Le prix à payer est un temps de calcul plus important que pour les nombres flottant. De plus la représentation en mémoire va demander plus d'espace.

2 État de l'art

2.1 Représentation à chiffres signés

L'arithmétique réelle exacte nécessite un moyen de représenter de façon exacte des objets infinis. On peut représenter un nombre réel de $[0, 1]$ par la suite infinie de ses décimales. Par exemple $\frac{1}{3}$ est représenté par $333333333333\ldots$ Connaître un préfixe fini d'une telle séquence, permet de connaître le nombre avec une certaine précision. En effet, si la séquence représentant un nombre de $[0, 1]$ commence par 1415 alors ce nombre est compris dans $[\frac{1415}{10000}, \frac{1416}{10000}]$.

Cette notation peut être étendue à une base quelconque, la base 2 étant la plus couramment utilisée en informatique. De plus on peut élargir l'intervalle des nombres représentés à $[-1, 1]$ grâce à l'utilisation de chiffres signés. Ainsi, en base β, une suite infinie de chiffre d_n représente le nombre :

$$\sum_{i=1}^{\infty} \frac{d_i}{\beta^i} \qquad \text{tel que} - \beta < d_i < \beta.$$

Cette représentation est inspirée en informatique des travaux d'Avizienis [1]. Plus ancienne dans le domaine des mathématiques, elle est attribuée par certains auteurs à Cauchy.

La notation $k :: x$ signifiera la séquence infinie commençant par le chiffre k et continuant par la séquence infinie x. Si d est un chiffre positif, on notera le chiffre négatif opposé $\bar{d}$. Enfin on notera $[x]_\beta$ le réel représenté par la séquence infinie x de chiffres signés de la base β. Par exemple en base 2, le nombre -1 sera la séquence infinie $\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\ldots$

Cette représentation est redondante car un même nombre peut admettre une infinité de représentations. Par exemple dans la base β, soit x une séquence infinie de chiffres signés, $0 :: \bar{1} :: x$ et $\bar{1} :: \beta{-}1 :: x$ représentent le même nombre :

$$\frac{0}{\beta} + \frac{-1}{\beta^2} + \frac{[x]_\beta}{\beta^2} = \frac{-1}{\beta} + \frac{\beta - 1}{\beta^2} + \frac{[x]_\beta}{\beta^2} = \frac{-1}{\beta^2} + \frac{[x]_\beta}{\beta^2}$$

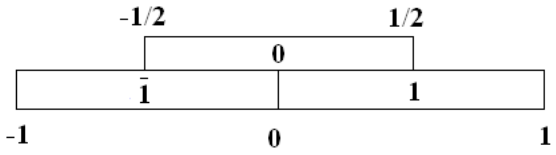
Avizienis [1] propose une représentation des approximations de nombres réels par une séquence finie de chiffres signés d'une base arbitraire. Il utilise des chiffres signés afin d'améliorer l'addition en supprimant le phénomène de la propagation des retenues. En effet, le résultat de l'addition de deux chiffres aura plusieurs représentations possibles grâce à la redondance. On pourra toujours choisir un résultat tel que la retenue soit $\bar{1}$, 0 ou 1 et le chiffre résultat ne soit ni le plus petit chiffre de la base ni le plus grand. Ainsi si on doit encore ajouter une retenue à ce résultat, ça ne changera pas la retenue de ce calcul. Par exemple si on additionne 4 et 5 en base 10 on peut rendre le chiffre 9 et la retenue 0. Mais si la retenue précédente est 1 alors la retenue sera changée et il faudra la propager. En revanche, comme $9 = 10 - 1$, on peut rendre le chiffre $\bar{1}$ et la retenue 1. Ainsi quelle que soit la retenue précédente, la retenue de ce calcul est inchangée. Lors de l'addition de deux nombres, on pourra donc faire dans un premier temps toutes les additions chiffre à chiffre en parallèle, et dans un deuxième temps l'ajout de toutes les retenues, aussi en parallèle.

Dans le cadre de l'arithmétique réelle exacte, les chiffres signés ne sont plus utilisés seulement pour améliorer les calculs, mais pour les rendre possibles.

2.2 Raisonner en terme d'intervalles

On peut interpréter nos chiffres signés de la base β comme des fonctions qui transforment un intervalle de longueur l en un sous intervalle de longueur $\frac{l}{\beta}$. Ainsi, déterminer un préfixe de n chiffres d'un réel de $[-1, 1]$ est équivalent à le connaître avec une précision de $\frac{1}{\beta^n}$. C'est d'ailleurs sur cette idée d'intervalles qu'il est le plus naturel de raisonner.

En terme d'intervalle, la redondance se traduit de la manière suivante : deux intervalles consécutifs se chevauchent, le milieu de l'un étant le début du suivant et la fin du précédent. Par exemple, si la base est 2, $\bar{1}$ correspond à la fonction qui prend la moitié gauche d'un intervalle, 1 la moitié droite et 0 la moitié qui est au centre, ce qui donne pour $[-1, 1]$: $[-1, 0]$, $[0, 1]$ et $[-\frac{1}{2}, \frac{1}{2}]$. 0 représente la partie redondante car $\bar{1}$ et 1 suffisent à partitionner un intervalle.



L'exemple suivant illustre l'importance de cette redondance. Pour faire l'addition de deux nombres réels exacts dans la représentation non redondante, il faut tout d'abord déterminer si le résultat commence par -1 ou 1, c'est-à-dire si ce nombre est positif ou pas. Or ce test n'est, en général, pas décidable. En revanche si l'on dispose du chiffre 0 on pourra toujours déterminer le premier chiffre du résultat, quitte à regarder les deux premiers chiffres des deux membres à additionner.

2.3 Comment effectuer des calculs

La technique générale pour calculer le résultat d'une fonction est d'extraire suffisamment de chiffres de ses arguments pour qu'on soit capable de donner un encadrement du résultat tel que l'on peut décider de son premier chiffre. Puis on ajuste le reste et l'on recommence le processus indéfiniment jusqu'à la précision souhaitée.

Dans leurs travaux, Edalat et Heckmann [6] proposent une formalisation de cette méthode au moyen de transformations fractionnelles linéaires : LFT (*Linear Fractional Transformation*). Une LFT d'ordre 1, appelée aussi homographie, est une fonction de la forme $x \mapsto \frac{ax+b}{cx+d}$. Dans la représentation décrite en section 2.1, l'opération qui consiste à interpréter la séquence x où on a ajouté le chiffre d en tête correspond à une telle transformation : $f_d : [x]_\beta \mapsto [d :: x]_\beta = \frac{d+[x]_\beta}{\beta}$. On peut donc associer à chaque chiffre une transformation et à une séquence $d_1 d_2 d_3 d_4 \ldots$ la composition infinie de transformations $f_{d_1} \circ f_{d_2} \circ f_{d_3} \circ f_{d_4} \circ \ldots$

2.4 Preuves formelles d'arithmétique réelle exacte

Le but de l'arithmétique réelle exacte est de fournir des résultats garantis. Un moyen de garantir ceux-ci est de démontrer leur validité au moyen d'un assistant de preuve.

2.4.1 Assistants de preuve

La vérification par des moyens "mécaniques" de preuves mathématiques a donné lieu à des travaux de mathématiques depuis la fin du XIX^{e}s. On sait que les preuves mathématiques ne peuvent pas être découvertes uniquement par des machines, mais la vérification d'une preuve correcte peut être effectuée par une machine. Les travaux dans ce domaines ont établi plusieurs façons de représenter les preuves et chacune de ces façons a mené à l'apparition d'un système informatique différent. A l'heure actuelle, il existe de nombreux systèmes dont les plus importants semblent être HOL [8], PVS [12], ACL2 [9] et Coq [5, 3].

2.4.2 Coq et la co-induction

Coq est un assistant de preuve développé par l'INRIA et fondé sur la théorie des types et plus particulièrement le calcul des constructions inductives [11]. Il comprend un langage de programmation fonctionnel typé dans lequel on peut écrire des programmes sur lesquels on peut démontrer ensuite des propriétés. Afin d'assurer que toutes les expressions écrites ont un sens, Coq n'autorise pas l'écriture de programmes qui ne se terminent pas.

En Coq, on peut représenter par des "types inductifs" les structures de données définies par induction comme les arbres finis. Par exemple l'ensemble des entiers naturels.

```
Inductive ℕ : Set :=
  | O : ℕ
  | S : ℕ →ℕ.
```

O et S sont appellés des *constructeurs*. On peut ensuite écrire des fonctions qui sont récursives par rapport à un seul des arguments qui doit évidemment

avoir un type inductif. Pour être acceptée par le système, la fonction doit faire ses appels récursifs uniquement sur des sous termes de cet argument. De plus la logique du système garantit qu'aucun terme d'un type inductif n'a de branche infinie. De cette manière on a l'assurance de définir des fonctions récursives dont l'exécution se termine.

```
Fixpoint plus (n m: ℕ) {struct n} : ℕ :=
  match n with
    | O => m
    | S p => S (plus p m)
  end.
```

Coq nous permet aussi de définir des types co-inductif qui peuvent contenir des termes avec des branches infinies, grâce aux travaux de Gimenez [7]. On peut par exemple définir le type des séquences infinies d'objets d'un même type A :

```
CoInductive stream (A:Set): Set :=
  | Cons : A → stream A → stream A.
```

Vu le caractère infini de ces objets, leur évaluation est retardée étape par étape au fur et à mesure qu'ils sont explorés, de manière *paresseuse*. On peut décrire des fonctions co-récursives qui permettent de construire ces objets infinis, les appels co-récursifs n'étant autorisés que s'ils sont *productifs*, c'est-à-dire qu'ils ne sont effectués qu'après avoir produit une partie non nulle de l'objet. Cette condition permet de garantir la terminaison de ces fonctions. Plus précisément, le calcul d'une partie finie de l'objet se fera en un temps fini. On peut définir par exemple la séquence infinie d'un même objet :

```
CoFixpoint repeat (A:Set) (a:A) : stream A :=
  Cons a (repeat A a).
```

Ici à chaque étape on produit un a et l'appel co-récursif permet de perpétuer cette production.

2.4.3 Formalisation en Coq

On peut considérer différentes approches pour définir un ensemble de nombres.

- Par axiomatisation : on peut postuler l'existence d'un ensemble de constantes, d'opérations et des propriétés sur ces opérations. C'est de cette façon que sont définis les réels en Coq [10].
- On peut aussi choisir une représentation, définir des opérations et démontrer qu'elles satisfont certaines propriétés. C'est ainsi que procèdent Ciaffaglione et Di Gianantonio [4] dans leur travaux sur l'arithmétique réelle exacte.
- Enfin si une définition existe déjà, on peut définir une représentation, une relation reliant cette représentation à la définition existante, puis définir les opérations et obtenir leur propriétés en utilisant cette relation. C'est l'approche choisie par Bertot [2].

Ciaffaglione et Di Gianantonio [4] proposent une formalisation des réels exacts en Coq pour la base 2. Ils utilisent la représentation décrite en section 2.1 et décrivent les réels par des suites infinies de $\bar{1}$, 0 ou 1. Ils définissent les opérateurs de base et démontrent les propriétés nécessaires pour identifier l'ensemble des nombres réels.

Une autre formalisation en Coq est celle de Bertot [2] dont ce stage constitue une extension. Il se place dans le cadre de la base 2 et considère les nombres réels de $[0,1]$. Il utilise les chiffres 0, $\frac{1}{2}$ et 1 qu'il désigne par L, C et R. Ces chiffres découpent l'intervalle $[0,1]$ en $[0,\frac{1}{2}]$, $[\frac{1}{4},\frac{3}{4}]$ et $[\frac{1}{2},1]$. Les approximations d'un nombre correspondant à la connaissance de ses n premiers chiffres $(\sum_{i=1}^{n}\frac{d_i}{2^i})$ constituent donc une suite croissante, ce qui a pour conséquence de simplifier de nombreuses preuves.

De plus, il choisit de mettre en relation les nombres exprimés par sa représentation avec la définition axiomatique des réels de Coq afin de tirer avantage de toutes les propriétés sur les réels ainsi que les théorèmes et les différentes tactiques de preuves définis dans les bibliothèques de Coq. La preuve de la correction d'une fonction f, étant donnée f_{Coq} la fonction ayant le même sens et définie sur les réels de Coq, consistera à montrer "si $x_1,\ldots,x_n$ representent les nombres réels de Coq $r_1,\ldots,r_n$ et si $f_{Coq}(r_1,\ldots,r_n)\in[0,1]$ alors $f(x_1,\ldots,x_n)$ représente $f_{Coq}(r_1,\ldots,r_n)$". Enfin il propose la première formalisation du calcul de séries entières et décrit le calcul de la multiplication comme un cas particulier de série entière.

2.4.4 Formalisation du calcul des séries entières

Pour calculer les séries entières dont les termes sont positifs, la technique proposée dans [2] repose sur le calcul de

$$f(x,y,n) = x + y \times \sum_{i=n}^{\infty} a_i$$

Le problème qu'on se pose est de déterminer le premier chiffre d du résultat et de trouver x', y' et n' tel que le calcul soit de la forme suivante :

$$f(x,y,n) = d :: (x' + y' \times \sum_{i=n'}^{\infty} a_i) = d :: f(x',y',n')$$

Pour déterminer ce premier chiffre il faut savoir placer $f(x,y,n)$ dans l'un des intervalles $[0,\frac{1}{2}]$, $[\frac{1}{4},\frac{3}{4}]$ ou $[\frac{1}{2},1]$.

La technique proposée est de trouver un $n' \geq n$ tel que $0 \leq y \times \sum_{i=n'}^{\infty} a_i \leq \frac{1}{8}$, en trouvant une fonction majorant assez finement $\sum_{i=n'}^{\infty} a_i$. On cherche ensuite à obtenir $x + y \times \sum_{i=n}^{n'-1} a_i$ avec une précision suffisante pour que l'erreur induite par le terme $y \times \sum_{i=n'}^{\infty} a_i$ ne puisse pas mener à un chiffre erroné. En regardant les différents cas possibles, on montre qu'il suffit de regarder au plus deux chiffres.

Par exemple, si $v = x + y \times \sum_{i=n}^{n'-1} a_i$ est de la forme $C :: C :: v'$, on sait que $\frac{1}{4} \leq v \leq \frac{5}{8}$, et en conséquence, $\frac{1}{4} \leq v + y \times \sum_{i=n'}^{\infty} a_i \leq \frac{3}{4}$, donc le premier chiffre sera un C. Comme C correspond à la transformation $x \mapsto \frac{x+\frac{1}{2}}{2}$, on a

$$\begin{aligned} f(x,y,n) &= v + y \times \sum_{i=n'}^{\infty} a_i \\ &= \frac{2v - \frac{1}{2} + 2y \times \sum_{i=n'}^{\infty} a_i + \frac{1}{2}}{2} \\ &= C :: f(C :: v', 2y, n'). \end{aligned}$$

Un exemple d'application de la technique est le calcul du nombre d'Euler e [2].

3 Présentation informelle du travail

Ce stage poursuit le travail sur les séries, tout d'abord par l'ajout du calcul de la soustraction et du nombre π et la preuve de leur correction. Ensuite j'ai transposé le calcul des séries dans le cadre des chiffres signés $\bar{1}$, 0 et 1, décrivant ainsi des nombres de $[-1, 1]$. Enfin la dernière partie du stage a consisté en l'adaptation à une base quelconque des calculs et des preuves. La véritable difficulté de ce travail a été de formaliser ces calculs et vérifier les preuves au moyen du système Coq, ce qui sera décrit dans la partie suivante. Tout le développement en Coq est disponible sur internet[1].

3.1 Soustraction et π

Dans cette première partie du stage, on a rajouté quelques définitions au travail existant et démontré leur validité. On se place donc dans le cadre de nombres de $[0, 1]$ représenté en base 2 par les chiffres L, C et R.

Pour calculer la soustraction, on définit tout d'abord une opération stable proche de la soustraction : $(x, y) \mapsto \frac{x-y+1}{2}$. Ensuite on définit la fonction

$$x \mapsto \begin{cases} 2x - 1 & \text{si } x \geq \frac{1}{2} \\ 0 & \text{sinon} \end{cases}$$

La composition des deux permet de définir une opération entre deux nombres dont on montre qu'elle correspond à la soustraction quand le résultat attendu est dans $[0, 1]$.

Un autre calcul défini dans ce cadre est celui du nombre $\frac{\pi}{4}$ au moyen d'une série entière à termes positifs. En effet, on se sert de l'égalité

$$\frac{\pi}{4} = \arctan \frac{1}{2} + \arctan \frac{1}{3}$$

et le calcul de la série

$$\arctan x = \sum_{i=0}^{\infty} \frac{-1^i x^{2i+1}}{2i+1}$$

en $\frac{1}{n}$, où n est un entier supérieur à deux. Ensuite après avoir trouvé une majoration suffisante de cette série, on utilise la technique de calcul des séries. Enfin on montre la correction du calcul en montrant d'abord quelques lemmes comme par exemple la validité de la majoration utilisée.

3.2 Passage à la représentation des réels de $[-1, 1]$

Mon travail a ensuite consisté à modifier le travail effectué sur les réels de l'intervalle $[0, 1]$, afin de considérer les réels de $[-1, 1]$, c'est-à-dire intégrer la notion de signe. Les nombres ne seront plus représentés par des chiffres *fractionnaires* (0, $\frac{1}{2}$ et 1) mais par des chiffres signés ($\bar{1}$, 0 et 1). On va donc perdre les facilités de preuves qui résultaient du fait que les chiffres étaient tous positifs, mais on va pouvoir exprimer une partie plus intéressante des réels.

[1] http ://www-sop.inria.fr/marelle/personnel/Nicolas.Julien/stageMaster/

3.2.1 Modification de la représentation

Comme on reste dans le cadre de la base 2, on représente toujours les nombres comme des séquences infinies des chiffres L, C et R. En revanche ces chiffres représentent désormais $\bar{1}$, 0 et 1. On peut aussi voir ces chiffres comme un découpage de $[-1,1]$ en $[-1,0]$, $[\frac{-1}{2},\frac{1}{2}]$ et $[0,1]$. On peut enfin les interpréter comme les transformations $L : x \mapsto \frac{x-1}{2}$, $C : x \mapsto \frac{x}{2}$ et $R : x \mapsto \frac{x+1}{2}$.

3.2.2 Modification des calculs

La plupart des calculs ont dû être adaptés à ce changement d'intervalle. Dans de nombreux calculs, comme on ne manipulait au départ que des nombres positifs, il ne se passait rien de problématique à la borne 0, celle ci ne risquant pas d'être dépassée. En revanche la borne 1 était, en général, plus subtile à traiter, demandant souvent de connaître plus de chiffres du paramètre pour décider de la progression du calcul. Avec le changement de paramètre la borne -1 pose aussi des problèmes et augmente la taille des algorithmes. Par exemple dans le cas de la multiplication par 2, sur $[0,1]$ l'intervalle image est $[0,2]$ alors qu'en $[-1,1]$ l'image est $[-2,2]$, ce qui entraîne une complication des deux cotés de l'intervalle.

Heureusement dans la plupart des cas étudiés, le problème se résout en regardant un chiffre de plus et en procédant ainsi.

- Si le premier chiffre est 1, le nombre est positif et on peut appliquer à peu près l'algorithme défini sur $[0,1]$.
- Si le premier chiffre est $\bar{1}$, le nombre est négatif, et on peut appliquer un raisonnement symétrique.
- Enfin le cas où le premier chiffre est 0 est en général trivial.

Le calcul des séries suit toujours la même technique. Bien évidemment comme on ne travaille plus sur le même intervalle les encadrements nécessaires ne sont plus les mêmes. On définit comment calculer la fonction

$$f(x,y,n) = x + y \times \sum_{i=n}^{\infty} a_i.$$

On doit d'abord chercher un $n' \geq n$ tel que

$$\frac{-1}{8} \leq y \times \sum_{i=n'}^{\infty} a_i \leq \frac{1}{8},$$

ce qui est déjà un peu plus contraignant. Ensuite on cherche à obtenir un encadrement suffisant de $x + y \times \sum_{i=n}^{n'-1} a_i$ pour décider du premier chiffre du résultat de la fonction. On montre qu'il suffit dans le pire des cas de calculer ses trois premiers chiffres, ce qui est un peu moins bien qu'en $[0,1]$. Le reste de la méthode est inchangé.

Bien que la méthode soit similaire, une légère complication apparaît. En effet, une partie de la technique consiste à calculer les premiers chiffres de la somme $x + y \times \sum_{i=n}^{n'-1} a_i$. La preuve du calcul d'une série nécessite donc la preuve que cette somme est correcte, en particulier que le resultat est compris dans $[-1,1]$. Cette preuve triviale dans le cadre des séries positives est devenue compliquée et en particulier dans le cadre de la multiplication.

3.2.3 Modification des preuves

Tout d'abord, les modifications dans les preuves ont été d'étendre le domaine de correction du calcul au nouvel intervalle. Les preuves de correction nécessitent régulièrement de démontrer que certains nombres sont compris dans l'intervalle $[-1, 1]$ en raison soit de l'application de l'hypothèse de co-récursion, soit de l'application du théorème de la correction d'une autre fonction utilisée par celle dont on veut faire la preuve. Par exemple dans une preuve de calcul de série, on a besoin de montrer que la somme $x + y \times \sum_{i=n}^{n'-1} a_i$ est correcte. Le théorème de correction de l'addition n'est applicable que si le résultat est dans $[-1, 1]$ et génère donc une sous preuve de ce genre. Ce style de preuves était plus aisé sur $[0, 1]$, le coté 0 étant trivial, tandis que l'autre borne, plus compliquée, nécessitait en général un lemme préalable. Avec ce changement il faut compter sur deux bornes compliquées à démontrer.

De plus, les algorithmes ayant vu leur taille augmenter, les preuves l'ont été aussi en conséquence avec heureusement souvent des cas ajoutés se prouvant de façon symétrique à d'autres en fonction du signe.

3.3 Passage à la représentation à base quelconque

Dans cette partie du stage nous nous sommes intéressés à décrire la représentation des réels, les calculs et leur preuve de correction, pour une base quelconque, ce qui est un apport original dans le domaine. Pour rendre plus simples certains algorithmes nous avons pris la décision de ne travailler qu'avec des bases paires. L'ajout de cette base dans la représentation a considérablement augmenté la complexité de description des calculs et la difficulté des preuves. De plus, raisonner en voyant les chiffres comme une façon de découper les intervalles n'est parfois plus d'une grande aide pour appréhender le déroulement d'un calcul.

3.3.1 Nouvelle représentation des réels

Dans notre nouvelle représentation, un nombre réel de $[-1, 1]$ sera une séquence infinie de chiffres de la même base. La relation qui permet de passer de notre représentation aux nombres réels de Coq exprimera "pour tout chiffre k de la base β, si la séquence infinie x de chiffres de la base β représente r et si $-1 \leq r \leq 1$ alors $k :: x$ représente le nombre $\frac{k+r}{\beta}$"

3.3.2 Définition des opérations de base

Les intervalles représentés par les chiffres de la base sont de la forme : $I_k = [\frac{k-1}{\beta}, \frac{k+1}{\beta}]$. En conséquence la taille du chevauchement de deux intervalles consécutifs est $|I_k \cap I_{k+1}| = \frac{1}{\beta}$. Tout intervalle inclus dans $[-1, 1]$ de taille inférieure à $\frac{1}{\beta}$ sera donc forcément compris dans un intervalle I_k. Les algorithmes des calculs consisteront donc à raffiner les arguments jusqu'à ce qu'on soit capable de donner un encadrement du résultat de cette taille.

La grande différence avec le travail en base 2 est que lorsqu'on doit connaître la valeur d'un chiffre en tête, on ne peut plus regarder exhaustivement tous les chiffres possibles, vu qu'ils dépendent de la base. Il faut systématiquement les regrouper en fonction des besoins de l'algorithme.

Par exemple dans le cas de la demi-somme $(x, y) \mapsto \frac{x+y}{2}$, si la somme des premiers chiffres de x et y est paire alors on peut décider tout de suite du premier chiffre du résultat sinon il faut regarder un chiffre de plus pour x ou y.

3.3.3 Définition des séries entières

Le calcul des séries suit toujours la même technique, les conditions d'encadrement suffisantes étant modifiées pour s'adapter à la base. On se place dans un cadre où la base β est supérieure à 3. Pour pouvoir calculer une série $\sum_{i=0}^{\infty} a_i$, convergeant dans l'intervalle $[-1, 1]$, on définit toujours le calcul d'une fonction

$$f(x, y, n) = x + y \times \sum_{i=n}^{\infty} a_i.$$

On calcule $p \geq n$, tel que $-\frac{\beta-2}{2\beta^2} \leq y \times \sum_{i=p}^{\infty} a_i \leq \frac{\beta-2}{2\beta^2}$. Comme la série est convergente et y et β sont fixés, on sait qu'un tel p existe. Ensuite on regarde les deux premiers chiffres de $x' = x + y \times \sum_{i=n}^{p-1} a_i$, ce qui nous permet de connaître un encadrement x' de longueur $\frac{2}{\beta^2}$. Ces 2 encadrements nous permettent alors de déduire un encadrement de $f(x, y, n) = x' + \sum_{i=p}^{\infty} a_i$ de $\frac{1}{\beta}$ ce qui est suffisant pour décider de son premier chiffre k. On calcule la séquence s des autres chiffres par un appel co-récursif de la même manière que décrit précédemment.

$$\begin{aligned}
f(x, y, n) &= \frac{k + [s]_\beta}{\beta} = x + y \times \sum_{i=n}^{p-1} a_i + y \times \sum_{i=p}^{\infty} a_i \\
\text{En conséquence} \quad [s]_\beta &= \beta \times (x + y \times \sum_{i=n}^{p-1} a_i + y \times \sum_{i=p}^{\infty} a_i) - k \\
&= \beta \times (x + y \times \sum_{i=n}^{p-1} a_i) - d + \beta \times y \times \sum_{i=p}^{\infty} a_i \\
&= f(\beta \times (x + y \times \sum_{i=n}^{p-1} a_i) - d, \beta \times y, p)
\end{aligned}$$

4 Présentation formelle du travail

4.1 La représentation "LCR" sur $[0, 1]$

Dans le développement qui sert de point de départ à notre travail, on trouve une définition en Coq d'un type co-inductif pour représenter des séquences infinies d'objets et le type des chiffres de la base.

```
CoInductive stream (A:Set) : Set :=
  Cons : A → stream A → stream A.

Inductive idigit := L | C | R.
```

L'expression `Cons a S` sera notée par la suite $a :: S$. Comme on l'a déjà vu, ces chiffres peuvent être vus comme les transformations suivantes : $L : x \mapsto \frac{x}{2}$, $C : x \mapsto \frac{x+\frac{1}{2}}{2}$ et $R : x \mapsto \frac{x+1}{2}$. Et une séquence infinie peut être interprétée

par la composition infinie de ces transformations. C'est cette idée qui va nous permettre de mettre en relation notre représentation avec la définition des réels de Coq. Cette relation exprime donc "Si la séquence s représente le réel r et que $-1 \leq r \leq 1$ alors la séquence $L :: s$ représente le réel $\frac{r}{2}$ et de même pour C et R". On formalise cette relation en Coq au moyen d'une propriété co-inductive.

```
CoInductive represents : stream idigit → ℝ →Prop :=
  | reprL : ∀ s r, represents s r → 0 ≤ r ≤ 1 →
      represents (L::s) x/2
  | reprC : ∀ s r, represents s r → 0 ≤ r ≤ 1 →
      represents (C::s) (x+1/2)/2
  | reprR : ∀ s r, represents s r → 0 ≤ r ≤ 1 →
      represents (R::s) (x+1)/2.
```

Ensuite on va utiliser des fonctions co-récursives pour définir des réels. Premièrement on va définir des constantes, par exemple le réel 0 sera la séquence constituée d'une infinité de L. Ça se décrit en Coq par une fonction co-récursive signifiant, "zero est la séquence infinie qui commence par L, suivi par la séquence infinie zero elle même".

```
CoFixpoint zero : stream idigit := L::zero.
```

On va pouvoir grâce au prédicat `represents` formaliser et prouver le théorème qui permet de dire que l'on a bien défini une représentation du réel 0

```
Theorem represents_zero : represents zero 0.
```

Intéressons nous maintenant à la formalisation en Coq de la technique de calcul des séries entières à termes positifs. Pour rappel, il faut d'abord que l'on décrive une fonction

$$f(x, y, n) = x + y \times \sum_{i=n}^{\infty} a_i.$$

Pour calculer cette fonction, on commence par trouver $p \geq n$ tel que

$$0 \leq y \times \sum_{i=p}^{\infty} a_i \leq \frac{1}{8}.$$

Ensuite une fois ce calcul effectué, la partie qui consiste à déterminer le premier chiffre du résultat en lisant les deux premiers chiffres de $v = x + y \times \sum_{i=n}^{p-1} a_i$, et à faire l'appel récursif ne dépend pas de la série et peut être formalisée une fois pour toute.

- Si v s'écrit $R :: v'$ alors on sait que $v + y \times \sum_{i=p}^{\infty} a_i \geq \frac{1}{2}$ et le résultat commence donc par un R et l'on peut faire un appel co-récursif sur le reste v'
- Si v s'écrit $L :: L :: v''$ ou $L :: C :: v''$, alors on sait que $v \leq \frac{3}{8}$ donc $v + y \times \sum_{i=p}^{\infty} a_i \leq \frac{1}{2}$, le résultat commence par un L et l'appel co-récursif se fait sur $L :: v''$ et $C :: v''$.
- Si v s'écrit $C :: L :: v''$ ou $C :: C :: v''$, alors on sait que $\frac{1}{4} \leq v \leq \frac{5}{8}$ donc $\frac{1}{4} \leq v + y \times \sum_{i=p}^{\infty} a_i \leq \frac{3}{4}$, le résultat commence par un C et l'appel co-récursif se fait sur $L :: v''$ et $C :: v''$.
- Si v s'écrit $L :: R :: v''$ ou $C :: R :: v''$ alors on utilise la redondance de la notation pour se ramener à un cas déjà traité : $L :: R :: v'' = C :: L :: v''$, $C :: R :: v'' = R :: L :: v''$.

On définit donc une fonction qui, étant donné une représentation x, une série représentée par sa fonction $f(x, y, n)$ et un argument `a` générique qui contiendra y, n et éventuellement d'autres paramètres servant au calcul de f, calcule le premier chiffre de la façon expliquée ci-dessus et continue le calcul de la série.

```
Definition series_body (A:Set)
  (series : stream idigit → A → stream idigit)
  (x : stream idigit) (a:A) : stream idigit :=
    let (dig,x'') := x in
    match dig with
      | R =>  R::series x'' a
      | L => match x'' with
         | R::x3 => C::series (L::x3) a
         | _ => L::series x'' a
         end
      | C => match x'' with
         | R::x3 => R::series (L::x3) a
         | _ => C::series x'' a
         end
      end.
```

4.2 Calcul de π

Grâce à la formalisation des séries entières, nous avons une méthode pour calculer le nombre $\pi = \arctan\frac{1}{2} + \arctan\frac{1}{3}$. Nous allons donc pour cela calculer la série entière $\arctan\frac{1}{N} = \sum_{i=0}^{\infty} \frac{-1^i}{N^{2i+1}(2i+1)}, N \geq 2$. On peut réécrire cette série de telle sorte que ses termes soient positifs :

$$\begin{aligned}
\arctan\frac{1}{N} &= \sum_{i=0}^{\infty}\left(\frac{1}{N^{4i+1}(4i+1)} - \frac{1}{N^{4i+3}(4i+3)}\right) \\
&= \sum_{i=0}^{\infty}\frac{N^2(4i+3)-(4i+1)}{N^{4i+3}(4i+3)(4i+1)}
\end{aligned}$$

Il nous faut maintenant trouver une fonction majorant cette série à partir du rang k.

$$\begin{aligned}
\sum_{i=k}^{\infty} \frac{N^2(4i+3)-(4i+1)}{N^{4i+3}(4i+3)(4i+1)} &\leq \sum_{i=k}^{\infty} \frac{N^2(4i+3)}{N^{4i+3}(4i+3)(4i+1)} \\
&\leq \sum_{i=k}^{\infty} \frac{1}{N^{4i+1}(4i+1)} \\
&\leq \sum_{i=0}^{\infty} \frac{1}{N^{4(i+k)+1}(4(i+k)+1)} \\
&\leq \frac{1}{N^{4k+1}(4k+1)} \times \sum_{i=0}^{\infty} \left(\frac{1}{N^4}\right)^i \\
&\leq \frac{1}{N^{4k+1}(4k+1)} \times \frac{N^4}{N^4-1} \\
&\leq \frac{1}{N^{4k-3}} \frac{1}{N^4-1} \\
&\leq \frac{1}{2^{4k-3}} \frac{1}{15} = \frac{1}{16^{k-1}} \frac{1}{30}
\end{aligned}$$

Il faut démontrer cette majoration en Coq pour qu'elle puisse servir à démontrer ensuite la correction du calcul de la série. Or en Coq il n'existe pas de fonction $a \mapsto \sum_{i=0}^{\infty} a_i$ car cette valeur n'est pas toujours définie. En revanche il existe une relation `infinit_sum a l`, où `a` est une fonction de $\mathbb{N}$ dans $\mathbb{R}$, qui exprime "la série $\sum_{i=0}^{\infty} a_i$ converge et sa limite est l".

Comme on veut définir la convergence de la série calculée à partir d'un rang p quelconque ($\sum_{i=p}^{\infty} a_i$), on va utiliser la relation sur la série ($\sum_{i=0}^{\infty} a_{i+p}$).

```
Definition terme_arctan_1_N (N i: ℤ) : R :=
  (N^2*(4*i+3)-4*i-1)/(N^(4*i+3)*(4*i+1)*(4*i+3)).

Theorem arctan_serie_shift_bound:
  ∀ N p r,
    2 ≤ N →
    1 ≤ p →
    infinit_sum (fun i : nat =>
      terme_arctan_1_N N (i+ p)) r →
      r ≤ 1/30 * 1/16^(p-1).
```

Pour démontrer cette preuve, il aura fallu d'abord démontrer de nombreux lemmes correspondant à certaines étapes de la preuve mathématiques et certaines propriétés générales nécessaires sur les puissances ou les séries.

```
Theorem Zpower_incr: ∀ (a b n : ℤ), 0 ≤ n →0 ≤ a ≤ b →a^n ≤ b^n.

Theorem scal_infinit_sum :
  ∀ (An : ℕ →ℝ) (x r: ℝ),
    infinit_sum An r → x ≠ 0 →
    infinit_sum  (fun i ∈ ℕ => ( An(i) * x)) (x * r).
```

On peut donc enfin définir la fonction qui va calculer

$$f(x,y,n) = x + y \times \sum_{i=n}^{\infty} \frac{N^2(4i+3)-(4i+1)}{N^{4i+3}(4i+3)(4i+1)}.$$

Comme on l'a vu pour la définition de `series_body`, les paramètres y et n sont englobés dans un type générique pour qu'on puisse y ajouter des informations permettant d'améliorer le calcul. Ici le paramètre générique représenté par s correspond à un quadruplet (y, n, N, log_y) tel que y et n sont les y et n de l'algorithme. N représente l'entier dont on veut calculer arctan $\frac{1}{N}$. Enfin log_y représente le logarithme à base 2 de y. Il faut donc trouver un n' tel que $y \times \sum_{i=n'}^{\infty} a_i \leq \frac{1}{8}$.

$$\begin{aligned} y \times \sum_{i=n'}^{\infty} \frac{N^2(4i+3)-(4i+1)}{N^{4i+3}(4i+3)(4i+1)} &\leq 2^{log\_y} \times \frac{1}{2^{4n'-3}} \frac{1}{15} \\ &\leq 2^{log\_y-4n'+3} \times \frac{1}{8} \\ &\leq \frac{1}{8} \qquad \text{si} \quad log\_y \leq 4n'+3 \end{aligned}$$

Donc pour calculer le nouveau n', si $log\_y \leq 4n+3$, le rang courant n est encore suffisant, sinon, comme n était valable à l'étape précédente, on a quand même $y \times \sum_{i=n}^{\infty} a_i \leq 2^{log\_y-4n+3} \times \frac{1}{8} \leq \frac{1}{4}$. On peut en déduire :

$$\begin{aligned} y \times \sum_{i=n+1}^{\infty} a_i &\leq 2^{log\_y-4(n+1)+3} \times \frac{1}{8} \\ &\leq \frac{1}{2^4} \times 2^{log\_y-4n+3} \times \frac{1}{8} \\ &\leq \frac{1}{2^4} \times \frac{1}{4}. \end{aligned}$$

Le nouveau n' est donc $n+1$. On utilise la fonction `rat_to_stream` qui permet d'obtenir la représentation d'un nombre rationnel.

```
CoFixpoint arctan_1_N_serie (x:stream idigit)(s :ℤ*ℤ*ℤ*ℤ)
    : stream idigit :=
  (*on décompose le paramètre générique en (y, n, N, log_y)*)
  let (y,n,N,log_y) := s in
    (*on calcule n' *)
    if log_y ≤ (4n-3) then
      series_body _ arctan_1_N_serie v (2y,n,N,log_y+1)
    else
   (*si n' = n+1, il faut ajouter a_n à x*)
     let c := (N^2(4n+3)-4n-1) in
       let d := (N^(4n+3)(4n+1)(4n+3)) in
         let x' := rat_to_stream (y×c) d in
            series_body _ arctan_1_N_serie (x+x')
                (2y,(n+1),N,log_y+1).
```

La majoration de la série utilisée pour ce calcul n'est valable qu'à partir du rang 1. On va donc calculer $a_0+1\times\sum_{i=1}^{\infty} a_i = f(a_0,1,1)$ pour calculer arctan $\frac{1}{N}$.

```
Definition arctan_1_N (N:ℤ): stream idigit :=
  arctan_1_N_aux (rat_to_stream  (3N^2-1) (3N^3)) (1,1,N,0).
```

On peut enfin définir $\frac{\pi}{4}$.

```
Definition pi_div_4 := arctan_1_N 2 + arctan_1_N 3.
```

La preuve du calcul de arctan $\frac{1}{N}$ se fait en trois étapes.
La fonction `arctan_1_N_serie` se sert de `series_body` pour calculer ses appels co-récursifs et les chiffres produits. Il faut donc tout d'abord montrer que l'utilisation de `series_body` est correcte, c'est-à-dire que les appels co-récursifs des différents cas possibles vont conserver certains invariants. Pour cela on suppose que la fonction `arctan_1_N_serie` est correcte sous certaines conditions. On suppose ensuite que ces conditions étaient valides à l'étape précédente. Enfin on montre que dans chaque cas lors de la production du chiffre et l'appel co-récursif, ces conditions sont conservées, ce qui nous permet d'appliquer le théorème supposé pour conclure. Voici l'énoncé du premier théorème.

```
Theorem arctan_1_N_series_body_correct :
  (*On suppose l'existence de la fonction arctan_1_N_serie
      ainsi que sa preuve*)
  ∀ (arctan_1_N_serie : stream idigit → ℤ*ℤ*ℤ*ℤ →stream
      idigit),
   (∀ v vr r y n N log_y,
    y = 2^log_y →
    2 ≤ N →
    1 ≤ y →
    1 ≤ n →
    log_y − 1 ≤ 4n − 3 →
    represents v vr →
    infinit_sum (fun k => terme_arctan_1_N N (k+n)) r →
    vr + y × r ≤ 1 →
    represents (arctan_1_N_series v (y, n, N, log_y)) (vr + y × r) →
    (*On suppose les invariants de l'étape précédente*)
    ∀ v vr r y n N log_y,
    y = 2^log_y →
    1 ≤ y →
    1 ≤ n →
    2 ≤ N →
    log_y ≤ 4n − 3 →
    represents v vr →
    infinit_sum (fun k => terme_arctan_1_N N (k+n)) r →
    vr + y × r ≤ 1 →
    (*On conclue que l'appel à series_body est correct*)
    represents (series_body _ arctan_1_N_series v
        (2y, n, N, log_y + 1)) (vr + y × r).
```

Ensuite on montre que la fonction `arctan_1_N_serie` est correcte. Le théorème est donc exactement celui supposé dans la preuve précédente. Cette preuve consiste grossièrement à montrer que lorque on fait appel à `series_body` les conditions de l'étapes en cours sont valides. Autrement dit que les conditions demandées dans la deuxième moitié de l'énoncé de la preuve précédente sont vérifiées.

```
Theorem arctan_1_N_aux_correct :
  ∀ v vr r y n N log_y,
    y = 2^log_y →
    2 ≤ N →
    1 ≤ y →
    1 ≤ n →
    log_y − 1 ≤ 4 × n − 3 →
```

```
represents v vr →
infinit_sum (fun k => terme_arctan_1_N N (k+n))) r →
vr + y × r ≤ 1 →
represents (arctan_1_N_aux v (y, n, N, log_y)) (vr + y × r).
```

Et enfin on vérifie que l'appel initial de la fonction est valide.

```
Theorem arctan_correct : ∀ r N, 2 ≤ N →
  infinit_sum (fun k => terme_arctan_1_N N (Z_of_nat k)) r →
  represents (arctan_1_N N) r.
```

La correction de ce théorème a pu être vérifiée au bout de 29 théorèmes et 1250 lignes de preuves.

4.3 Passage à l'intervalle $[-1, 1]$

Dans cette partie du travail nous avons redéfini le sens des chiffres L, C et R afin de représenter les réels de l'intervalle $[-1, 1]$. Puis nous avons adapté les algorithmes et leur preuve à ce nouvel intervalle.

Le nouveau sens donné aux chiffres s'interprète par les transformations $L : x \mapsto \frac{x-1}{2}$, $C : x \mapsto \frac{x}{2}$ et $R : x \mapsto \frac{x+1}{2}$. On peut donc formaliser la relation permettant d'exprimer qu'une séquence représente un réel de Coq.

```
CoInductive represents : stream idigit → ℝ →Prop :=
  | reprL : ∀ s r, represents s r → −1 ≤ r ≤ 1 →
     represents (L::s) (r−1)/2
  | reprC : ∀ s r, represents s r → −1 ≤ r ≤ 1 →
     represents (C::s) r/2
  | reprR : ∀ s r, represents s r → −1 ≤ r ≤ 1 →
     represents (R::s) (r+1)/2.
```

La modification des calculs et des preuves a pu être faite assez méthodiquement en regardant le premier chiffre des arguments. On fabrique alors différents cas à partir desquels on essaie de se ramener à l'algorithme défini sur $[0, 1]$.

Le calcul où cette méthode se reflète le mieux est celui de la multiplication par 2. Nous en profiterons pour voir un peu plus en détail comment se déroule une preuve. On définit tout d'abord le calcul de la fonction définie par morceaux suivante :

$$x \mapsto \begin{cases} 1 & \text{si } x \geq \frac{1}{2} \\ -1 & \text{si } x \leq -\frac{1}{2} \\ 2x & \text{sinon.} \end{cases}$$

Pour calculer cette fonction, on regarde les différents cas.

- Si x est de la forme $C :: x'$, alors on sait que $x = \frac{x'}{2}$. Le résultat est donc x'.
- Si x est de la forme $R :: x'$, alors on sait que $x \in [0, 1]$, on va donc calculer de la même façon que l'algorithme existant. On regarde donc le chiffre suivant :
  - Si x' est de la forme $L :: x''$, alors grâce à la redondance, on peut écrire x sous la forme $C :: R :: x''$ car les préfixes RL et CR sont équivalents. On se ramène donc au cas précédent et le résultat est donc $R :: x''$
  - Si x' est de la forme $R :: x''$ alors on peut en déduire que $x \geq \frac{1}{2}$ le résultat est donc 1.

- Si x' est de la forme $C :: x''$ alors on peut en déduire que $2x = 2\frac{\frac{x''}{2}+1}{2} = \frac{\frac{2(x''+1)}{2}+1}{2} = R :: (2 \times (R :: x''))$ c'est donc un appel co-récursif.
- Enfin si x est de la forme $L :: x'$, alors on sait que $x \in [-1, 0]$ et ce cas est symétrique au précédent.

La constante `one` est la séquence infinie de R et qui représente donc le réel 1. De même `sub_one` est la séquence infinie de L qui représente le réel -1. L'algorithme s'écrit alors de la façon suivante en Coq :

```
CoFixpoint mult2 (x:stream idigit) : stream idigit :=
  match x with
    | C::x' => x'
    | R::x' =>
      match x' with
        | L::x'' => R::x''
        | R::_   => one
        | C::x'' => R::mult2 (R::x'')
      end
    | L::x' =>
      match x' with
        | R::x'' => L::x''
        | L::_ => sub_one
        | C::x'' => L::mult2 (L::x'')
      end
  end.
```

Pour montrer que cette définition calcule bien la fonction que l'on espère, il nous faut démontrer le théorème suivant :

```
Theorem mult2_correct : ∀ (x : stream idigit) (u : ℝ),
  -1/2 ≤ u ≤ 1/2 →represents x u →represents (mult2 x) (2u)
```

Coq nous affiche alors la liste des hypothèse le but à démontrer en cours qui est l'énoncé du théorème séparé par une ligne des hypothèses qui pour l'instant sont inexistantes.

```
1 subgoal

============================
∀ (x : stream idigit) (u : ℝ), -1/2 ≤ u ≤ 1/2→
represents x u → represents (mult2 x) (2u)
```

Comme notre but est une relation co-inductive, nous pouvons utiliser la tactique `cofix` qui permet de nous aider à résoudre ce genre de preuve. Cette tactique copie le but courant dans une hypothèse sans changer le but. Nous pourrons donc utiliser cette hypothèse de "récurrence" plus tard quand nous aurons prouvé une étape de la preuve et qu'il nous reste à prouver la partie "à l'infini". On pourrait être tenté de l'appliquer tout de suite, l'hypothèse ayant le même énoncé que le but, mais nous construirions alors un terme de preuve non productif, ce qui est contraire aux besoins de la co-induction. Notre nouveau but est alors :

```
1 subgoal
∀ (x : stream idigit) (u : ℝ), -1/2 ≤ u ≤ 1/2→
represents x u → represents (mult2 x) (2u)
```

```
============================
∀ (x : stream idigit) (u : ℝ), -1/2 ≤ u ≤ 1/2 →
represents x u → represents (mult2 x) (2u)
```

On applique alors la tactique `intros` qui permet de fixer les variables quantifiées universellement et de nommer les faits connus.

```
intros x u H Hrep.

1 subgoal

mult2_correct : ∀ (x : stream idigit) (u : ℝ),
   -1/2 ≤ u ≤ 1/2 →represents x u →represents (mult2 x) (2u)
x : stream idigit
u : ℝ
H : -1/2 ≤ u ≤ 1/2
Hrep : represents x u
============================
 represents (mult2 x) (2u)
```

La suite de la preuve va dépendre de la façon dont on calcule `mult2 x`. On va donc faire suivre à la preuve un raisonnement par cas sur x au moyen d'une série de commandes un peu compliquée et dont l'explication n'est pas utile à la compréhension de la preuve.

```
str_unfold (mult2 x).
destruct x as [ [ | | ] x'];[ destruct x' as [ [ | | ] x''] |
                              destruct x' as [ [ | | ] x''] |
              change (represents (str_decompose x') (2*u));
              rewrite <- str_dec_thm ]; simpl.

7 subgoals

...
Hrep : represents (L::L :: x'') u
============================
represents (L :: sub_one) (2u)

subgoal 2 is:
 represents (L :: x'') (2u)
subgoal 3 is:
 represents (L :: mult2 (L :: x'')) (2u)
subgoal 4 is:
 represents (R :: x'') (2u)
subgoal 5 is:
 represents (R :: one) (2u)
subgoal 6 is:
 represents (R :: mult2 (R :: x'')) (2u)
subgoal 7 is:
 represents x' (2u)
```

Dans le sous but courant, on se trouve dans le cas $x = L :: L :: x''$. On sait que l'hypothèse `Hrep : represents (L : :L : : x") u` n'a pu être obtenue qu'à partir du constructeur `reprL`. On aimerait bien pouvoir déduire les prémisses de ce constructeur, ce qui nous donnerait des informations plus

précises. Ceci est possible grâce à la tactique `inversion`. Dans le cas actuel, on obtiendra alors plus d'information sur $L :: x''$. On voudra donc utiliser cette tactique une deuxième fois pour obtenir encore plus d'informations sur x''. Nous avons défini une tactique `inversion_to_variable` qui exécute la tactique `inversion` autant que nécessaire .

```
inversion_to_variable x''

...
r : ℝ
H0 : represents (L :: x'') r
H3 : -1 ≤ r ≤ 1
H4 : (r-1)/2 = u
r0 : ℝ
H6 : represents x'' r0
H7 : -1 ≤ r0 ≤ 1
H8 : (r0-1)/2 = r
============================
represents (L :: sub_one) 2 × ((r0-1)/2 - 1)/2
```

On va utiliser un lemme déjà démontré qui dit que $L :: sub\_one = sub\_one$. On va aussi utilise la tactique `replace` qui nous permet de remplacer un terme par un autre, la preuve de l'égalité des deux termes étant demandée dans un deuxième sous but. Enfin si $r0$ vaut 1, le terme $2 \times \frac{\frac{r0-1}{2}-1}{2}$ se simplifie en -1. On exprime ceci en utilisant la tactique `replace` aussi en lui disant tout de suite que l'égalité engendrée dans le deuxième but se résout par la tactique `newfield`, qui sait résoudre des égalités que l'on peut montrer en utilisant les règles de réécriture d'un corps.

```
rewrite cons_L_sub_one.
replace r0 with 1.
replace 2 × ((r0-1)/2 - 1)/2 with -1.
2:newfield.

...
============================
represents (sub_one) -1
```

Ceci est exactement l'énoncé d'un lemme démontré après avoir défini `sub_one`. Il nous reste a démontrer l'égalité du remplacement de $r0$ par 1. Pour cela, on applique le théorème `Rle_antisym` qui exprime que la relation "$\leq$" définie sur les réels est une relation anti-symétrique. Ceci nous génère donc deux buts qui sont $-1 \leq r0$ et $r0 \leq -1$. Le premier fait partie des hypothèses et est donc trivial, le suivant découle du fait que x commence par $L :: L$. Comme il est un peu fastidieux on l'a démontré dans un lemme préalable `mult2_aux`.

```
apply Rle_antisym.
trivial.
apply mult2_aux.

...
Hrep : represents (L::R :: x'') u
============================
```

```
represents (L :: x'') (2u)
```

Le cas où x est de la forme $L :: L$ étant terminé, on est passé au but suivant qui est le cas $L :: R$. On commence tout d'abord par utiliser la tactique `inversion_to_variable`.

```
inversion_to_variable x''.

...
H0 : represents (R :: x'') r
H3 : -1 ≤ r ≤ 1
H4 : (r-1)/2 = u
r0 : ℝ
H6 : represents x'' r0
H7 : -1 ≤ r0 ≤ 1
H8 : (r0+1)/2 = r
============================
represents (L :: x'') 2 × ((r0+1)/2 - 1)/2
```

Si on simplifie le terme $2 \times \frac{\frac{r0+1}{2}-1}{2}$ on obtient $\frac{r0-1}{2}$. On a plus qu'à appliquer le constructeur `reprL` dont les prémisses sont les hypothèses données par la tactique `inversion_to_variable`.

```
replace 2 × ((r0+1)/2 - 1)/2 with (r0-1)/2.
apply reprL; assumption.
newfield.

...
============================
represents (L :: mult2 (L :: x'')) (2u)
```

Ce cas est terminé; on passe au cas $L :: R$ qui correspond à un appel co-récursif dans la fonction. On aura donc besoin de l'hypothèse de co-récursion. Encore une fois on utilise la tactique `inversion_to_variable` pour obtenir les informations liées au fait qu'on soit de la forme $L :: R$.

```
inversion_to_variable x''.

...
H0 : represents (C :: x'') r
H3 : -1 ≤ r ≤ 1
H4 : (r-1)/2 = u
H6 : represents x'' r0
H7 : -1 ≤ r0 ≤ 1
H8 : r0/2 = r
============================
represents (L :: mult2 (L :: x'')) 2 × (r0/2 - 1)/2
```

On peut simplifier $2 \times \frac{\frac{r0}{2}-1}{2}$ en $\frac{(r0-1)-1}{2}$, ce qui nous permet de pouvoir appliquer le constructeur `reprL`.

```
replace 2 × (r0/2 - 1)/2 with ((r0-1)-1)/2
2:newfield
```

```
apply reprL

...
============================
represents (mult2 (L :: x'')) (r0 − 1)
```

Avoir appliqué le constructeur signifie qu'on a réussi à montrer une étape de la preuve. On peut désormais utiliser l'hypothèse de co-récurrence. On réécrit d'abord $r0 - 1$ en $2 \times \frac{r0-1}{2}$ pour faire apparaître une multiplication par deux. Ainsi on peut appliquer notre hypothèse de récurrence.

```
replace (r0 − 1) en 2 × (r0−1)/2
2:newfield
apply mult2_correct

...
============================
−1/2 ≤ (r0−1)/2 ≤ 1/2

subgoal 2 is:
  represents (L :: x'') (r0−1)/2
```

Les deux sous buts restants de ce cas sont faciles et les quatres autre cas se résolvent de la même manière qu'un des cas exposés.

4.4 Ajout de la base en paramètre

Dans cette partie, nous avons adapté le travail pour une base quelconque. Les réels seront représentés par une séquence infinie d'entiers relatifs (`stream` $\mathbb{Z}$). Nous rappelons que pour simplifier certains algorithmes, nous avons fait le choix de ne considérer que des bases paires. Pour garantir que la base est paire, ce n'est pas la base qui est passée en paramètre des fonctions mais sa moitié. La transformation associée au chiffre k d'une base paire 2β est $x \mapsto \frac{k+x}{2\beta}$. Notre relation permettant de passer de notre représentation aux réels de Coq se charge désormais aussi d'assurer que tous les chiffres font partie de la même base.

```
CoInductive represents (b :ℤ): stream ℤ →ℝ →Prop :=
  | rep : ∀ s r k, represents b s r → −1 ≤ r ≤ 1 →
    −2b < k < 2b →represents b (k::s) (k+r)/2b.
```

Malheureusement dans cette nouvelle représentation, les algorithmes des travaux précédents ne sont plus réutilisables. En effet on ne peut plus raisonner par cas en regardant toutes les valeurs de chiffres possibles. On créé alors des cas en regroupant les chiffres par des propriétés comme "être pair", "être positif" ...

Illustrons le problème en détaillant le calcul de la demi-somme $(x, y) \mapsto \frac{x+y}{2}$ dans la base 2β. Il nous faut tout d'abord regarder le premier chiffre des arguments. Si x s'écrit $k_1 :: x'$ et y s'écrit $l_1 :: y$ alors la demi-somme peut se réécrire

$$\frac{\frac{k_1+x'}{2\beta} + \frac{l_1+y'}{2\beta}}{2} = \frac{\frac{k_1+l_1}{2} + \frac{x'+y'}{2}}{2\beta}$$

- Si $k_1 + l_1$ est pair, alors le prochain chiffre du résultat est $\frac{k_1+l_1}{2}$ et le reste du calcul est un appel co-récursif sur x' et y'.

- Sinon il existe un j tel que $2j+1 = k_1 + l_1$. On peut donc déduire que le résultat commence soit par j soit par $j+1$, mais l'encadrement actuel dont on dispose est trop grand pour décider lequel. Il faut donc regarder un chiffre de plus de $x = k_1 :: k_2 :: x''$.
  - Si $k_2 \leq -1$, comme k_2 est un chiffre de 2β, on a aussi $-2\beta+1 \leq k_2$. De plus comme x'' et y' sont des réels de $[-1, 1]$, on peut déduire

$$\begin{array}{rcl} \frac{\frac{-2\beta+1-1}{2\beta} - 1 + 1}{2} \leq & \frac{\frac{k_2+x''}{2\beta}+y'+1}{2} & \leq \frac{\frac{-1+1}{2\beta} + 1 + 1}{2} \\ \frac{-1}{2} \leq & \frac{((k_2+2\beta)::x'')+y'}{2} & \leq 1 \end{array}$$

    On peut aussi réécrire la demi-somme :

$$\begin{array}{rcl} \frac{k_1+\frac{k_2+x''}{2\beta}}{2\beta} + \frac{l_1+y'}{2\beta} & = & \frac{\frac{k_1+l_1-1}{2} + \frac{\frac{k_2+x''}{2\beta}+y'+1}{2}}{2\beta} \\ & = & \frac{\frac{2j+1-1}{2} + \frac{\frac{k_2+2\beta+x''}{2\beta}+y'}{2}}{2\beta} \\ & = & \frac{j + \frac{((k_2+2\beta)::x'')+y'}{2}}{2\beta} \end{array}$$

    Le premier chiffre est donc j. Et le reste du calcul se fait par un appel co-récursif sur $k_2 + 2\beta :: x''$ et y'.
  - Si $1 \leq k_2$ alors on applique le même raisonnement et on déduit que le premier chiffre est $j+1$.
  - Sinon $k_2 = 0$ et on ne peut toujours pas décider du prochain chiffre et il faut regarder le prochain chiffre de $y = l_1 :: l_2 :: y'$
    - Si $l_2 \leq -1$ par le même raisonnement sur les encadrements on trouve que le prochain chiffre est j
    - Sinon $0 \leq l_2$ et de manière similaire on obtient que le prochain chiffre est $j+1$.

```
CoFixpoint half_sum (b : ℤ) (x y : ℤ) : stream ℤ :=
  match (x, y) with (k1 :: x', l1 :: y') =>
    if (Zmod (k1 + l1) 2) = 0 (* si k1 + l1 est pair *)
      then (k1+l1)/2 ::half_sum b x' y'
      else match x' with k2 :: x'' =>
        if k2 ≤ -1
          then (k1+l1-1)/2 ::half_sum b (k2 + 2b :: x'') y'
          else if 1 ≤ k2
            then (k1+l1+1)/2 ::half_sum b (k2 - 2b) :: x'' y'
            else match y' with l2 :: y'' =>
              if l2 ≤ -1
                then (k1+l1-1)/2 ::half_sum b x' (l2 + 2b) :: y''
                else (k1+l1+1)/2 ::half_sum b (k2 - b) :: x'' (l2 - b) :: y''
             end
        end
  end.
```

Comme les algorithmes ne correspondent plus à ceux des travaux précédents, nous n'avons plus pu nous aider de leur preuves. De plus, au cours de nos

preuves, de nombreuses étapes demandent de démontrer des inégalités. Ces inégalités sont devenues plus complexes car la base y apparait comme une variable supplémentaire et non plus comme la constante 2.

Nous avons donc réussi à décrire la fonction `rat_to_stream` qui permet de calculer la représentation d'un rationel de $[-1, 1]$, l'opposé, l'addition, la soustraction, la multiplication.

A part la multiplication dont la preuve n'est pas tout à fait terminée, nous avons démontré formellement la correction de toutes les fonctions définies.

5 Conclusion

A l'issue de ce stage, nous disposons d'une petite bibliothèque de calculs qui permet d'exprimer des opérations de base sur les réels de $[-1, 1]$. Elle dispose de plus d'un moyen de calculer des séries entières à partir duquel on a pu définir le nombre d'Euler et π et calculer leurs décimales de façon exacte. La preuve de correction de ces calculs est vérifiée formellement par l'ordinateur.

Plusieurs directions prolongent naturelement ce travail.

- La formalisation d'une technique de calcul sur les séries formelles permettrait de définir de nouvelles fonctions comme la division.
- Une partie de la méthode calcul des séries entières est formalisée grâce à la fonction `series_body` qui se charge de trouver les chiffres de la somme et de lancer les appels récursifs dès lors qu'on a bien défini la fonction $f(x, y, n)$. Lorsque ensuite on veut démontrer la correction du calcul de la série, on doit démontrer la correction de la fonction $f(x, y, n)$ mais aussi que la fonction `series_body` conserve bien certains invariants de la preuve de $f(x, y, n)$. Une amélioration de cette méthode serait de formaliser aussi la preuve `series_body` en identifiant bien quels sont les invariants toujours nécessaires.
- Lorsque dans des calculs on manipule plusieurs fois le même nombre, ce dernier doit toujours être recalculé du début. Par exemple si on veut calculer $x + x$, l'algorithme va demander au premier x son premier chiffre ce qui peut mettre un certain temps suivant la définition de x puis au deuxième qui n'a aucun moyen de récupérer le résultat déjà calculé et va donc tout reprendre du début. De bonnes améliorations des algorithmes devrait être possibles en travaillant sur ce problème.
- L'utilisation combinée d'induction de de co-induction est nécessaire à la descriptions de certains algorithmes. Malheureusement cette approche n'est pas toujours possible en raison de la composition des leurs contraintes respectives. L'étude de méthodes permettant de faciliter le mélange de ces deux techniques dans la descriptions de fonction et l'élaboration de preuves serait un aspect essentiel.

Références

[1] Algirdas A. Avizienis. Signed-digit number representations for fast parallel arithmetic. *IRE Transactions on Electronic Computers*, 10 :389–400, 1961. URL : `http ://cr.yp.to/bib/entries.html#1961/avizienis`.

[2] Yves Bertot. Calcul de formules affines et de séries entières en arithmétique exacte avec types co-inductifs. In Thérèse Hardin et Luc Moreau, editor, *Journées francophones des langages applicatifs*. INRIA, Janvier 2006.

[3] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development, Coq'Art :the Calculus of Inductive Constructions*. Springer-Verlag, 2004.

[4] Alberto Ciaffaglione and Pietro Di Gianantonio. A coinductive approach to real numbers. In Th. Coquand, P. Dybjer, B. Nordström, and J. Smith, editors, *Types 1999 Workshop, Lökeberg, Sweden*, number 1956 in LNCS, pages 114–130. Springer-Verlag, 2000.

[5] Coq development team. *The Coq Proof Assistant Reference Manual, version 8.0*, 2004.

[6] Abbas Edalat and Reinhold Heckmann. Computing with real numbers. In Gilles Barthe, Peter Dybjer, Luis Pinto, and João Saraiva, editors, *APPSEM*, volume 2395 of *Lecture Notes in Computer Science*, pages 193–267. Springer, 2000.

[7] Eduardo Giménez. Codifying guarded definitions with recursive schemes. In Peter Dybjer, Bengt Nordström, and Jan Smith, editors, *Types for proofs and Programs*, volume 996 of *LNCS*, pages 39–59. Springer Verlag, 1994.

[8] Michael J. C. Gordon and Thomas F. Melham. *Introduction to HOL : a theorem proving environment for higher-order logic*. Cambridge University Press, 1993.

[9] Matt Kaufmann, Panagiotis Manolios, and J. Strother Moore. *Computer-aided reasoning : an approach*. Kluwer Academic Publishing, 2000.

[10] Micaela Mayero. *Formalisation et automatisation de preuves en analyses réelle et numérique*. PhD thesis, Université Paris VI, décembre 2001.

[11] Christine Paulin-Mohring. Inductive definitions in the system Coq : Rules and properties. In M. Bezem and J.F. Groote, editors, *Typed Lambda Calculi and Applications*, LNCS 664, pages 328–345. Springer, 1993.

[12] Natarajan Shankar, Sam Owre, and John M. Rushby. *The PVS Proof Checker : A Reference Manual*. Computer Science Laboratory, SRI International, Menlo Park, CA, February 1993. A new edition for PVS Version 2 is expected in 1998.