N° d'ordre : 292 N° attribué par la bibliothèque : 04ENSL0 292

École Normale Supérieure de Lyon
Laboratoire de l'Informatique du Parallélisme

THÈSE

présentée et soutenue publiquement le 22 novembre 2004 par

Sylvie Boldo

pour l'obtention du grade de

Docteur de l'École Normale Supérieure de Lyon
spécialité : Informatique

au titre de l'École doctorale de mathématiques et d'informatique fondamentale de Lyon

Preuves formelles en arithmétiques à virgule flottante

Directeur de thèse : Marc Daumas

Après avis de : Gilles Dowek
Paul Zimmermann

Devant la commission d'examen formée de :

Marc Daumas	Membre
Gilles Dowek	Membre/Rapporteur
William Kahan	Membre
Laurent Théry	Membre
Paul Zimmermann	Membre/Rapporteur

Remerciements

La vérité vaut bien qu'on passe quelques années sans la trouver.
Jules Renard

Merci tout d'abord à mon très cher directeur de thèse Marc Daumas pour m'avoir aidée, encadrée, soutenue, embêtée, motivée... Ce travail lui doit beaucoup, son intuition et ses connaissances bibliographiques ayant fait merveille. Ses directions ont été bonnes, même si ses évaluations de distance laissèrent parfois à désirer.

Merci aussi pour m'avoir proposé ce fantastique sujet de thèse qui me fait manipuler des bits but also make me play with Coq.

Merci ensuite à celui qui fut mon tuteur à mon arrivée à l'ÉNS, et probablement grâce auquel j'ai fait une thèse en arithmétique des ordinateurs, Jean-Michel Muller.

Merci aussi à Paul Zimmermann pour avoir encadré ma première rencontre avec la recherche et m'avoir rendu mon rapport de stage plus rouge que noir et d'avoir tout de même accepté de relire ce manuscrit.

Un grand merci à Laurent Théry pour son énorme aide en Coq, sa disponibilité, ses conseils et sa gentillesse.

Un grand merci au professeur Kahan, pour avoir relu ce manuscrit en français, être venu m'écouter de si loin et pour nos discussions passionnantes.

Un grand merci aux rapporteurs pour avoir relu ce tapuscrit jusqu'au bout de la dernière annexe et avoir amélioré mes connaissances en typographie.

Merci au jury dans son ensemble pour être venu m'écouter et avoir partagé ce moment avec moi.

Un merci spécial à mes cobureaux successifs pour m'avoir supportée, en particulier merci Stef et Guillaume !

Merci à tous ceux qui ont croisé mon chemin scientifique à Nancy, Sophia ou Lyon, ou par delà les mers : Guillaume H, Laurence, Loïc, Yves, Laurent F, Daniel, Philippe L, Nathalie, Arnaud, Florent, Claude-Pierre, Gilles, Nicolas W, Catherine, Jean-Luc, David D, Claire, Pascal, Jérémie, Saurabh plus tous ceux que j'oublie.

Pour la partie enseignement, merci à Pierre, Romain, Stéphane, Jean-François P, Véronique, Jean-François B et tous les autres.

Pour le non scientifique, j'ajoute un grand merci à Georges, Guillaume D, Guillaume D, Caroline, Jill, David T, Philippe G, Mahgally, ... et aussi à Joe, Lydia, Alexane et bien sûr mes parents (doublement).

Merci aux concepteurs et aux contributeurs des logiciels libres que j'ai utilisé (LaTeX, Debian, emacs, Coq ...)

Merci Nicolas.

Table des matières

Chapitre 1

Introduction(s)

It was the type of mistake that gives programmers and managers alike nightmares - and theoreticians and analysts endless challenge.
John R. Garman [Gar81]

Ce chapitre détaille les motivations de ce travail et présente les deux principaux thèmes, c'est-à-dire l'arithmétique des ordinateurs et la preuve formelle, avant de faire un état de l'art et de développer l'organisation de ce document.

C'est en pensant au type d'erreurs décrit par Garman dans [Gar81] que ce travail a été effectué. Dès que l'on s'attaque à des problèmes complexes, le risque d'erreur devient grand. En effet, les résultats deviennent plus difficiles à appréhender et à décrire, les démonstrations s'allongent et leur construction peut vaciller. C'est pour éviter cela que j'ai appliqué des méthodes formelles strictes à un domaine sensible : les calculs sur ordinateur. Le programmeur a en effet souvent tendance à considérer que les calculs que fait le processeur sont exacts, et ainsi à estimer que les algorithmes s'appliquent à des réels, alors qu'il n'en est rien. Le manque ou le flou des spécifications est également un problème récurrent. Comme le dit Davis [Dav72], « The program itself is the only complete description of what the program will do. ».

Parmi les « bugs » les plus retentissants se trouvent le report du décollage de la première navette spatiale américaine [Gar81], l'explosion du missile Patriot [Uni92], les erreurs dans la division du Pentium [Mul95, Coe95], un problème de drapeau dans le Pentium II [Col97], l'explosion du premier vol d'Ariane 5 [L$^+$96], un problème sur une plate-forme pétrolière [CVSG97], l'arrêt de l'USS Yorktown [Hay98] et un autre « bug » dans le Pentium Pro [OZGS99].

Notre héritage des grecs, dont Socrate et Platon, contient la notion de preuve : partir d'axiomes évidents et appliquer un raisonnement, c'est-à-dire un enchaînement d'énoncés et d'implications indiscutables pour obtenir une conclusion désormais considérée comme exacte. Les preuves mathématiques actuelles n'ont plus grand-chose à voir avec ce scénario idyllique, surtout à cause de la complexité croissante des preuves et énoncés. Comment alors garantir la véracité d'un énoncé ? On trouve dans la littérature quelques réponses originales : dans

[DLP77], la vérité d'un théorème n'est pas un fait donné, mais une qualité qui s'acquiert par un long processus social. Dans [Dav72], la correction d'une preuve n'est qu'une variable aléatoire. Que cette preuve soit faite à la main ou par ordinateur, seule varie la probabilité d'erreur bien plus faible dans le second cas. La façon dont sont conçues nos preuves papier pourrait être améliorée pour en faire une preuve structurée, c'est-à-dire organisée par niveau, qui serait bien plus rigoureuse [Lam95]. On pourrait également envisager une preuve à mi-chemin entre preuve informelle et preuve formelle [Wie03b].

Ce problème est réel : il arrive même que certains articles qui ont été relus, vérifiés, appliqués et maintes fois cités tels que [LMS85] de Lamport et Melliar-Smith contiennent plusieurs erreurs. Or ces erreurs n'ont été découvertes que lorsque ce résultat a été démontré à l'aide d'un assistant de preuve [RvH91].

Le travail présenté ici est à la croisée entre deux sujets de recherche, l'arithmétique des ordinateurs et les preuves formelles. Je vais maintenant introduire chacun de ces sujets, avant de faire un état de l'art du domaine et de présenter le plan de ce document.

1.1 Arithmétique des ordinateurs ?

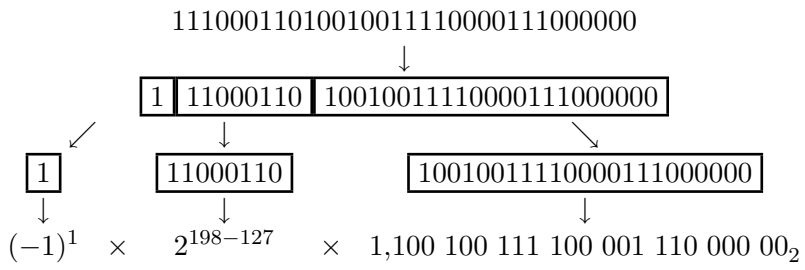
L'arithmétique des ordinateurs est l'étude de la façon dont les ordinateurs calculent. En effet, leur mémoire étant finie, le nombre de valeurs représentables l'est également et on ne peut donc pas représenter exactement tous les résultats possibles de toutes les opérations. Je m'intéresse ici aux valeurs réelles représentables. La façon de représenter ces « quelques » valeurs est ordinairement celle décrite par la norme IEEE-754 [S$^+$81, S$^+$87] de 1985 sous le nom nombre à virgule flottante ou nombre flottant. On considère un format flottant et une suite de bits sur ce format à laquelle on donne un sens selon des tailles données pour s (signe), e (exposant) et f (fraction) comme dans la figure 1.1.

FIG. 1.1 – Nombre à virgule flottante.

Sauf cas particuliers, sa valeur est alors le résultat du calcul :

$$(-1)^s \times 2^{e-B} \times 1 \bullet f$$

où f est la fraction composée des bits $b_1 b_2 \dots b_{p-1}$ et où $1 \bullet b_1 b_2 \dots b_{p-1} = 1 + \sum_{i=1}^{p-1} b_i 2^{-i}$ et où B est une valeur connue dépendant du format flottant appelée le biais. Ainsi par exemple la simple précision IEEE-754 est composée de 32 bits et le nombre flottant suivant a pour valeur :



$$= -2^{54} \times 206727$$
$$\approx -3{,}724 \times 10^{21}.$$

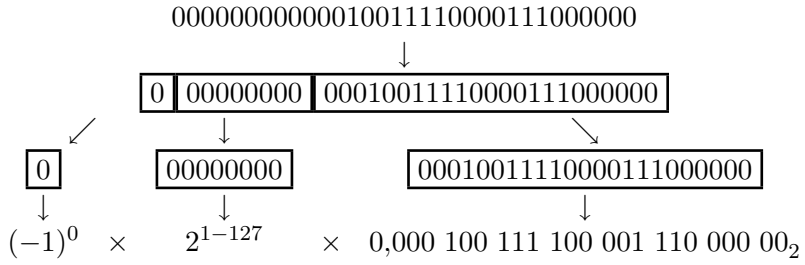
Cet algorithme vaut pour tous les nombres flottants sauf ceux qui ont des valeurs spéciales de l'exposant : la valeur minimale et la valeur maximale. Pour la valeur minimale (nulle) de l'exposant, les nombres flottants correspondant sont appelés dénormalisés et représentent des nombres plus petits que les plus petits représentables par la méthode précédente.

La valeur d'un nombre flottant dénormalisé est alors le résultat du calcul :

$$(-1)^s \times 2^{1-B} \times 0 \bullet f$$

où $0 \bullet b_1 b_2 \dots b_{p-1} = \sum_{i=1}^{p-1} b_i 2^{-i}$.

Ainsi par exemple en simple précision IEEE-754, le nombre flottant suivant a pour valeur :



$$= 2^{-143} \times 10119$$
$$\approx 9{,}075 \times 10^{-40}.$$

La valeur minimale de l'exposant permet également de représenter les deux valeurs ± 0, tandis que la valeur maximale permet de représenter $\pm\infty$ et les *NaN* (voir le chapitre 5). Pour plus d'informations sur la norme et ses conséquences, se référer par exemple à [Gol91, Ove95, Sun96, DM97b, MF01] ou les premiers chapitres de nombreux livres d'analyse numérique tels que [Hig02].

Il est habituel de schématiser les nombres à virgule flottante par des rectangles représentant leurs mantisses. L'écart entre les rectangles est alors la différence des exposants des différents nombres à virgule flottante. Par exemple, les nombres à virgule flottante en base 10 sur 6 chiffres suivants sont représentés sur la figure 1.2.

$$\begin{array}{rcrcrcrcr} x & = & 2{,}713\ 67 & \times & 10^8 & = & 271367 & \times & 10^3 \\ y & = & 7{,}230\ 03 & \times & 10^{10} & = & 723003 & \times & 10^5 \end{array}$$

FIG. 1.2 – Représentations en rectangles de deux nombres à virgule flottante.

Les flèches en bout de rectangles désignent ici l'exposant du nombre flottant pour une mantisse entière. Une flèche désigne l'exposant du chiffre juste à sa gauche.

1.2 Preuves formelles ?

Je m'intéresserai ici uniquement aux preuves vérifiées par ordinateur, par un programme appelé assistant de preuves. Plus précisément, pour un théorème donné, je dois le traduire dans le langage de l'assistant de preuves et ensuite détailler la preuve dans le langage choisi par l'intermédiaire de tactiques (application de théorème, réécriture, simplification...) afin de faire comprendre et accepter à l'assistant toutes les étapes du raisonnement.

De nombreux assistants de preuves sont disponibles, ils sont souvent inspirés de l'isomorphisme de Curry-Howard [CF58, How80] : chaque preuve acceptée fournit un λ-terme [Bar84] dont le type est exactement le théorème. Créer ce terme correspond à construire la preuve et vérifier la preuve correspond à typer ce terme, ce qui est une opération simple exécutable par un programme de petite taille. L'assistant de preuves a pour fonction de construire ce λ-terme à partir de la preuve fournie par l'utilisateur, qui peut ensuite être typé soit par l'assistant de preuves, soit par un autre programme. Il est ainsi possible de nettement diminuer les risques d'erreur en « contre-vérifiant » un λ-terme avec deux programmes distincts.
Parmi les assistants de preuve les plus connus se trouvent notamment :

- ACL2 [KMM00a, KMM00b] : voir `http://www.cs.utexas.edu/users/moore/acl2/`,
- Coq [Coq04, BC04] : voir `http://coq.inria.fr/`,
- HOL Light [Har96] : voir `http://www.cl.cam.ac.uk/users/jrh/hol-light/`,
- Isabelle [NPW02] : voir `http://www.cl.cam.ac.uk/Research/HVG/Isabelle/`,
- LEGO [LP92] : voir `http://www.dcs.ed.ac.uk/home/lego/`,
- PVS [ORS92] : voir `http://pvs.csl.sri.com/`.

Notre développement est actuellement intégralement fait en Coq, qui est basé sur le calcul des constructions inductives [Dow02]. Pour un comparatif des assistants de preuve cités et d'autres, voir [Wie03a].

Un des problèmes des développements formels est celui du chapeau mexicain présenté par H. Geuvers à la réunion MAP de Luminy en 2004 et repris en figure 1.3. Le travail consiste à formaliser un domaine avec des définitions et un assez grand nombre de lemmes basiques et ensuite, de prouver un théorème difficile. Pour combler le fossé entre les lemmes basiques et le sommet à atteindre, une énorme quantité de lemmes intermédiaires non réutilisables est démontrée. L'inconvénient est que la formalisation obtenue est peu utilisable par d'autres, car le travail à faire depuis les lemmes basiques est toujours important.

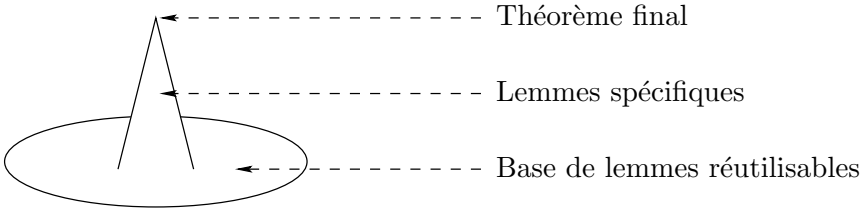


Fig. 1.3 – Chapeau mexicain.

J'ai donc voulu obtenir le chapeau mexicain amélioré de la figure 1.4, moins pratique pour se protéger du soleil, mais plus réutilisable par d'autres. Le but est de faire beaucoup plus de lemmes de base, de façon à rapprocher les théorèmes finaux de ce qui existe déjà. Pour justifier cette approche, il faut faire plusieurs théorèmes finaux difficiles et montrer que le travail à faire pour les obtenir à partir de ce qui existe n'est pas outrancier et est faisable

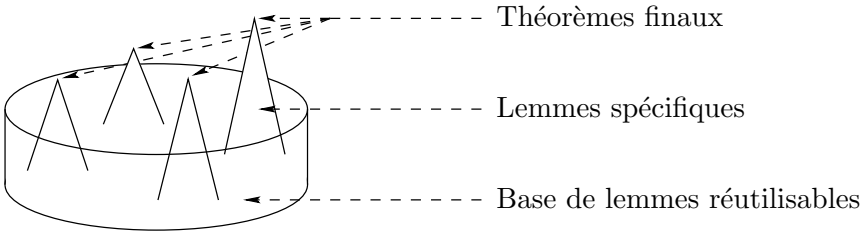


FIG. 1.4 – Chapeau mexicain amélioré.

par d'autres que le(s) concepteur(s) de la bibliothèque initiale. Le fait qu'il y ait plusieurs théorèmes finaux est également une garantie de salubrité de la formalisation : elle peut être appliquée à différents types de problèmes ce qui limite le risque d'erreur dans les définitions.

1.3 État de l'art

Avec le développement des outils formels, il était évident qu'un domaine sensible comme les calculs flottants et les processeurs serait formalisé et donné en pâture aux assistants de preuves. Voir notamment [KG99] pour une vue d'ensemble des techniques formelles appliquées aux architectures matérielles (pas seulement concernant l'arithmétique flottante).

La figure 1.5 récapitule les principaux travaux dans le domaine et leur(s) institution(s), la date correspond à la date de la première publication. Pour chacun, je présente quelques compléments et références :

- G. Barrett [Bar89] : ce n'est qu'une spécification en Z de très bas niveau de la norme IEEE-754, avec notamment les valeurs en hexadécimal de certains champs pour les valeurs spéciales de la norme. Il n'y a aucune preuve, donc aucun test de la validité du modèle, mais c'est la première formalisation et l'effort est important et extrêmement louable.
- P. Miner et V. Carreño [Car95, Min95, CM95] : les travaux de P. Miner en PVS et de V. Carreño en HOL à la NASA sont des spécifications presque complètes de la norme IEEE-854. Les définitions valent donc pour les bases 2 et 10 et les définitions sont plus génériques que celle de G. Barrett. Néanmoins, de l'aveu même des auteurs, peu de

Auteur(s)	**Langage(s)**	**Institution(s)**	**Date**
G. Barrett	Z	Oxford University	1989
P. Miner V. Carreño	PVS HOL	NASA	1995
J. Harrison	HOL Light	Cambridge Intel	1997
D. Russinoff	ACL2	AMD	1998
C. Jacobi	PVS	Saarland University, Saarbruck	2001
L. Théry, S. Boldo *et al.*	Coq	INRIA/CNRS/ENS Lyon	2001

FIG. 1.5 – Récapitulatif de l'état de l'art.

démonstrations ont été faites ce qui ne permet pas d'avoir une grande confiance dans ces spécifications, même si [ML96] a été publié depuis.

- D. Russinoff [Rus98, Rus99, Rus00] : ses travaux en ACL2 sont des vérifications formelles des unités flottantes des AMD-K5 et AMD-K7. La seule base envisagée est donc 2 et la spécification de la norme IEEE-754 est incomplète, mais les démonstrations concernées sont très impressionnantes. Sa formalisation a d'ailleurs été réutilisée par d'autres dans [MLK98].
- C. Jacobi [Jac01, Jac02] : ses travaux en PVS sont composés d'une part d'une spécification très matérielle de la norme IEEE-754, mais également des preuves utilisant cette formalisation garantissant la correction d'unités flottantes réelles.
- J. Harrison [Har97a, Har97b, Har99, Har00] : ses travaux en HOL Light (outil qu'il a lui-même écrit) sont assez proches des nôtres. Il ne considère pas les bits sous-jacents mais directement la valeur entière de la mantisse, tout en conservant les valeurs exceptionnelles. Les démonstrations faites concernent l'évaluation de fonctions élémentaires notamment l'exponentielle et ont nécessité de nombreuses preuves mathématiques sur l'analyse réelle.
- Pour les caractéristiques de nos propres travaux en Coq, voir le chapitre 2.

Ces travaux sont donc assez différents car ils répondent tous à une attente extrêmement précise soit de spécification, soit de preuve(s) donnée(s). Je présente en profondeur au chapitre suivant la formalisation utilisée. Ainsi, je pourrai résumer quelques caractéristiques comparables de ces formalisations dans le tableau de la figure 2.5 page 20.

1.4 Plan

Cette thèse est composée de 11 chapitres organisés comme suit :

- deux chapitres d'introductions : ce chapitre très général, ainsi que le suivant décrivant précisément la formalisation utilisée ;
- une partie détaillant des résultats simples déduits de ce modèle, ce qui vise aussi à le valider devant des cas réels :
  - un chapitre décrivant comment et quand obtenir le terme d'erreur d'un calcul ;
  - un chapitre décrivant certains cas où je peux prévoir qu'un calcul est exact ;
- une partie détaillant quelques extensions et enrichissements du modèle de façon à tester sa souplesse et son adaptabilité :
  - un chapitre décrivant un enrichissement du modèle avec une représentation des nombres flottants du type IEEE (en vecteurs de bits) ;
  - un chapitre décrivant une modification du modèle visant à représenter d'autres systèmes (dont le complément à la base) ;
  - un chapitre décrivant un autre « arrondi » plus facile à obtenir et ses caractéristiques ;
- une partie détaillant l'application à deux bibliothèques :
  - une bibliothèque de calculs sur les expansions ;
  - des résultats sur l'évaluation de fonctions élémentaires :
    - un chapitre décrivant des résultats sur l'évaluation polynomiale par la méthode de Horner ;

  - un chapitre décrivant une technique de réduction d'arguments ;
- un chapitre de conclusions et perspectives.

Il faut noter que cet ordre vise uniquement à la clarté de l'exposé. Ce n'est en effet pas l'ordre chronologique de réalisation des travaux. Ce n'est pas non plus un ordre respectant les dépendances entre les résultats.

Plus précisément, certains des résultats acquis dépendent de résultats précédents selon le graphe de dépendance des fichiers Coq dont une version simplifiée est en figure 1.6. Cette figure se lit ainsi : une flèche de a vers b signifie que les résultats de b dépendent de ceux de a. Une version complète des graphes de dépendances se trouve en annexe A.

Bien sûr, tous les résultats se basent sur la formalisation des nombres flottants décrite dans le chapitre 2. Le chapitre 8 sur la bibliothèque de calculs sur les expansions se base sur la représentation exacte des erreurs de calculs dont les résultats sont présentés au chapitre 3. Le chapitre 6 sur les autres systèmes dont le complément à 2 permet d'obtenir de meilleurs résultats au chapitre 4 où je prévois que certains calculs sont exacts. Ces prévisions permettent d'obtenir des résultats sur la réduction d'arguments du chapitre 10. Les résultats du chapitre 7 sur l'arrondi fidèle, un autre « arrondi » plus faible mais plus facile à obtenir permettent d'énoncer et de démontrer les résultats du chapitre 9 sur l'évaluation polynomiale par la méthode de Horner. Le chapitre 5 visant à rapprocher notre formalisation des vecteurs de bits de la norme IEEE-754 n'est pour l'instant pas utilisé par d'autres développements.

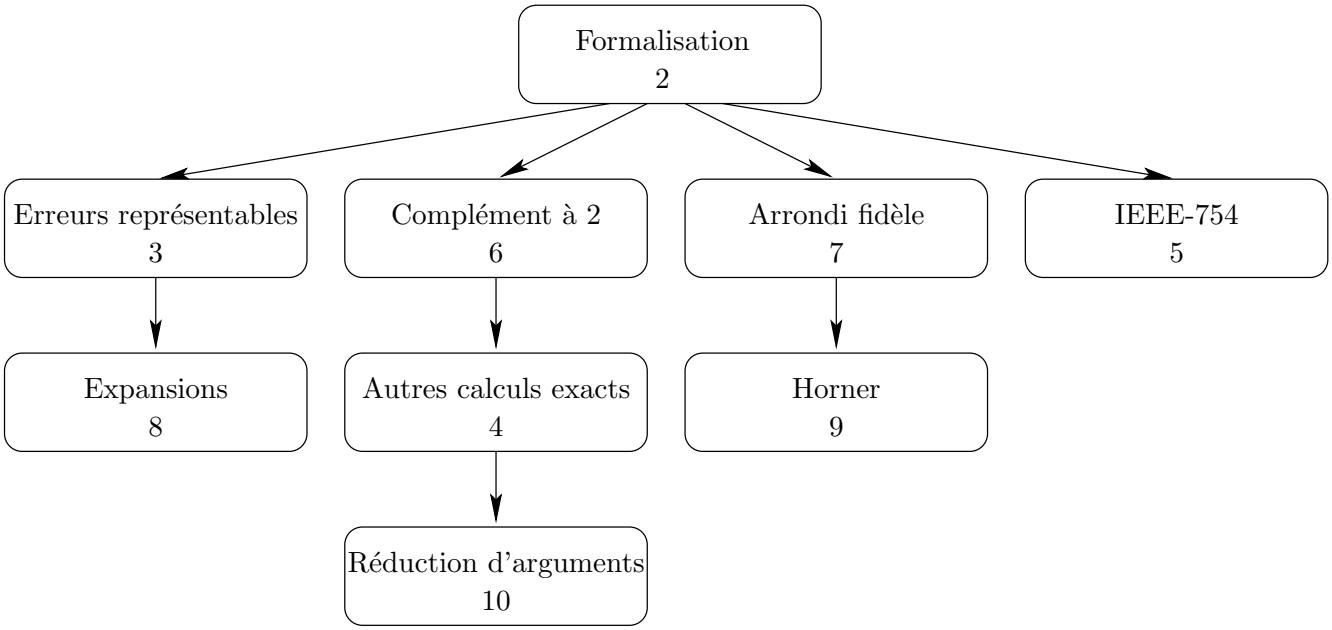


FIG. 1.6 – Dépendances des chapitres.

Chapitre 2

Formalisation des nombres à virgule flottante

Our hero is the intrepid, yet sensitive matrix A.
Our villain is E, who keeps perturbing A.
When A is perturbed he puts on a crumbled hat: $\widetilde{A} = A + E$.
G. W. Stewart et J.-G. Sun, Matrix Perturbation Theory (1990)

Ce chapitre détaille la formalisation Coq des nombres flottants qui est utilisée dans ce document. Sa lecture n'est pas nécessaire à la compréhension des résultats des chapitres suivants, mais éclaire sur les difficultés rencontrées ainsi que les méthodes de démonstration choisies, parfois contraires à l'intuition.

Dans ce chapitre et tout au long de cette thèse, les énoncés Coq entiers ou partiels sont écrits sur fond gris : mon énoncé et en syntaxe de la version 8.0. Pour faire le lien entre un énoncé mathématique et l'énoncé Coq correspondant, j'utilise la flèche ➢, ce qui donne par exemple : « la base β ➢ radix est un entier... ». Les énoncés Coq plus longs (définition, théorème) sont également sur fond gris, mais de plus en fonte `verbatim` et précédés de la mention ÉNONCÉ COQ. Seuls les résultats les plus utiles et originaux sont appelés théorèmes et sont accompagnés de leur contrepartie en Coq. Ils sont listés en page 121. Néanmoins, tous les résultats (lemmes et théorèmes) présentés ici sont certifiés en Coq, c'est pourquoi le nom Coq ainsi que le fichier le contenant sont donnés pour chaque résultat.

Les nombres flottants binaires sont définis par la norme américaine ANSI/IEEE-754 [S$^+$81, S$^+$87] de 1981, une norme internationale équivalente ISO/IEC bilingue français/anglais a été adoptée en 1989 [IEC89]. Seule la norme IEEE-854 [CK$^+$84] prend en compte d'autres bases, en particulier la base 10. Un comité de l'IEEE se réunit depuis 2001 et vise à l'actualisation et à la révision de la norme IEEE-754 en une version que j'appelle 754R [IEE04]. Les révisions n'étant pas terminées au moment de la rédaction de cette thèse, seules les décisions estimées définitives par ce comité sont envisagées.

Ces normes sont longues, difficiles à lire et mélangent à la fois définitions et propriétés. Parmi les définitions se trouvent naturellement les formats des nombres flottants et ce que signifient les arrondis. Mais elles définissent également et implicitement des propriétés en définissant des opérations : ainsi l'opération de reste euclidien FPREM entre x et y est définie comme suit :

$$\begin{array}{rcll} n & = & \left\lceil \frac{x}{y} \right\rfloor & \text{est l'entier le plus proche de } \frac{x}{y} \\ x \text{ FPREM } y & = & x - n \times y & \text{est représentable exactement.} \end{array}$$

La définition de FPREM contient implicitement une propriété : celle que le résultat est exactement représentable dans le format flottant utilisé (celui de x et de y). Il n'y a pourtant aucune preuve de ce fait dans aucune des normes : cela appartient à une culture commune aux gens du domaine. Cela pose un problème de fond sur la validité de ces références, mais que faut-il faire ?

- Ajouter des preuves à la norme ne ferait que la rallonger sans la clarifier. Il est toujours possible de faire référence à un article contenant une preuve mais, comme vu précédemment, comment faire confiance à une preuve papier ?
- Faire référence à une preuve formelle serait une solution (l'ajouter à la norme me semble sans plus d'intérêt que pour une preuve papier et bien plus long). Le problème se pose alors également du choix de l'assistant de preuves et de la formalisation choisie. Un autre problème est alors d'être sûr que la preuve formelle correspond bien à la propriété voulue.

C'est pourquoi notre formalisation privilégie la lisibilité : les définitions sont peu nombreuses et compréhensibles. De plus, si l'on considère l'approche par niveau de la norme révisée 754R [IEE04, §3], nous nous plaçons au niveau 3 : nous ignorons la représentation binaire sous-jacente pour ne considérer que la valeur des différents champs (mantisse et exposant).

2.1 Un peu d'histoire

L'idée de spécifier formellement l'arithmétique des ordinateurs et de prouver formellement des algorithmes ou des processeurs n'est pas neuve (voir l'état de l'art en section 1.3). Néanmoins aucune spécification ne semblait satisfaisante en terme de généricité, c'est pourquoi diverses équipes françaises se sont réunies dans une Action de Recherche Coopérative de l'INRIA appelée AOC (Arithmétique des Ordinateurs Certifiée) en 2000 et 2001.

Elle réunissait les projets Arénaire (Lyon), Lemme (Sophia-Antipolis) et PolKA (désormais SPACES, Nancy). Arénaire et PolKA apportaient leurs connaissances en arithmétique des ordinateurs et leurs applications tandis que Lemme apportait ses connaissances et son expertise en méthodes formelles. La spécification a alors été conçue par L. Théry, L. Rideau et M. Daumas. Arrivée peu après, je n'ai donc pas influé sur la spécification, mais je l'ai intensivement utilisée et j'ai également contribué à l'enrichissement de la bibliothèque de base. J'ai ajouté les lemmes qui me semblaient nécessaires à une telle bibliothèque au vu de mon expertise sur les nombres flottants. Ma première application concernait les travaux de C. Moreau-Finot [MF01] sur les expansions (voir chapitre 8).

Les travaux décrits dans ce chapitre ne sont donc en aucun cas miens, ils appartiennent essentiellement à L. Théry. Ceci explique pourquoi ce chapitre a été mis en exergue en seconde introduction avant le corps du document contenant mes propres travaux (sauf mention

contraire). Mon travail consiste ici en une nouvelle explication de ces bases avec plus de pédagogie, car plus de longueur, et également de nouvelles notations qui sont je l'espère plus simples et plus claires.

2.2 Nombres à virgule flottante

La plupart des processeurs actuels se basent sur la norme IEEE-754 pour leurs calculs flottants, les nombres flottants sont donc représentés en signe-valeur absolue et en base 2. La représentation signe-valeur absolue est concurrencée dans les processeurs dédiés par la notation en complément à la base, que je décris et étudie dans le chapitre 6 page 57.

Quant au choix de la base, la base 2 s'est quasiment imposée partout. La base 10 est utilisée par quelques ordinateurs IBM [CSSW01] et reste défendue [IEE04], notamment par les banques, car elle permet de calculer directement avec les valeurs effectivement affichées (et lues par un humain), ce qui évite les erreurs de conversion tout en restant efficace [Cow03]. Quelques systèmes conservent une compatibilité historique avec la base 16 [SSK99] mais d'autres bases restent envisageables [Avi61, Cod73].

Dans l'esprit de la norme IEEE-854, et sans se baser spécifiquement sur la norme IEEE-754, nos nombres flottants sont dans une base quelconque. La base β ➢ radix est un entier relatif ➢ Z quelconque tel que $1 < \beta$ ➢ hypothèse radixMoreThanOne, où le 1 est l'entier relatif de valeur 1 et l'inégalité est sur les entiers relatifs ➢ %Z. Dans certains développements, j'impose $\beta = 2$, mais par défaut la base est laissée libre ➢ Variable et est un paramètre de chaque fichier.

Énoncé Coq
```
Variable radix : Z.
Hypothesis radixMoreThanOne : (1 < radix)%Z.
```

Un nombre flottant est un enregistrement ➢ Record composé de deux entiers relatifs ➢ Z, le premier étant appelé la mantisse ➢ Fnum et le second l'exposant ➢ Fexp. Sa valeur numérique, obtenue par la fonction FtoR, est le produit signé dans $\mathbb{R}$ ➢ %R de sa mantisse par la base élevée à la puissance de son exposant.

Énoncé Coq
```
Record float : Set := Float {
   Fnum: Z;
   Fexp: Z }.

Definition FtoR (x : float) :=
  (Fnum x * (powerRZ (IZR radix) (Fexp x)))%R.
```

On note tout nombre flottant comme une paire (n, e) à laquelle est associée la valeur réelle

$$n \times \beta^e.$$

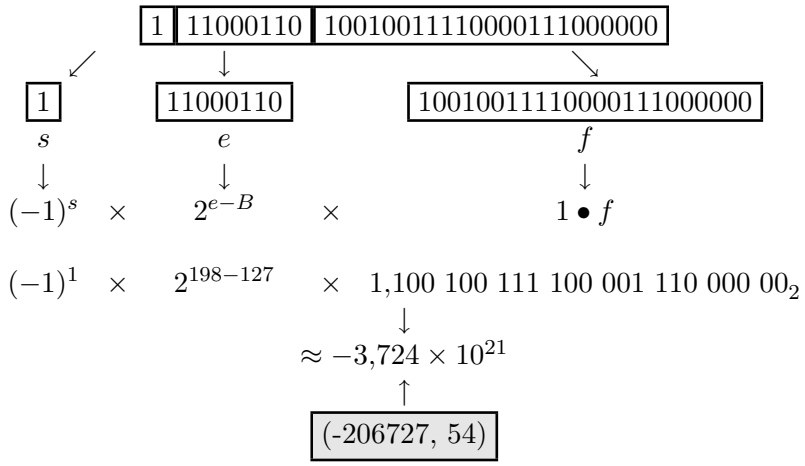


FIG. 2.1 – Schéma d'équivalence entre nombres flottants machine et nombres flottants Coq.

Ainsi, comme l'explique la figure 2.1, il existe une équivalence de valeur entre les nombres flottants machines et les nombres flottants Coq.

La fonction FtoR a deux arguments, d'une part la base ➢ radix (implicite dans la définition ci-dessus) et d'autre part un nombre flottant à évaluer. La définition, locale à un fichier de la fonction FtoRradix permet d'utiliser une fonction à un seul argument de type float.

Comme dans les normes, les nombres flottants (*i.e.* paires d'entiers) sont automatiquement convertis ➢ Coercion en leur valeur réelle au besoin, c'est-à-dire lorsqu'ils sont utilisés comme des réels et ceci par la fonction FtoRradix.

ÉNONCÉ COQ
```
Let FtoRradix := FtoR radix.
Coercion FtoRradix : float >-> R.
```

Cela signifie que l'on a deux égalités sur les nombres flottants. D'une part, l'égalité de Leibniz, notée $=$ correspondant à l'égalité des champs composant la paire :

$$(n, e) = (n', e') \quad \Leftrightarrow \quad n = n' \text{ et } e = e'.$$

En pratique, on utilise surtout l'autre égalité, notée $=_{\mathbb{R}}$ qui correspond à l'égalité des valeurs réelles associées :

$$(n, e) =_{\mathbb{R}} (n', e') \quad \Leftrightarrow \quad n \times \beta^e = n' \times \beta^{e'}.$$

Cette égalité devrait dépendre de la base, mais on travaille toujours ici à base constante et ce paramètre est implicite.

La limite de cette définition des nombres flottants est qu'elle contient beaucoup trop de nombres flottants par rapport à la réalité : il n'y a aucune limite aux valeurs possibles des mantisses et des exposants. L'ensemble obtenu est $\mathbb{Z}\ \beta^{\mathbb{Z}}$, qui est infini, dénombrable et même dense dans $\mathbb{R}$!

2.3 Formats flottants et nombres flottants représentables

Pour remédier à ce problème, on limite ces paires pour ne garder que celles qui tiennent dans un format flottant donné. Plus précisément, on définit un format flottant comme un enregistrement composé de deux valeurs : N ➢ vNum qui est un entier strictement positif ➢ positive et E_i ➢ dExp qui est un entier positif ➢ nat.

Un nombre flottant (n, e) est dit « représentable » ➢ Fbounded si et seulement si

$$|n| < N \quad \text{et} \quad -E_i \leq e.$$

Énoncé Coq
```
Record Fbound : Set := Bound {
   vNum: positive;
   dExp: nat}.

Definition Fbounded (b : Fbound) (d : float) :=
  (Zabs (Fnum d) <  Zpos (vNum b))%Z
    /\ (- dExp b <= Fexp d)%Z.
```

En pratique, la valeur de N est une puissance de la base, c'est-à-dire $N = \beta^p$ ➢ hypothèse pGivesBound. Cela correspond à l'intuition que la mantisse est représentée sur p ➢ precision chiffres en base β et j'adopte cette convention dans toute la suite et je n'utilise plus que p et la valeur β^p plutôt que N.

Énoncé Coq
```
Hypothesis pGivesBound :
   Zpos (vNum b) = Zpower_nat radix precision.
```

Je supposerai également que $p \geq 2$ ➢ hypothèse precisionGreaterThanOne.

Énoncé Coq
```
Hypothesis precisionGreaterThanOne : 1 < precision.
```

Il faut noter que cette formalisation contient les normalisés et les dénormalisés, mais ne contient qu'un seul zéro et aucune valeur spéciale (NaN et $\pm\infty$). Voir le chapitre 5 pour plus de précisions.

2.4 Arrondis

2.4.1 Arrondi générique

Un arrondi est un moyen de passer de l'ensemble infini continu des réels à l'ensemble fini discret des nombres flottants représentables comme le montre la figure 2.2 : les nombres

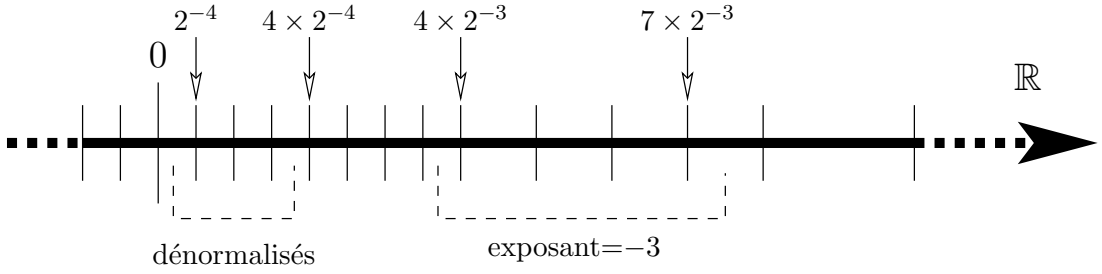


FIG. 2.2 – Ensemble discret des nombres flottants avec $p = 3$ et $-E_i = -4$.

flottants sont liés aux barres verticales placées sur la flèche horizontale qui correspond aux nombres réels. Les normes IEEE-754, IEEE-854 et ISO/IEC 60559 définissent toutes quatre modes d'arrondi : vers $+\infty$, vers $-\infty$, vers 0 ou au plus proche pair, comme le montre l'exemple de la figure 2.3. Ce dernier arrondi est celui utilisé par défaut et renvoie toujours le nombre flottant le plus proche du réel considéré ; dans le cas où deux nombres flottants sont à égale distance du réel, on renvoie celui dont la mantisse est paire, *i.e.* finit par un zéro en base 2. La révision 754R ajoute l'arrondi au plus proche « extérieur » (voir ci-dessous). Toutes ces normes sont suffisamment spécifiées pour qu'à chaque réel, un et un seul nombre flottant soit associé de façon déterministe.

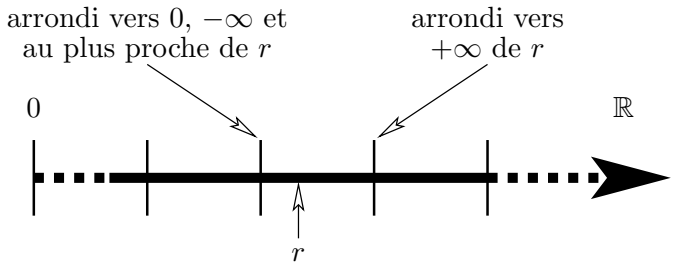


FIG. 2.3 – Exemple d'arrondis IEEE-754 possibles d'un réel r.

Notre vision est plus large : nous souhaitons entre autres pouvoir énoncer des propriétés valables pour tout arrondi. Une solution aurait été de quantifier l'arrondi sur un ensemble fini à 4 éléments. Mais alors, l'ajout d'un nouvel arrondi impliquerait de redémontrer la plupart des lemmes pour ce nouvel arrondi. Cela semble absurde alors que la notion de mode d'arrondi générique est relativement intuitive car les propriétés à remplir sont simples. Pour certains, c'est une projection croissante des réels vers les nombres flottants. Nous avons choisi ici une définition équivalente, ou presque.

ÉNONCÉ COQ
```
Definition RoundedModeP (P : R -> float -> Prop) :=
  TotalP P /\ CompatibleP P /\ MinOrMaxP P /\ MonotoneP radix P.
```

La première différence et la plus frappante est que l'arrondi a jusqu'à maintenant toujours été une fonction. Or, dans la formalisation, l'arrondi n'est plus une fonction mais devient

une relation ➢ P : R → float → Prop. Ce n'est plus une projection de $\mathbb{R}$ vers l'ensemble des nombres flottants, mais une relation ➢ Prop liant un réel ➢ R et un nombre flottant ➢ float (ainsi implicitement que la base et le format utilisé pour le résultat). Cela s'explique facilement en considérant un arrondi au plus proche générique où le choix n'est pas tranché lorsque le réel est à mi-distance entre deux nombres flottants : on a alors deux nombres flottants acceptables, tous deux arrondis corrects du réel. Une fonction ne peut refléter cette dualité alors qu'une relation le peut : le réel est en relation avec deux nombres flottants distincts sans que cela ne pose de problème.

Pour expliciter la définition qui précède d'un arrondi, je dois d'abord définir les projections vers $\pm\infty$. La projection vers $-\infty$ ➢ isMin de r est un nombre flottant représentable g tel que $g \leq r$ et pour tout nombre flottant représentable f, on a $f \leq r \Rightarrow f \leq g$.

La projection vers $+\infty$ ➢ isMax de r est un flottant représentable g tel que $g \geq r$ et pour tout nombre flottant représentable f, on a $f \geq r \Rightarrow f \geq g$.

Ces deux définitions dépendent du format flottant utilisé ➢ b.

Énoncé Coq

```
Definition isMin (r : R) (min : float) :=
  Fbounded b min /\
  (min <= r)%R /\
  (forall f : float, Fbounded b f -> (f <= r)%R -> (f <= min)%R).

Definition isMax (r : R) (max : float) :=
  Fbounded b max /\
  (r <= max)%R /\
  (forall f : float, Fbounded b f -> (r <= f)%R -> (max <= f)%R).
```

Revenons maintenant aux propriétés nécessaires à notre arrondi générique : pour pouvoir être qualifiée d'arrondi ➢ RoundedModeP, une relation doit satisfaire chacune des quatre propriétés :

Total Tout réel a un arrondi.

Compatible Soit un nombre flottant f_1 qui est un arrondi d'un réel r_1. Soit un nombre flottant f_2 représentable tel que $f_2 =_{\mathbb{R}} f_1$. Soit un réel r_2 tel que $r_1 = r_2$. Alors f_2 est un arrondi de r_2.

MinOrMax Un arrondi d'un réel ne peut être que l'un des nombres flottants encadrant ce réel, c'est-à-dire soit la projection vers $+\infty$, soit celle vers $-\infty$ de ce réel.

Monotone L'arrondi est croissant : soient deux réels tels que $r_1 < r_2$ et soient deux nombres flottants tels que f_1 est un arrondi de r_1 et f_2 est un arrondi de r_2, alors $f_1 \leq f_2$.

On peut ensuite aisément vérifier que les projections vers $\pm\infty$ précédemment définies sont bien des arrondis.

2.4.2 Arrondis IEEE-754

Les arrondis vers $\pm\infty$ étant déjà définis convenablement, on s'intéresse maintenant aux autres arrondis. Je présente tout d'abord les définitions des arrondis au plus proche et au plus proche pair.

Énoncé Coq
```
Definition Closest (r : R) (p : float) :=
  Fbounded b p /\
  (forall f : float, Fbounded b f -> (Rabs (p - r) <= Rabs (f - r))%R).

Definition EvenClosest (r : R) (p : float) :=
  Closest r p /\
  (FNeven b radix precision p
      \/ (forall q : float, Closest r q -> q = p :>R)).
```

L'arrondi au plus proche ➢ Closest est noté $\circ(\cdot)$ et correspond à l'un des nombres flottants les plus proches du réel. Cette notation est à rapprocher des notations mathématiques pour des fonctions f et g aux alentours de l'infini : $f = \Theta(g)$, pour f est de l'ordre de grandeur de g, à une constante près. Cette notation est bien souvent préférée à $f \in \Theta(g)$ pourtant plus rigoureuse.

Il est générique : pour un réel r, l'arrondi au plus proche est un nombre flottant représentable g tel que pour tout nombre flottant représentable f, on a $|g - r| \leq |f - r|$. Cela signifie que, en général, il n'y a qu'un seul arrondi au plus proche qui est le flottant le plus proche de r. Mais lorsque le réel est à mi-distance de deux nombres flottants, aucune règle n'est édictée et les deux nombres flottants conviennent, comme le montre la figure 2.4. Cela permet de recouvrir à la fois l'arrondi au plus proche pair (voir ci-dessous), l'arrondi au plus proche « extérieur » [IEE04] (on choisit le nombre flottant le plus loin de zéro en cas d'égalité) et l'arrondi au plus proche vers $+\infty$, le plus économique en matériel et qui est « ajouter un demi-ulp à la mantisse et tronquer ». Le choix du bon nombre flottant lorsque le réel est à mi-chemin est un problème difficile bien connu [RK75].

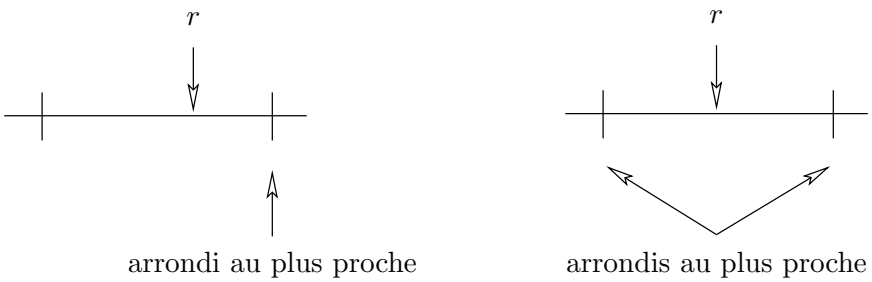


Fig. 2.4 – Arrondi(s) au plus proche.

L'arrondi au plus proche pair ➢ EvenClosest est défini ainsi : pour un réel r, c'est un nombre flottant g qui est un des arrondis au plus proche, mais en plus, soit g est pair, soit

g est l'unique arrondi au plus proche. Cela permet d'imposer le choix défini par la norme IEEE-754 : en cas d'égalité, il faut choisir le nombre flottant dont la mantisse est paire (finit par un zéro en base 2). Parfois, seul ce dernier arrondi est implanté, comme en Java, ce qui gêne beaucoup toute tentative d'amélioration des calculs [KD98].

On s'intéresse enfin à l'arrondi vers zéro ➢ ToZeroP :

Énoncé Coq
```
Definition ToZeroP (r : R) (p : float) :=
  (0 <= r)%R /\ isMin b radix r p \/ (r <= 0)%R /\ isMax b radix r p.
```

Pour un réel r donné, c'est l'arrondi vers $-\infty$ si $r \geq 0$ et l'arrondi vers $+\infty$ si $r \leq 0$, ce qui est bien équivalent à la définition usuelle.

J'ai donc présenté un arrondi générique en terme de propriétés mathématiques ainsi que les quatre modes d'arrondi de la norme IEEE-754. L. Théry a prouvé dans [DRT01] que ces projections sont des arrondis selon le sens ci-dessus.

2.5 Opérations

Les normes définissent un certain nombre d'opérations admissibles sur les nombres flottants. Celles-ci incluent :
- les opérations mathématiques de base ($+$, $-$, $\times$, $/$, $\sqrt{}$, reste euclidien) ;
- les autres opérations (comparaisons, conversions en tous genres...)

Pour les opérations mathématiques de base, l'exigence est claire : c'est celle d'un arrondi correct. Cela signifie que le résultat du calcul doit être le même que si l'ordinateur calcule le résultat idéal avec une précision infinie et arrondit ensuite.

On considère également des opérations mathématiques plus complexes telles que les fonctions trigonométriques (cos, tan...), trigonométriques inverses, hyperboliques... Elles ne sont pas normalisées mais sont fournies d'une manière ou d'une autre à l'utilisateur final, parfois en logiciel. Pour ces opérations, il y a peu de spécifications à part celles de [IEC01] et il n'y a aucune exigence sur comment elles sont calculées, j'ai étudié la façon dont elles sont calculées en pratique (voir les chapitres 9 et 10). Il faut noter que pour ces fonctions, un arrondi correct est très difficile à obtenir à cause du dilemme du fabricant de tables [Gol91, Lef00, LM01].

Là encore, en ce qui concerne la formalisation, la généralisation est maximale : nous ne voulons pas lister les opérations autorisées, car une nouvelle opération (telle le FMA, voir la section 9.4) peut surgir en cas de révision de la norme. Nous utilisons tout simplement un mode d'arrondi (comme défini ci-dessus) ainsi que l'opération mathématique exacte. Cela couvre également facilement les conversions, même difficiles [Gol67].

Ainsi, le programme

$$u = \circ_{p_1}(a + b) \tag{2.1}$$
$$v = \triangle_{p_1}(u \times c) \tag{2.2}$$
$$w = \circ_{p_2}(v) \tag{2.3}$$

devient, après avoir défini la base et les formats flottants b1 correspondant à p_1 bits et b2 correspondant à p_2 bits :

Énoncé Coq
```
Variables a b c u v w: float.
Hypotheses H1 : (Closest b1 radix (a+b)%R u).
Hypotheses H2 : (isMax   b1 radix (u*c)%R v).
Hypotheses H3 : (Closest b2 radix   v     w).
```

2.6 Avantages/inconvénients

2.6.1 Multiples représentations

La principale caractéristique de cette formalisation est la multiplicité des représentants. En effet, plusieurs nombres flottants, que j'appelle une cohorte comme dans [IEE04], peuvent correspondre à la même valeur réelle :

$$(1,3)_{10} =_{\mathbb{R}} (10,2)_{10} =_{\mathbb{R}} (100,1)_{10} =_{\mathbb{R}} (1000,0)_{10}.$$

Ce n'est pas toujours le cas, mais lorsque cela arrive, tous ces représentants ne partagent pas les mêmes propriétés : les uns peuvent être représentables sans que tous le soient, l'un (au plus) peut avoir une mantisse normalisée (commençant par un chiffre non nul)... Cela peut être un inconvénient lorsque l'on perd l'intuition « du » nombre flottant égal à 1. Mais, cela peut être un avantage, car cela permet de manipuler toute la cohorte en même temps et donc de choisir le représentant adapté selon le moment. De plus, une implantation réelle non normalisée peut servir à faire de l'analyse d'erreur [AM59, Car59, Wad60].

Pour être en mesure de revenir au comportement attendu des nombres flottants, nous avons défini les nombres flottants canoniques ➢ Fcanonic. Un canonique est soit un normalisé ➢ Fnormal soit un dénormalisé ➢ Fsubnormal.

Un normalisé ➢ Fnormal est un nombre flottant (n, e) représentable tel que la mantisse a son chiffre de poids le plus fort non nul, ce qui s'écrit $\beta^p \leq |\beta \times n|$.

Un dénormalisé ➢ Fsubnormal est un nombre flottant (n, e) représentable tel que son exposant est minimal et sa mantisse a un chiffre de poids fort nul, ce qui s'écrit $e = -E_i$ et $|\beta \times n| < \beta^p$.

Énoncé Coq
```
Definition Fnormal (p : float) :=
  Fbounded b p /\ (Zpos (vNum b) <= Zabs (radix * Fnum p))%Z.

Definition Fsubnormal (p : float) :=
  Fbounded b p /\
  Fexp p = (- dExp b)%Z /\ (Zabs (radix * Fnum p) < Zpos (vNum b))%Z.

Definition Fcanonic (a : float) := Fnormal a \/ Fsubnormal a.
```

Bien sûr, il a été prouvé que chaque nombre flottant a un et un seul représentant canonique, que l'on obtient par la fonction Fnormalize. Ainsi, en ne considérant que des nombres flottants canoniques au lieu de nombres flottants représentables, on peut, si on le désire, revenir au comportement habituel.

2.6.2 Définition de l'ulp

Une donnée très souvent utilisée pour caractériser un nombre flottant et l'erreur éventuellement commise en le calculant est l'ulp, qui signifie « Unit in the Last Place » [Knu97]. C'est la valeur intrinsèque du dernier bit du nombre flottant machine (s'il est mis à 1). Pour un nombre flottant IEEE-754 en précision p et d'exposant biaisé e, cette valeur est obtenue en calculant 2^{e-p+1}.

Cette valeur est très utile car si le nombre flottant f est le résultat d'un arrondi quelconque du réel r, alors $|r - f| < \text{ulp}(f)$ et si l'arrondi est au plus proche, on sait alors que

$$|r - f| \leq \frac{1}{2}\text{ulp}(f).$$

La définition nécessite donc la connaissance de l'exposant, mais aussi la garantie que la mantisse est normalisée (ou à défaut que l'exposant est minimal), bref que la paire Coq soit canonique. On définit donc l'ulp ➢ Fulp comme la base puissance l'exposant du canonique associé.

Énoncé Coq
```
Definition Fulp (p : float) :=
  powerRZ radix (Fexp (Fnormalize radix b precision p)).
```

Cette définition est malheureusement assez difficile à utiliser. En effet, soit j'oblige les nombres flottants à être canoniques et je perds tous les avantages des représentations multiples, soit je dois considérer en supplément le nombre flottant canonique pour manipuler la valeur de l'ulp.

2.6.3 Nombres flottants et réels

Notre formalisation se base lourdement sur les réels ➢ R de Coq, or les réels de Coq sont axiomatisés [May01]. Si nous voulions nous passer de ces axiomes, nous pourrions utiliser d'autres travaux ayant pour objectif d'avoir des réels constructifs [GPWZ02]. Notre bibliothèque n'utilise pas la définition explicite des réels mais seulement des théorèmes permettant de calculer sur les réels : simplifications, inégalités, calculs simples... donc à condition de retrouver ces résultats basiques, nous restons libres du choix des réels que nous utilisons. Nous avons choisi les réels de Coq car ils sont dans la bibliothèque standard, ce qui assure un suivi et une compatibilité avec d'autres développements.

2.7 Conclusion

La formalisation présentée a de nombreux avantages en comparaison d'autres formalisations : elle est notamment plus générique au niveau de la base et des représentations autorisées. Cela a quelques conséquences non-intuitives, mais qui peuvent se résoudre en ajoutant des contraintes.

Ayant présenté toutes les formalisations, je peux maintenant présenter quelques caractéristiques importantes comparables de ces formalisations. Je les ai résumées dans le tableau de la figure 2.5.

Auteur(s)	**Langage(s)**	**Spé**	**P**	**V2**	$\forall\beta$	**E**
Barrett	Z	✓		✓		✓
Miner/Carreño	PVS/HOL	✓		✓	✓	✓
Russinoff	ACL2	✓	✓	✓		
Jacobi	PVS	✓	✓	✓		✓
Harrison	HOL Light	✓	✓			✓
Théry, Boldo ...	Coq	✓	✓[1]	✓[2]	✓	

Fig. 2.5 – Caractéristiques principales des différents travaux du domaine.

L'assistant de preuve utilisé a été repris en colonne 2. La colonne 3, marquée « Spé », indique la création d'une spécification nouvelle. La colonne 4, marquée « P », indique l'existence d'une quantité importante de preuves utilisant la spécification. La colonne 5, marquée « V2 », indique la possibilité de l'utilisation de vecteurs de bits pour représenter la mantisse (en base 2). La colonne 6, marquée « $\forall\beta$ », indique la possibilité de choisir une base autre que 2. La colonne 7, marquée « E », indique que le mécanisme des exceptions de la norme IEEE-754 a été formalisé.

Les seules formalisations permettant l'utilisation d'une base différente de 2 sont donc la nôtre et celles de Miner et Carreño sachant que ces dernières n'autorisent en pratique que 2 et 10 et ne sont que des spécifications. Je n'ai pas formalisé le mécanisme des exceptions pour l'instant pour les raisons expliquées en section 11.4.1.

Étant donnée cette formalisation, je vais tout d'abord présenter un certain nombre de preuves simples qui permettent aussi de tester le système formel et son usage dans les conditions où il est supposé être le plus performant.

[1] Voir tous les résultats des chapitres 3 à 10.
[2] Ceci est présenté au chapitre 5.

Première partie

Test et propriétés du modèle formel

Chapitre 3

Affiner des résultats connus

Anecdotal evidence suggests that as many as a third of all papers published in mathematical journals contain mistakes – not just minor errors, but incorrect theorems and proofs.
Leslie Lamport [Lam95]

J'ai retrouvé des résultats connus concernant la représentabilité de l'erreur d'un calcul. L'intérêt est alors dans l'utilisation de l'assistant de preuves qui permet de cerner précisément les hypothèses nécessaires lors d'un dépassement de capacité inférieur. J'ai pu extraire des conditions nécessaires et suffisantes à l'existence de ces termes de correction, quelles que soient les entrées.

Selon Lamport, au moins un tiers des articles de journaux en mathématiques sont faux ! Comme expliqué précédemment, les preuves formelles permettent d'éviter cet écueil. Elles permettent aussi de limiter l'influence d'une autre pierre d'achoppement : les hypothèses excédentaires. Les preuves papiers sont en effet très difficiles à manipuler après leur conception : l'auteur peut estimer qu'une hypothèse est excédentaire, mais pour être sûr, il faudrait revérifier chaque ligne de la preuve, ce qui est extrêmement long et ennuyeux et donc facilement sujet à erreurs. Les preuves formelles permettent d'essayer de supprimer de telles hypothèses en toute sécurité : chaque utilisation d'une hypothèse est claire (soit explicitement dans la preuve, soit pour l'assistant de preuves) et une suppression abusive fait échouer la preuve. Je peux ainsi travailler sur la preuve après son élaboration de façon très simple dans le but d'améliorer les résultats existants.

Dans [BWKM91], G. Bohlender *et al.* montrent l'existence de termes d'erreurs « à condition qu'il n'y ait pas de dépassement de capacité inférieur ». J'ai vérifié l'exactitude de ces faits et étudié à l'aide de Coq les conséquences d'un dépassement de capacité inférieur : dans quels cas exactement ces termes ne peuvent-ils pas exister ? Peut-on donner des conditions nécessaires et suffisantes à l'existence de ces termes ? La réponse à cette dernière question est oui grâce à notre formalisation générique et à la multiplicité des représentations possibles.

Bien avant [BWKM91], l'existence de ces termes correctifs a permis le développement des expansions (voir [Møl65b, Møl65a, Dek71, She97, MF01] et le chapitre 8) et de la correction des erreurs numériques d'arrondi [Lan01] (méthode CENA).

Plus précisément, les opérations implantées (addition, soustraction, multiplication, division, racine carrée) peuvent fournir à la fois un résultat et un terme exact de correction, tous deux utilisant le même format que les entrées. Depuis 1965, tous les auteurs ont supposé qu'aucun dépassement de capacité inférieur ou supérieur ne se produisait [Møl65b, Møl65a, Dek71, BWKM91, PV93, Knu97]. Un dépassement de capacité supérieur n'est pas dangereux car il produit rapidement des infinis et/ou des *NaN*. Un dépassement de capacité inférieur peut être fatal au processus car il produit des résultats faux avec peu d'avertissements.

On appelle terme correctif les termes suivants selon les opérations :

Résultat	Terme correctif
$x \boxplus y$	$x + y - x \boxplus y$
$x \boxminus y$	$x - y - x \boxminus y$
$x \boxtimes y$	$x \times y - x \boxtimes y$
$x \boxslash y$	$x - (x \boxslash y) \times y$
$\boxsqrt{} y$	$y - (\boxsqrt{} y)^2$

où $\boxplus$, $\boxminus$, $\boxtimes$, $\boxslash$ et $\boxsqrt{}$ sont respectivement l'addition, la soustraction, la multiplication, la division et la racine carrée arrondies suivant un mode d'arrondi quelconque respectant les conditions du chapitre précédent, mais un arrondi au plus proche est parfois nécessaire comme dans les paragraphes 3.1 et 3.4. Les opérations en arrondi au plus proche seront notées dans un cercle (plutôt qu'un carré pour l'arrondi quelconque), comme par exemple $\oplus$ et $\ominus$.

Il s'agit d'un terme additif pour l'addition/soustraction et la multiplication, du reste euclidien pour la division et d'un reste pour la racine carrée. Je considérerai ici un unique format flottant de précision p, sur lequel seront représentées les entrées et, sous conditions, le résultat et le terme correctif.

Étant donnée notre formalisation des flottants (voir chapitre 2), les preuves sont ici légèrement différentes. Par exemple, pour deux flottants x et y, pour prouver que $x + y$ est représentable, il suffit de trouver un entier relatif e tel que $e \geq -E_i$, $e \leq e_x$, $e \leq e_y$ et $|x + y|2^{-e} < 2^p$. L'exposant sera toujours choisi pour garantir que la mantisse est entière.

3.1 Addition/soustraction

Le terme correctif de l'addition peut être représenter uniquement lorsque l'arrondi choisi est au plus proche [Møl65b, Møl65a, Dek71, Knu97], et ceci est vrai quel que soit l'arrondi au plus proche, même l'arrondi optimal de [PV93]. L'un des résultats de L. Théry dans [DRT01] est :

Lemme 1 (`errorBoundedPlus` provenant de `ClosestPlus`)
Soient deux nombres flottants représentables x et y, alors il existe des nombres flottants représentables dont les valeurs sont $x + y - x \oplus y$ et $x - y - x \ominus y$.

Pour justifier que l'arrondi doit être au plus proche, je donne un contre-exemple : si le mode d'arrondi n'est pas au plus proche, cela peut rendre le terme correctif non représentable. C'est le cas en utilisant l'arrondi dirigé vers $+\infty$: je considère la base 2 et une précision $p > 4$ arbitraire. Je considère les deux nombres flottants normalisés $(-(2^{p-1}+1), p+1)_2$ et $(2^p-3, 0)_2$.

Le premier nombre flottant vaut $-2^{2p}-2^{p+1}$ et le résultat exact de l'addition est $-2^{2p}-2^p-3$, arrondi en -2^{2p}. L'erreur est alors -2^p-3 qui ne peut pas être représentée. L'utilisation d'un double arrondi comme celui du TMS 320 C3x de Texas Instruments (voir le chapitre 7) sur les mêmes entrées fournit le même terme d'erreur non représentable -2^p-3.

3.2 Multiplication

La multiplication est l'opération la plus simple en ce qui concerne les termes correctifs, car l'ordre de grandeur de l'erreur est connu. Un algorithme pour calculer l'arrondi et la correction est connu depuis longtemps [Dek71, Lin81]. Vu son faible coût en matériel, l'IBM S/370 avait dès 1971 une instruction pour calculer à la fois l'arrondi et le terme correctif de la multiplication [KM97].

L'algorithme peut être simplifié si la machine dispose d'un FMA (Fused-Multiply-and-Add, voir la section 9.4) qui permet de calculer $a \times x + y$ avec un seul arrondi. Ce lemme est également un développement de L. Théry dans [DRT01] :

Lemme 2 (`errorBoundedMult` provenant de `FroundMult`)
Soient x et y deux nombres flottants représentables. Un nombre flottant représentable égal à $x \times y - x \boxtimes y$ existe s'il existe deux représentants (n_x, e_x) et (n_y, e_y) de x et y tels que

$$e_x + e_y \geq -E_i.$$

Pour justifier la rigueur de l'inégalité, j'exhibe un contre-exemple à la limite de sa validité. Je me place en base 2 et en précision $p = 4$ et avec une valeur $E_i > 0$ arbitraire. Je considère les deux nombres flottants représentant $9 \times 2^{-\left\lfloor \frac{E_i}{2} \right\rfloor - 1}$ et $11 \times 2^{-\left\lceil \frac{E_i}{2} \right\rceil}$:

$$\left(1001_2, -\left\lfloor \frac{E_i}{2} \right\rfloor - 1\right)_2 \qquad \text{et} \qquad \left(1011_2, -\left\lceil \frac{E_i}{2} \right\rceil\right)_2.$$

Le produit exact $99 \times 2^{-E_i-1}$ s'arrondit au plus proche en $96 \times 2^{-E_i-1} = 12 \times 2^{-E_i+2}$ représenté par le nombre flottant $(1100_2, -E_i + 2)_2$. Le terme correctif idéal est $3 \times 2^{-E_i-1}$ qui ne peut être représenté dans ce format. Il faut noter qu'aucune des entrées n'est un dénormalisé.

Il manque au lemme précédent que les nombres flottants $x \boxtimes y$ et $x \times y - x \boxtimes y$ sont « collés ». En effet en pratique, leurs mantisses se suivent comme le montre la figure 3.1 et le théorème suivant.

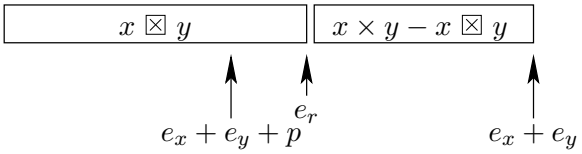


FIG. 3.1 – Positions des mantisses de l'arrondi et du reste d'une multiplication.

Lemme 3 (`errorBoundedMultExp` provenant de `FroundMult`)
Soient x et y deux nombres flottants représentables dans un format (β^p, E_i). On suppose qu'il existe deux représentants (n_x, e_x) et (n_y, e_y) de x et y tels que $e_x + e_y \geq -E_i$. Alors il existe deux nombres flottants représentables r et s tels que $r = (n_r, e_r) =_{\mathbb{R}} x \boxtimes y$ et $s = (n_s, e_s) =_{\mathbb{R}} x \times y - x \boxtimes y$ et

$$e_x + e_y = e_s \leq e_r \leq e_x + e_y + p.$$

3.3 Division

Le théorème suivant explicite deux conditions pour que le terme correctif soit représentable. La première condition (3.1) était attendue et est du même type que les conditions précédentes. Mais la seconde condition (3.2) est bien plus surprenante : elle est nouvelle et ne concerne que les arrondis dirigés.

Théorème 4 (`errorBoundedDiv` provenant de `FroundDivSqrt`)
Soient x et y deux nombres flottants représentables avec $y \neq 0$. Soit q un arrondi de x/y. Le terme correctif $x - q \times y$ est représentable s'il existe deux représentants (n_y, e_y) et (n_q, e_q) de y et q tels que :

$$e_y + e_q \geq -E_i, \tag{3.1}$$

et

$$|q| \neq \beta^{-E_i} \quad \text{ou} \quad \frac{\beta^{-E_i}}{2} \leq \left|\frac{x}{y}\right|. \tag{3.2}$$

Énoncé Coq

```
Theorem errorBoundedDiv :
 forall (x y q : float) P,
  (RoundedModeP b radix P) ->
  (Fbounded b x) -> (Fbounded b y) -> (Fbounded b q) ->
  y <> 0 :>R -> (P (x / y)%R q) ->
  (- dExp b <= Fexp q + Fexp y)%Z ->
  (Rabs q) <> (powerRZ radix (- dExp b)) \/
   ((powerRZ radix (- dExp b)) / 2%nat <= Rabs (x / y))%R ->
      (Fbounded b (Fminus radix x (Fmult q y))).
```

Je suppose que les hypothèses du théorème sont remplies. Soit (n_x, e_x) un représentant de x. Soit $(n_{\tilde{q}}, e_{\tilde{q}})$ la représentation canonique de q. J'ai donc $n_q \times \beta^{e_q} = n_{\tilde{q}} \times \beta^{e_{\tilde{q}}}$.

J'utilise alors le lemme suivant :

Lemme 5 (`FcanonicLeastExp` provenant de `Fnorm`)
Soient deux nombres flottants représentables x et y et que y est canonique. Je suppose également que $x =_{\mathbb{R}} y$. Alors,

$$e_y \leq e_x.$$

J'en déduis donc que $e_{\tilde{q}} \leq e_q$. Or je sais $\left|\frac{x}{y} - q\right| < \text{ulp}(q)$ d'après les propriétés de l'arrondi. Je définis r par :

$$\begin{aligned} e_r &= \min(e_x, e_q + e_y) \\ n_r &= n_x \times \beta^{e_x - e_r} \\ &\quad -n_q \times n_y \times \beta^{e_q + e_y - e_r}. \end{aligned}$$

Il est aisé de montrer que $r =_{\mathbb{R}} x - q \times y$. Pour montrer que r est représentable, je subdivise la preuve en deux cas :

1. Si $e_q + e_y \leq e_x$, alors $e_r = e_q + e_y$ et $e_r \geq -E_i$ par (3.1). J'ai alors

$$\begin{aligned} |n_r| &= |r| \times \beta^{-e_r} = |x - qy| \times \beta^{-e_q - e_y} \\ &\leq |y| \times \left|\frac{x}{y} - q\right| \times \beta^{-e_q - e_y} \\ &< |n_y| \times \text{ulp}(q) \times \beta^{-e_q} \\ &= |n_y| \times \beta^{e_{\tilde{q}} - e_q} \leq |n_y| \\ &< \beta^p \end{aligned}$$

2. Sinon, j'ai $e_x < e_q + e_y$ donc $e_r = e_x$ et $e_r \geq -E_i$ car x est représentable. Je subdivise alors ici encore selon la valeur de $|n_{\tilde{q}}|$.
   Si $|n_{\tilde{q}}| = 0$, alors $q = 0$ et $r = x$ est représentable.
   Si $|n_{\tilde{q}}| > 1$,

$$\begin{aligned} \left|\frac{x}{y} - q\right| &< \text{ulp}(q) \\ |n_{\tilde{q}}| \times |x - qy| &< |n_{\tilde{q}}| \times \text{ulp}(q) \times |y| \\ &= |q| \times |y| \\ &\leq |x| + |x - qy| \end{aligned}$$

   et donc $(|n_{\tilde{q}}| - 1) \times |x - qy| \leq |x|$. Comme $|n_{\tilde{q}}| \geq 2$, je sais que $|x - qy| \leq |x|$ et donc $|n_r| \leq |n_x| < \beta^p$.
   Le dernier cas correspond à $|n_{\tilde{q}}| = 1$. Comme $(n_{\tilde{q}}, e_{\tilde{q}})$ est une représentation canonique, cela signifie que q est dénormalisé et donc que $q = \pm\beta^{-E_i}$. Je déduis alors $\beta^{-E_i}/2 \leq |x/y|$ de l'hypothèse (3.2). Or $|x/y| < 2 \times \beta^{-E_i}$ puisque q est un arrondi de x/y. Je conclus donc que

$$\frac{|q|}{2} \leq \left|\frac{x}{y}\right| < 2\,|q|$$

   Si $\left|\frac{x}{y}\right| \geq |q|$, alors $\left|\left|\frac{x}{y}\right| - |q|\right| = \left|\frac{x}{y}\right| - |q| < 2\,|q| - |q| = |q| \leq \left|\frac{x}{y}\right|$.
   Si $\left|\frac{x}{y}\right| \leq |q|$, alors $\left|\left|\frac{x}{y}\right| - |q|\right| = |q| - \left|\frac{x}{y}\right| \leq 2\,\left|\frac{x}{y}\right| - \left|\frac{x}{y}\right| = \left|\frac{x}{y}\right|$.

Dans tous les cas, on a

$$\left|\left|\frac{x}{y}\right| - |q|\right| \leq \left|\frac{x}{y}\right|.$$

Comme $\frac{x}{y}$ et q ont le même signe (propriétés RleRoundedR0 et RleRoundedLessR0 de tout mode d'arrondi), il vient

$$\left|\frac{x}{y} - q\right| \leq \left|\frac{x}{y}\right| \qquad \text{et} \qquad |x - qy| \leq |x|,$$

ce qui finit la preuve.

J'ai de plus prouvé que l'hypothèse (3.2) est toujours vérifiée en arrondi au plus proche dans le théorème errorBoundedDivClosest (du même fichier) et en arrondi vers zéro dans le théorème errorBoundedDivToZero.

Je vais maintenant justifier la rigueur des hypothèses (3.1) et (3.2) : j'exhibe des contre-exemples lorsque ces conditions ne sont pas vérifiées.

– Pour l'hypothèse (3.1), considérons la base 2, une précision $p = 4$ et les nombres flottants

$$\begin{aligned} x &= 1001_2 \times 2^{-E_i+3} \\ y &= 1101_2 \times 2^{-\left\lfloor \frac{E_i}{2} \right\rfloor}. \end{aligned}$$

La division est arrondie au plus proche (c'est-à-dire vers $-\infty$ dans ce cas). J'obtiens $q = 1011_2 \times 2^{-1-\left\lceil \frac{E_i}{2} \right\rceil}$ et $e_q + e_y = -E_i - 1$. Le terme correctif $x - qy$ ne peut être représenté :

$$r = 2^{-E_i-1}.$$

– Le contre-exemple correspondant à (3.2) est plus surprenant. Je considère la base 2, une valeur de E_i et une précision arbitraires et les nombres flottants

$$\begin{aligned} x &= 2^{-E_i+p} - 2^{-E_i+1} \\ y &= 2^{p+1}. \end{aligned}$$

La division exacte $\frac{x}{y} = \frac{2^{-E_i}}{2} - 2^{-E_i-p}$ est arrondie vers $+\infty$ ou en utilisant un double arrondi. J'obtiens $q = 2^{-E_i}$.
Le terme correctif $x - qy$ est

$$-\left(2^p + 1\right) \times 2^{-E_i},$$

qui d'une part ne peut être représenté, et d'autre part est bien plus grand que x.

Un terme correctif grand (relativement aux entrées) est obtenu par exemple en arrondi vers $+\infty$ avec $x = 2^{-E_i+p}$ et $y = 2^{2p+1}$: l'arrondi q vaut 2^{-E_i} et $x - qy = -\left(2^{p+1} - 1\right) \times 2^{-E_i+p}$ c'est-à-dire $|x - qy| \approx x2^{p+1}$! Une telle situation n'a jamais été présentée auparavant mais peut être reliée aux sauts erreur relative/erreur absolue présentés dans [Dem84].

Ici aussi, aucune entrée n'est un dénormalisé.

3.4 Racine carrée

Par rapport au théorème précédent, on a encore une hypothèse du type 3.1. Par contre, on n'a plus l'hypothèse du type 3.2 qui est remplacée par le fait que le terme correctif ne peut être défini que si le mode d'arrondi est au plus proche. Cette preuve est tout à fait nouvelle car elle contient un cas qui n'avait jamais été exploré jusqu'à maintenant et donc une preuve originale.

Théorème 6 (`errorBoundedSqrt` provenant de `FroundDivSqrt`)
Soit x un nombre flottant représentable positif. Le terme correctif $x - (\circ(\sqrt{x}))^2$ est représentable s'il existe un nombre flottant (n_q, e_q) représentant $\circ(\sqrt{x})$ tel que

$$2\ e_q \geq -E_i.$$

Énoncé Coq
```
Theorem errorBoundedSqrt :
 forall x q : float,
 (Fbounded b x) -> (Fbounded b q) ->
 (0 <= x)%R -> (Closest b radix (sqrt x) q) ->
 (- dExp b <= Fexp q + Fexp q)%Z ->
   (Fbounded b (Fminus radix x (Fmult q q))).
```

Je note $q = (n_q, e_q)$ le représentant de l'arrondi au plus proche de $\sqrt{x}$ vérifiant la condition et (n_x, e_x) un représentant de x. Soit $(n_{\tilde{q}}, e_{\tilde{q}})$ la représentation canonique de q. Je sais par les propriétés de l'arrondi au plus proche que :

$$\left|\sqrt{x} - q\right| \leq \frac{\mathrm{ulp}(q)}{2}.$$

Je définis r comme suit :

$$\begin{array}{rcl} e_r & = & \min(e_x, 2\ e_q) \\ n_r & = & n_x \times \beta^{e_x - e_r} \\ & & -n_q^2 \times \beta^{2\ e_q - e_r}. \end{array}$$

Je montre facilement que $r =_{\mathbb{R}} x - q^2$. Pour montrer que r est représentable, je subdivise ici encore les cas.

1. Si $2\ e_q \leq e_x$, alors $e_r = 2\ e_q$ et $e_r \geq -E_i$ par hypothèse. J'ai ensuite :

$$\begin{aligned}
|n_r| &= \left|x - q^2\right| \times \beta^{-2\ e_q} \\
&= \left|\sqrt{x} - q\right| \times \left|\sqrt{x} + q\right| \times \beta^{-2\ e_q} \\
&\leq \frac{\mathrm{ulp}(q)}{2} \times \left(\left|\sqrt{x} - q\right| + 2\ |q|\right) \times \beta^{-2\ e_q} \\
&\leq \frac{\mathrm{ulp}(q)}{2} \times \left(\frac{\mathrm{ulp}(q)}{2} + 2\ |q|\right) \times \beta^{-2\ e_q} \\
&\leq \frac{1}{4} \times \beta^{2\ e_{\tilde{q}} - 2\ e_q} + |n_{\tilde{q}}| \times \beta^{2\ e_{\tilde{q}} - 2\ e_q} \\
&\leq \frac{1}{4} + |n_{\tilde{q}}| \leq \frac{1}{4} + (\beta^p - 1) = \beta^p - \frac{3}{4} \\
&< \beta^p
\end{aligned}$$

2. Sinon, j'ai $e_x < 2\ e_q$ et $e_r = e_x$ donc $e_r \geq -E_i$. Il faut ici examiner deux cas selon $\tilde{q}$.

   Si $(n_{\tilde{q}}, e_{\tilde{q}})$ est normalisé, alors j'utilise le fait que $\mathrm{ulp}(q) \leq |q| \times \beta^{1-p}$ selon FulpLe2, que $q \geq 0$ (par les propriétés de l'arrondi et car $\sqrt{x} \geq 0$) et que $p \geq 2$, donc

$$\begin{aligned}
q &\leq \sqrt{x} + \frac{\mathrm{ulp}(q)}{2} \\
&\leq \sqrt{x} + \frac{1}{2} \times \beta^{1-p} \times q \\
q &\leq \frac{\sqrt{x}}{1 - \frac{\beta^{1-p}}{2}} \\
q^2 &\leq x \times \left(\frac{1}{1 - \frac{\beta^{1-p}}{2}}\right)^2 \\
&\leq 2\ x
\end{aligned}$$

   Comme q^2 et x ont le même signe (positif) et que $q^2 \leq 2\ x$, j'en déduis $|x - q^2| \leq x$ d'où $|n_r| \leq |n_x| < \beta^p$.

   Le cas où $(n_{\tilde{q}}, e_{\tilde{q}})$ est dénormalisé ne peut arriver dans un système flottant *raisonnable* comme la simple ou la double précision, il a donc été logiquement négligé par les précédents auteurs. Mais si la base est 2 et la précision $p = 5$ et que l'exposant minimum est $-E_i = -3$, alors la représentation canonique de la racine carrée de 1 (nombre flottant $(1, 0)$) est dénormalisée : $(01000_2, -3)$! Dès que $p > E_i$, j'ai des valeurs relativement grandes (d'exposant positif) qui ont un représentant dénormalisé : voir la figure 3.2. Bien sûr, ce format est idiot, mais le système de déduction automatique n'a aucun moyen de le deviner. Et c'est souhaitable, car l'assistant de preuve nous permet ici de mettre précisément le doigt sur toutes les hypothèses implicites des preuves papier.

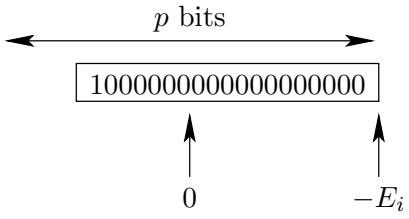


FIG. 3.2 – Format flottant où le calcul d'une racine carrée peut être dénormalisé.

Je sais donc que $e_{\tilde{q}} = -E_i$ et

$$\begin{aligned}
|n_r| &= \left|x - q^2\right| \times \beta^{-e_x} \\
&= \left|\sqrt{x} - q\right| \times \left|\sqrt{x} + q\right| \times \beta^{-e_x} \\
&\leq \frac{\beta^{-E_i}}{2} \times \left(\left|\sqrt{x} - q\right| + 2\,\sqrt{x}\right) \times \beta^{-e_x} \\
&\leq \frac{\beta^{-E_i}}{2} \times \left(\frac{\beta^{-E_i}}{2} + 2\,\sqrt{x}\right) \times \beta^{-e_x} \\
&\leq \frac{1}{4}\beta^{-2\,E_i - e_x} + \sqrt{n_x}\sqrt{\beta^{e_x}}\beta^{-E_i - e_x} \\
&\leq \frac{1}{4} + \sqrt{n_x} \times \sqrt{\beta^{-2E_i - e_x}} \\
&\leq \frac{1}{4} + n_x \leq \beta^p - \frac{3}{4} < \beta^p
\end{aligned}$$

donc la propriété est toujours valable dans ce cas qui n'avait jamais été étudié.

Je justifie la rigueur de l'inégalité et de l'arrondi au plus proche avec les contre-exemples suivants :

– Je prends la base 2 et une précision $p = 4$. Je distingue les cas selon la parité de E_i.
Si E_i est pair, je considère

$$\begin{aligned}
x &= 1010_2 \times 2^{-E_i+1} \\
q &= 1001_2 \times 2^{-1-\frac{E_i}{2}}.
\end{aligned}$$

Je sais alors que $2\,e_q = -E_i - 2$ et que le terme correctif $x - q^2$ vaut -2^{-E_i-2}, qui ne peut être représenté.
Si E_i est impair, j'arrive à la même conclusion en prenant :

$$\begin{aligned}
x &= 1010_2 \times 2^{-E_i+3} \\
q &= 1101_2 \times 2^{\frac{-1-E_i}{2}} \\
x - q^2 &= -1001_2 \times 2^{-E_i-1}.
\end{aligned}$$

– Pour montrer que l'arrondi au plus proche est nécessaire, je considère la base 2 et une précision $p > 4$ arbitraire. Je choisis le nombre flottant $x = (2^p - 3, p+2)_2$ dont la valeur est $2^{2p+2} - 6 \times 2^{p+1}$.
La racine carrée de x peut s'écrire $\sqrt{x} = 2^{p+1} - 3 - \alpha \times 2^{-p}$ avec α entre $\frac{9}{16}$ et $\frac{9\sqrt{2}}{4}$ d'après l'inégalité de Taylor-Lagrange. Un arrondi au plus proche devrait répondre $2^{p+1} - 4$

correspondant au nombre flottant $(2^p - 2, 1)_2$ mais en cas d'arrondi dirigé vers $+\infty$ ou de double arrondi, j'obtiens $2^{p+1} - 2$ correspondant au nombre flottant $(2^p - 1, 1)_2$. Dans ce cas, le terme correctif est $x - q^2 = -2 \times 2^{p+1} - 4 = -(2^p + 1) \times 4$ qui ne peut être représenté.
Pour une preuve en arrondi vers $-\infty$ ou vers 0, il suffit de prendre $x = (2^{p-1} + 3, p + 3)$.

Il est étrange d'avoir dû envisager un cas nouveau et qui peut sembler idiot, mais notre spécification n'est pas censé le deviner. Il faudrait si on le désirait ajouter et spécifier des contraintes telles que celles présentées dans la norme IEEE-854 [CK+84, §3.2] et formalisées dans [Car95, Min95] pour imposer que 1 ne puisse être dénormalisé. Mais nous considérons que les spécifications doivent rester aussi concises que possibles et j'ai réussi à démontrer le résultat sans axiome additionnel. Il est en effet souhaitable d'avoir un système formel avec un nombre minimal d'axiomes pour éviter tout risque de contradiction. Notre bibliothèque ne contient donc aucun axiome additionnel (mais inclut ceux de la bibliothèque des réels), mais chaque fichier et chaque théorème contient son lot d'hypothèses.

3.5 Division entière

Les normes IEEE-754 et 854 définissent l'opération de reste (FPREM) qui doit être implantée pour satisfaire la norme. Étant donnés deux nombres flottants x et y, on définit n qui est l'entier le plus proche de x/y avec la convention de l'arrondi pair et le résultat est défini comme :

$$r = x - ny.$$

Comme pour l'arrondi au plus proche, je considère l'un quelconque des entiers les plus proches de x/y, sans convention particulière en cas d'égalité.

Les deux normes affirment que ce reste peut toujours être représenté dans le même format que les entrées mais cela n'avait jamais été prouvé formellement [Min95]. Les preuves usuelles utilisent l'algorithme d'Euclide : le quotient est calculé bit après bit et le reste peut toujours être représenté dans le même format. Cet invariant a aussi été utilisé dans [BWKM91] pour la preuve concernant la division.

Théorème 7 (`errorBoundedRem` provenant de `FroundDivSqrt`)
Soient x et y deux nombres flottants représentables avec $y \neq 0$ et soit n un entier le plus proche de x/y. Alors $x - n \times y$ est représentable.

Énoncé Coq

```
Theorem errorBoundedRem :
 forall (x y : float) (n : Z),
  (Fbounded b x) -> (Fbounded b y) ->
   y <> 0 :>R ->
  (forall z : Z, (Rabs (n - x / y) <= Rabs (z - x / y))%R) ->
    (Fbounded b
      (Fminus radix (Fnormalize radix b precision x)
      (Fmult (Float n 0) (Fnormalize radix b precision y)))).
```

Je pourrais décrire complètement l'algorithme d'Euclide et prouver ses propriétés mais cela représente un gros travail. Je présente ici une preuve plus élémentaire et qui concerne tous les arrondis au plus proche.

Je définis l'entier le plus proche d'une valeur réelle z comme étant l'un des entiers $n \in \mathbb{Z}$ tel que

$$\forall n' \in \mathbb{Z}, \quad |n - z| \leq |n' - z|.$$

En fait, je n'utilise que le fait que si n est l'entier le plus proche de z alors on a l'inégalité $|n - z| \leq 1/2$.

Je considère que x et y sont positifs et canoniques, les autres cas s'y ramenant assez facilement. Je pose alors r tel que $e_r = \min(e_x, e_y)$ et $r =_{\mathbb{R}} x - n \times y$. Je distingue deux cas :

1. Si $e_y \leq e_x$, alors $e_r = e_y \geq -E_i$ et

$$\begin{array}{rcl} |n_r| & = & |x - n \times y|\beta^{-e_y} \\ & = & |y| \times \left|n - \frac{x}{y}\right| \beta^{-e_y} \\ & = & |n_y| \times \left|n - \frac{x}{y}\right| \\ & \leq & |n_y| \times 1/2 \\ & \leq & |n_y| \\ & < & \beta^p. \end{array}$$

2. Sinon, $e_x < e_y$ et $e_r = e_x \geq -E_i$. Je distingue alors les cas selon la valeur de n.
   Si $n = 0$, alors $r = x$ est représentable.
   Si $n = 1$, alors $r =_{\mathbb{R}} x - y$ est représentable d'après le théorème de Sterbenz (voir [Ste74] et le chapitre 4 page 35). En effet, comme $n = 1$, j'ai $|1 - x/y| \leq 1/2$ d'où $1/2 \leq x/y \leq 3/2$.
   Les autres cas sont impossibles. Comme $|x/y - n| < 1$, j'ai $0 \leq x/y < n + 1$ donc $n \geq 0$. Comme x et y sont canoniques et que $e_x < e_y$, j'ai $x < y$ donc $x/y < 1$ ce qui implique $n \leq 1$. J'aurais donc un entier n tel que $n \geq 0$, $n \leq 1$ et n n'est ni 0 ni 1, ce qui est impossible.

Grâce à l'assistant de preuves, j'ai établi des conditions nécessaires et suffisantes pour la représentabilité des termes d'erreur des opérations de base, même en présence de dépassements de capacités inférieurs. Pour cela, j'exhibe un des éléments de la cohorte des nombres flottants égaux à la valeur réelle donnée et je démontre qu'il est représentable.

Ce résultat est intéressant en soi, mais l'intérêt est également dans la méthode. Au lieu de reprendre les preuves de la littérature, j'ai développé un nouveau style de preuve. Ce n'est absolument pas par manque de confiance envers les preuves de mes prédécesseurs, mais par volonté de simplicité dans l'usage de l'assistant de preuves. Coq rend certains types de preuves bien plus simples à garantir que d'autres et celles de la littérature sont ici mal adaptées. En particulier, la preuve de représentabilité du reste de la division se fait en général par l'algorithme de division « à la main », mais je l'ai faite en exhibant un nombre flottant représentable égal au reste plutôt qu'en expliquant comment le calculer.

Cette méthode est d'une part indiscutable, car elle prouve l'existence du terme d'erreur indépendamment de la façon dont on le calcule et donc de l'algorithme choisi. Mais elle est

d'autre part contestable, car je ne garantis pas la validité du résultat si l'algorithme « à la main » est utilisé : je sais qu'un résultat idéal est représentable, mais pas que c'est celui effectivement fourni. Si le reste de la division entière ou de la division flottante est calculé par un FMA (voir 9.4), mon théorème et les propriétés des arrondis sont suffisants pour garantir que le résultat est exact.

Cette nécessité d'un nouveau type de preuves plus adapté est en partie due à la façon de travailler de l'assistant de preuves. Mais je pense qu'elle est surtout due au caractère récent des assistants de preuves. Les résultats déjà prouvés sont en nombre extrêmement restreint en comparaison des énormes connaissances mathématiques amassées depuis des siècles.

Chapitre 4

Autres calculs exacts

Il existe pour chaque problème complexe une solution simple, directe et fausse.
Henry Louis Mencken

Il est extrêmement pratique de pouvoir déterminer *a priori* ou *a posteriori* qu'un calcul est exact et donc que l'erreur commise a été nulle. Après le très connu théorème de Sterbenz, je montre d'autres résultats du même type dont l'usage me fut bien plus que théorique.

La grande majorité des calculs produit une petite erreur, petite au sens soit de l'erreur absolue, soit de l'erreur relative. Ces erreurs ne peuvent qu'être bornées et propagées comme dans l'analyse de l'erreur directe (« forward analysis ») [Hig02]. Néanmoins, pour pouvoir contrôler ces erreurs et les limiter, beaucoup d'auteurs se sont très tôt intéressés aux calculs exacts, c'est-à-dire ceux dont l'erreur commise est nulle. Le résultat mathématique parfait est dans certains cas représentable exactement par un nombre flottant et les propriétés de projection des arrondis utilisés font alors que le résultat effectivement renvoyé par le processeur est ce résultat idéal.

4.1 Théorème de Sterbenz

Le théorème de ce genre le plus connu est celui de Sterbenz [Ste74], mais sa paternité est contestée par Kahan. Le théorème correspondant a été démontré en Coq par L. Théry.

Lemme 8 (`Sterbenz` provenant de `Fprop`)
Soient deux nombres flottants x et y représentables tels que

$$\frac{y}{2} \leq x \leq 2 \times y,$$

alors $x -_{\mathbb{F}} y$ est un nombre flottant représentable.

où $x -_{\mathbb{F}} y$ est le nombre flottant

$$\left(n_x\beta^{e_x-\min(e_x,e_y)} - n_y\beta^{e_y-\min(e_x,e_y)}, \min(e_x, e_y)\right).$$

Comme précédemment, on exhibe ici un nombre flottant dont la valeur réelle est $x - y$, il ne reste ensuite qu'à prouver qu'il est représentable. Ce théorème bien connu en base 2 est ici généralisé en base quelconque. Pour une extension en notation non IEEE, voir les théorèmes 41 et 42 du chapitre 6.

4.2 Extension : salmigondis de formats flottants

Le théorème 8 ne considère qu'un seul format flottant, or il arrive que plusieurs formats de précisions différentes soient utilisés. Or il est clair que si deux nombres flottants très précis sont très proches, le résultat de leur soustraction est bien entendu exact sur le format très précis, mais le résultat exact tient sur moins de chiffres, comme le montre la figure 4.1.

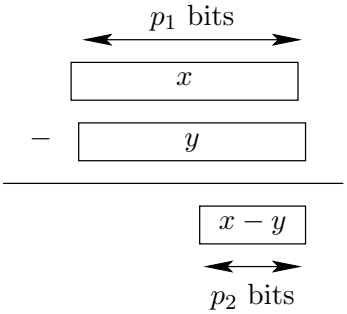


FIG. 4.1 – Extension du théorème de Sterbenz.

Bien sûr, tout ceci est approximatif mais a été formalisé dans le théorème suivant. Il faut considérer deux formats flottants ayant deux précisions *a priori* différentes, mais si les deux précisions sont égales, je retrouve exactement le théorème précédent.

Théorème 9 (`SterbenzApprox2` provenant de `FArgReduct`)
On considère deux formats flottants, un format dit étendu $\mathcal{B}_1$ avec p_1 chiffres et un exposant minimal de $-E_i^1$ et un format dit réduit $\mathcal{B}_2$ avec p_2 chiffres et un exposant minimal de $-E_i^2$, tels que $-E_i^2 \leq -E_i^1$.
Soient deux nombres flottants x et y représentables dans le format étendu $\mathcal{B}_1$ tels que

$$\frac{y}{1+\beta^{p_2-p_1}} \leq x \leq \left(1+\beta^{p_2-p_1}\right) y,$$

alors il existe un nombre flottant représentable dans le format réduit $\mathcal{B}_2$ dont la valeur est $x - y$.

Énoncé Coq

```
Theorem SterbenzApprox2 :
 forall (rho : R) (x y : float),
   (0 < rho)%R ->
   IZR (Zpos (vNum b1)) = (rho * Zpos (vNum b2))%R ->
   (- dExp b2 <= - dExp b1)%Z ->
   Fbounded b1 x -> Fbounded b1 y ->
   (/ (1 + / rho) * y <= x)%R -> (x <= (1 + / rho) * y)%R ->
 Fbounded b2  (Fminus radix
     (Fnormalize radix b1 prec1 x) (Fnormalize radix b1 prec1 y)).
```

Le théorème Coq introduit $\rho \succ$ rho tel que $\rho = \beta^{p_1-p_2}$. En fait, les formats sont dits étendu ou réduit, mais rien ne l'oblige : on peut avoir $\rho \geq 1$ ou $\rho < 1$, c'est-à-dire $p_1 \leq p_2$ ou le contraire.

La preuve est plus simple qu'il n'y paraît. Je suppose que x et y sont canoniques et je considère $u =_{\mathbb{R}} x - y$ d'exposant $\min(e_x, e_y)$. Il reste à montrer que u est représentable dans le format réduit $\mathcal{B}_2$. Une remarque immédiate est que le théorème ainsi rédigé implique que x et y sont positifs.

Je suppose tout d'abord que $y \leq x$ et donc que $x-y \geq 0$. Or comme x et y sont canoniques, le fait que $y \leq x$ implique que $e_y \leq e_x$.

Si je reviens à u, je sais tout d'abord que $e_u = \min(e_x, e_y) = e_y \geq -E_i^1 \geq -E_i^2$ par hypothèse. Je considère maintenant n_u.

$$\begin{aligned} |n_u| &= |u|\beta^{-e_u} = |x-y|\beta^{-e_y} \\ &= (x-y)\beta^{-e_y} \\ &\leq \left(\left(1+\beta^{p_2-p_1}\right)\ y - y\right)\beta^{-e_y} \\ &\leq \beta^{p_2-p_1-e_y} y \\ &\leq \beta^{p_2-p_1} n_y \\ &< \beta^{p_2} \end{aligned}$$

Je suppose maintenant que $x \leq y$ et donc que $x-y \leq 0$. Or comme x et y sont canoniques, le fait que $x \leq y$ implique que $e_x \leq e_y$.

Si je reviens à u, j'ai tout d'abord que $e_u = \min(e_x, e_y) = e_x \geq -E_i^1 \geq -E_i^2$ par hypothèse. Je considère maintenant n_u.

$$\begin{aligned} |n_u| &= |u|\beta^{-e_u} = |x-y|\beta^{-e_x} \\ &= (y-x)\beta^{-e_x} \\ &\leq \left(\left(1+\beta^{p_2-p_1}\right)\ x - x\right)\beta^{-e_x} \\ &\leq \beta^{p_2-p_1-e_x} x \\ &\leq \beta^{p_2-p_1} n_x \\ &< \beta^{p_2} \end{aligned}$$

Ceci achève la preuve dans tous les cas.

J'ai donc quantifié facilement la distance à laquelle doivent être deux nombres flottants pour que leur différence tienne dans un format donné. L'hypothèse sur les exposants minimaux autorisés est naturelle : si x et y sont trop petits pour être représentés dans le format réduit, alors leur différence ne peut être représentée exactement (sauf si elle est nulle). Je demande donc que la plage d'exposant du format réduit recouvre celle du format étendu.

4.3 Seconde extension : le retour du salmigondis

Je peux légèrement améliorer le résultat précédent :

Lemme 10 (`SterbenzApprox` provenant de `FArgReduct`)
On considère deux formats flottants, un format dit étendu $\mathcal{B}_1$ avec p_1 chiffres et un exposant minimal de $-E_i^1$ et un format dit réduit $\mathcal{B}_2$ avec p_2 chiffres et un exposant minimal de $-E_i^2$, tels que $-E_i^2 \leq -E_i^1$.
Soient deux nombres flottants x et y représentables dans le format étendu $\mathcal{B}_1$ tels que

$$\frac{\beta^{p_1}}{\beta^{p_1}+\beta^{p_2}+1}\ y \leq x \leq \frac{\beta^{p_1}+\beta^{p_2}+1}{\beta^{p_1}}\ y,$$

alors il existe un nombre flottant représentable dans le format réduit $\mathcal{B}_2$ dont la valeur est $x-y$.

La preuve se base ici sur les résultats du chapitre 6 sur les systèmes plus généraux et en particulier le complément à la base.

Le premier argument pour aboutir au résultat est l'utilisation des théorèmes 38 et 39. Ils affirment qu'il y a égalité entre les ensembles des flottants représentables par un format flottant usuel : $\mathcal{B} = (\beta^p, E_i)$, et par le format générique (c'est-à-dire du type (N_i, N_s, E_i)) qui correspond : $\mathbb{B} = (\beta^p - 1, \beta^p, E_i)$.

Cela permet de plonger le théorème 43 dans les nombres flottants usuels et en fait, cet énoncé est une application directe du théorème 43 page 65. Ce théorème est utilisé avec les formats flottants génériques suivants : $\mathbb{B}_1 = (\beta^{p_1} - 1, \beta^{p_1}, E_i^1)$ et $\mathbb{B}_2 = (\beta^{p_2} - 1, \beta^{p_2}, E_i^2)$.

Utiliser des formats flottants en complément à la base plutôt que de faire la preuve directement comme précédemment permet d'améliorer très légèrement le théorème 9 : au lieu de

$$\begin{array}{rcccll} \dfrac{y}{1+\beta^{p_2-p_1}} & \leq & x & \leq & (1+\beta^{p_2-p_1})\ y, & \text{j'ai} \\ \dfrac{y}{1+\beta^{p_2-p_1}+\beta^{-p_1}} & \leq & x & \leq & (1+\beta^{p_2-p_1}+\beta^{-p_1})\ y. & \end{array}$$

L'amélioration est certes minime, mais la façon de l'obtenir est intéressante. Je passe par une formalisation plus générale et un théorème difficile. Je prends un corollaire de ce théorème que je plonge dans la formalisation courante pour obtenir un résultat meilleur que celui obtenu directement. Bien sûr, il est possible de prouver directement le théorème 10 sans passer par le complément à la base, mais cela nécessite de regarder un certain nombre de cas particuliers. La preuve est moins directe et simple que ne le sont les preuves présentées.

4.4 Reste de la division euclidienne saupoudré de salmigondis

Cette question me fut soumise par W. Kahan en mai 2004. Elle concerne la représentabilité du reste de la division euclidienne lorsque les entrées sont dans des formats différents. Ce résultat n'était alors supposé être vrai qu'en arrondi au plus proche pair, le reste de la division euclidienne étant supposé occuper un bit de plus lorsque l'arrondi n'est pas pair. J'ai démontré que ce bit supplémentaire n'était pas utile quelle que soit la façon dont on arrondit en cas d'égalité.

On se place en base 2. On suppose ici que les exposants ne sont pas limités : il n'y a donc ni dépassement de capacité supérieur, ni dépassement de capacité inférieur avec des dénormalisés. On suppose donc que tous les nombres flottants sont normalisés.

Théorème 11 (`Divnk` provenant de `Divnk`)
La plage d'exposant est illimitée. Soient deux entiers n et k tels que $n > 1$ et $k > 1$. Soient trois nombres flottants x, y et z tels que
- z est représentable sur n bits et est non nul,
- x est représentable sur $n + k$ bits,
- y est représentable sur k bits et est un arrondi au plus proche de $\frac{x}{z}$ (quelle que soit la règle en cas d'égalité)

Alors $x - yz$ est représentable sur n bits.

Énoncé Coq
```
Hypotheses nGreaterThanOne :  (lt (S O) n).
Hypotheses kGreaterThanOne :  (lt (S O) k).
Hypotheses nGivesBound1 : (Zpos (vNum b1)) = (Zpower_nat radix n).
Hypotheses kGivesBound2 : (Zpos (vNum b2)) = (Zpower_nat radix k).
Hypotheses nkGivesBound3: (Zpos (vNum b3)) = (Zpower_nat radix (plus n k)).
Hypotheses Normz : (Fnormal radix b1 z).
Hypotheses Normy : (Fnormal radix b2 y).
Hypotheses Normx : (Fnormal radix b3 x).
Hypotheses zNonZero: ~((FtoRradix z)=0)%R.
Hypotheses Roundy : (Closest b2 radix (Rdiv x z) y).

Theorem Divnk: ((Fexp y) <> (-(dExp b2)))%Z ->
  (Rlt (Zabs (Fnum (Fminus radix x (Fmult y z)))) (powerRZ radix n)).
```

Pour prouver ce théorème, je vais avoir besoin de 3 lemmes intermédiaires.

Lemme 12 (`Divnk_FexpyGe` provenant de `Divnk`)
Sous les conditions du théorème 11,

$$e_x - e_z \leq e_y.$$

En effet, comme y est un arrondi au plus proche de x/z, j'ai $\left|y - \frac{x}{z}\right| \leq \frac{1}{2}\text{ulp}(y) = 2^{e_y-1}$, donc $|y| + 2^{e_y-1} \geq \left|\frac{x}{z}\right|$.

Comme x, y et z sont représentables sur $n+k$, k et n bits, j'ai $|y| \leq \left(2^k - 1\right) 2^{e_y}$ et $|x| \geq 2^{n+k-1+e_x}$ et enfin $|z| \leq (2^n - 1)\, 2^{e_z}$, donc :

$$\begin{aligned}
\left(2^k - \frac{1}{2}\right) 2^{e_y} &\geq |y| + 2^{e_y-1} \geq \left|\frac{x}{z}\right| \geq \frac{2^{n+k-1+e_x}}{(2^n-1)\, 2^{e_z}} \\
2^{e_y} &\geq \frac{2^{n+k}}{(2^n-1)\left(2^k - \frac{1}{2}\right)} 2^{e_x-e_z-1} > 2^{e_x-e_z-1} \\
e_y &\geq e_x - e_z
\end{aligned}$$

En effet, $1 < \frac{2^{n+k}}{(2^n-1)\left(2^k-\frac{1}{2}\right)}$ est équivalent après calculs à $-2^{n-1} - 2^k + \frac{1}{2} < 0$ qui est vrai puisque n et k sont plus grands que 1.

Lemme 13 (`Divnk_FexpyLe` provenant de `Divnk`)
Sous les conditions du théorème 11,

$$e_y \leq 2 + e_x - e_z.$$

La preuve suit le même chemin que la précédente et est valide pour toute valeur de y.
De $\left|y - \frac{x}{z}\right| \leq 2^{e_y-1}$, j'obtiens $|y| - 2^{e_y-1} \leq \left|\frac{x}{z}\right|$.
Comme $|y| \geq 2^{k-1+e_y}$ et $|x| \leq \left(2^{n+k} - 1\right) 2^{e_x}$ et enfin $|z| \geq 2^{n-1+e_z}$, je déduis

$$\begin{aligned}
\left(2^{k-1} - \frac{1}{2}\right) 2^{e_y} &\leq |y| - 2^{e_y-1} \leq \left|\frac{x}{z}\right| \leq \frac{\left(2^{n+k} - 1\right) 2^{e_x}}{2^{n-1+e_z}} \\
2^{e_y} &\leq \frac{2^{n+k} - 1}{2^{n-1}\left(2^{k-1} - \frac{1}{2}\right)} 2^{e_x-e_z} < 2^{3+e_x-e_z} \\
e_y &\leq 2 + e_x - e_z
\end{aligned}$$

En effet, $\frac{2^{n+k}-1}{2^{n-1}\left(2^{k-1}-\frac{1}{2}\right)} < 2^3$ est équivalent après calculs à $2^{n+1} < 1 + 2^{n+k}$ qui est vrai puisque k est plus grand que 1.

Lemme 14 (`Divnk_FexpyLe2` provenant de `Divnk`)
Sous les conditions du théorème 11,

$$2^{k-1+e_y} \leq \left|\frac{x}{z}\right| \quad \Rightarrow \quad e_y \leq 1 + e_x - e_z.$$

Cette preuve ressemble à celle du lemme précédent, mais c'en est une amélioration qui est presque toujours applicable. Elle l'est sauf lorsque y est une puissance de 2 et que l'arrondi a

été fait vers le haut. Mais ce cas est traité séparément.

$$\begin{aligned} 2^{k-1+e_y} &\leq \left|\frac{x}{z}\right| \leq \frac{\left(2^{n+k}-1\right)2^{e_x}}{2^{n-1+e_z}} \\ 2^{e_y} &\leq \frac{2^{n+k}-1}{2^{n-1}2^{k-1}}2^{e_x-e_z} = \left(2^{n+k}-1\right)2^{e_x-e_z-n-k+2} \\ 2^{e_y} &< 2^{2+e_x-e_z} \\ e_y &\leq 1+e_x-e_z \end{aligned}$$

Je peux maintenant prouver le théorème 11 : $r = x - yz$ est représentable sur n bits.

Je pose $e = \min(e_x, e_y + e_z)$. Je peux alors représenter r par le nombre flottant $r = (n_x 2^{e_x-e} - n_y n_z 2^{e_y+e_z-e}, e)$. Du lemme 12, je déduis que $r = (n_x - n_y n_z 2^{e_y+e_z-e_x}, e_x)$.

Pour prouver sous les hypothèses données que r est représentable, il suffit de prouver que $|n_r| < 2^n$ avec $n_r = (x - yz)2^{-e_x}$. Je sépare alors en 2 sous-cas :

– si $2^{k-1+e_y} \leq \left|\frac{x}{z}\right|$, alors j'utilise le lemme 14.

$$\begin{aligned} |n_r| &= 2^{-e_x}|x-yz| = 2^{-e_x} \times |z| \times |\frac{x}{z} - y| \\ &\leq 2^{-e_x} \times (2^n - 1)\, 2^{e_z} \times 2^{e_y-1} \\ &< 2^n \times 2^{-e_x+e_z+e_y-1} \leq 2^n \end{aligned}$$

– sinon, j'ai $\left|\frac{x}{z}\right| < 2^{k-1+e_y}$.
Comme y est un arrondi de $\frac{x}{z}$ et que $|y| \geq 2^{k-1+e_y}$, cela signifie que $|y| = 2^{k-1+e_y}$. Donc y est une puissance de 2 et l'arrondi est dirigé vers le haut. Cela signifie donc que $\left|y - \frac{x}{z}\right| \leq \frac{1}{4}\text{ulp}(y) = 2^{e_y-2}$ puisque $\frac{x}{z}$ est plus proche de y que de son prédécesseur (qui est à distance $\frac{1}{2}\text{ulp}(y)$ de y). Et donc, j'utilise le lemme 13.

$$\begin{aligned} |n_r| &= 2^{-e_x}|x-yz| = 2^{-e_x} \times |z| \times |\frac{x}{z} - y| \\ &\leq 2^{-e_x} \times (2^n - 1)\, 2^{e_z} \times 2^{e_y-2} \\ &< 2^n \times 2^{-e_x+e_z+e_y-2} \leq 2^n \end{aligned}$$

Dans tous les cas, le r choisi est représentable sur n bits.

4.5 Calcul de l'ulp de l'inverse

Ce problème nous fut soumis par P. Markstein en juillet 2002. Il concerne l'évaluation exacte de ulp $\left(\frac{1}{f}\right)$ connaissant ulp(f). On se place ici en base β quelconque et dans un format (β^p, E_i). On utilise un mode d'arrondi quelconque.

Si le nombre flottant étudié f n'est pas trop petit, ni trop grand, l'ulp de l'inverse de f est calculable. Il faut séparer deux cas bien distincts selon que f est une puissance de la base.

Si f est une puissance de la base et qu'il n'y a pas de dépassement de capacité, alors le calcul de l'inverse est exact et il est facile d'en calculer l'ulp.

Lemme 15 (`FulpRinv_div` provenant de `FIA64elem`)
Soit f un nombre flottant représentable tel que $|f| =_{\mathbb{R}} \beta^n$ avec $-E_i+p-1 \leq n \leq E_i+1-p$. Alors,

$$\mathrm{ulp}(f) \times \mathrm{ulp}(1 \boxslash f) = \beta^{2-2p}.$$

Si f n'est pas une puissance de la base, je peux borner suffisamment précisément $1/f$ pour pouvoir avoir une idée précise de $1 \boxslash f$ et en connaître l'ulp.

Lemme 16 (`FulpRinv_div_not` provenant de `FIA64elem`)
Soit f un nombre flottant représentable qui n'est pas une puissance de la base et tel que $e_f \leq E_i - 1 + 2p$. Alors,

$$\mathrm{ulp}(f) \times \mathrm{ulp}(1 \boxslash f) = \beta^{1-2p}.$$

Ce sont des points-clés de la démonstration de la correction de la division implantée dans la bibliothèque mathématique de l'IA-64 décrite dans [Mar00]. Plus précisément, ces résultats vérifient et étendent les théorèmes 6.8 et 6.9 pages 92-93.

4.6 Conclusion

Ces quelques exemples de propriétés connues ou nouvelles valident l'intérêt de notre modèle. Il est en effet faisable, voire assez aisé dans les cas précédents, de montrer des propriétés d'exactitude. Prouver que tel nombre flottant est représentable ou que tel calcul est exact est facilement exprimable. De telles preuves peuvent devenir bien plus complexes, comme celles du chapitre 10 mais les énoncés restent simples et la technique de preuve reste la même : il faut exhiber le nombre flottant adapté. Cette démarche est relativement non-intuitive en arithmétique des ordinateurs, mais est assez facile à utiliser et très efficace. D'autres types de preuves, du type évaluation et majoration d'erreurs, sont explorées dans les chapitres 8 et 9. Les énoncés sont alors plus compliqués et bien moins intuitifs.

Les preuves des 2 chapitres précédents ont été détaillées car la façon de les faire est originale et assez inhabituelle. C'est pourquoi j'ai voulu les expliciter de façon à faire comprendre leur principe. À partir de maintenant, les preuves ne sont plus aussi détaillées, seule l'idée et les arguments principaux des preuves sont donnés afin de limiter la taille et d'accroître la lisibilité de ce qui suit.

Avant cela, j'ai voulu toucher aux limites de notre modèle. Le but est de le modifier légèrement ou de l'enrichir et de voir les conséquences de ces ajustements. Le chapitre 6 montre la modification du modèle pour l'adapter notamment aux notations en complément à la base. Les chapitres 5 et 7 montrent des ajouts à la formalisation. Le premier détaille des essais pour faire coller notre formalisation à la norme IEEE-754 et le second ajoute un arrondi moins puissant appelé fidèle que j'utilise ensuite dans les résultats du chapitre 9.

Deuxième partie

Extensions du modèle

Chapitre 5

Processeurs génériques et norme IEEE

MATLAB's creator Cleve Moler used to advise foreign visitors not to miss the country's two most awesome spectacles: the Grand Canyon, and meetings of IEEE p754.
Michael L. Overton, Numerical Computing with IEEE Floating Point Arithmetic (2001)

Notre formalisation est assez éloignée des nombres flottants machine réellement utilisés par les processeurs. Nous avons donc voulu nous en approcher afin d'être sûrs de la validité de nos résultats dans les applications réelles. Ce chapitre décrit donc une formalisation plus matérielle des nombres flottants et ses liens avec la formalisation initiale présentée au chapitre 2.

Notre formalisation de haut niveau est pratique à utiliser mais ne convient pas à la vision matérielle de vecteurs de bits qu'impose la norme IEEE-754. J'ai donc également formalisé la représentation matérielle réelle en base 2 des nombres flottants dans le système Coq et montré l'équivalence entre les deux formalisations.

Une formalisation en Coq de la norme IEEE-754 avait été faite en 1997 par Loiseleur [Loi97]. Je ne l'ai pas reprise car je souhaite être aussi générique que possible de façon à couvrir la plupart des formats, des plus réduits (moins précis) aux plus étendus (plus précis) alors que seuls 5 formats sont définis par Loiseleur avec des nombres de bits fixés. Je veux également utiliser les nombres réels de Coq pour leurs automatisations et pour la compatibilité avec la formalisation initiale, mais les flottants IEEE de [Loi97] prennent leur valeurs dans un type dyadique peu utilisable. Enfin, les vecteurs de bits utilisés sont du type `register` que Loiseleur a défini, qui sont plus difficiles à utiliser et disposent de bien moins de résultats que la bibliothèque `Bvector` de Jean Duprat qui est bien plus récente.

Une formalisation de bas niveau est d'une part le seul moyen de pouvoir comparer mes résultats avec ceux des formalisations qui se basent sur les vecteurs de bits. Celles-ci représentent la quasi-totalité des formalisations disponibles [Bar89, Car95, Min95, Rus98, Jac01] (voir les paragraphes 1.3 et 2.7).

C'est d'autre part une démonstration de la validité des autres résultats : les nombres flottants machines correspondent aux nombres flottants IEEE de la formalisation présentée dans ce chapitre, qui correspondent eux-mêmes aux nombres flottants de la formalisation présentée au chapitre 2. Je ne souhaite pas montrer l'emboîtement de nombreuses formalisations, mais juste que celle que j'utilise est en accord avec à la réalité.

C'est enfin le meilleur moyen d'énoncer des théorèmes de haut niveau (un flottant est un arrondi au plus proche, un flottant est représentable...) sur un algorithme de bas niveau, basé sur la représentation binaire des flottants utilisés. Je pense en particulier au résultat de Coonen de 1978 [Coo78] qui garantit l'arrondi correct des opérations de base ($+$, $-$, $*$, $/$, $\sqrt{}$) en utilisant 3 bits supplémentaires.

Je présente tout d'abord plus précisément les nombres flottants définis par la norme IEEE-754 et ma formalisation de ceux-ci et enfin les liens entre formalisations. Je présente aussi de véritables arrondis qui sont des fonctions et non plus des propriétés.

5.1 Les nombres flottants machine

5.1.1 Définitions

Un nombre flottant machine de la norme IEEE-754 est un vecteur de bits composé de trois champs, le signe s, l'exposant e et la fraction f comme décrit dans la figure 5.1.

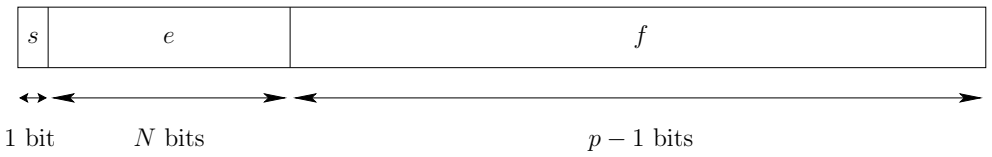


Fig. 5.1 – Représentation interne d'un nombre flottant machine.

Un format flottant est composé des nombres de bits alloués pour l'exposant et la fraction. La norme définit ainsi les formats suivants :

- le `single` ou simple précision sur 32 bits : 8 pour l'exposant et 23 pour la fraction ;
- le `double` ou double précision sur 64 bits : 11 pour l'exposant et 52 pour la faction ;
- le `double-extended` ou précision double étendue sur au moins 80 bits : au moins 15 pour l'exposant et au moins 63 pour la fraction (avec le 1 explicite, voir les exemples ci-dessous).

À partir de ces données, je déduis la précision p : le nombre de bits alloué à la fraction auquel est ajouté 1 ($p = 24$ en simple précision et $p = 53$ en double) qui est bien sûr la même précision que celle utilisée par notre formalisation.

Je note N le nombre de bits alloués pour l'exposant ($N = 8$ en simple précision et $N = 11$ en double). L'exposant stocké est positif, c'est pour cela que l'exposant réel du nombre flottant machine est obtenu après l'ajout d'un biais $B = 2^{N-1} - 1$, dont la valeur est également le plus grand exposant biaisé autorisé.

5.1.2 Valeurs spéciales

La norme IEEE-754 définit 3 types de valeurs spéciales (caractérisées par des valeurs réservées de l'exposant) :

- *NaN*, ou Not-A-Number : c'est le résultat d'une opération absurde telle que $\sqrt{-1}$ ou $0/0$ ou $+\infty-\infty$. Un *NaN* signifie qu'aucune valeur ou n'importe quelle valeur est acceptable comme résultat pour cette opération.
- $\pm\infty$: ce sont les résultats possibles d'une opération dont le résultat correct est trop grand (en valeur absolue) pour être représenté.
- ± 0 : ces deux nombres flottants correspondent au zéro habituel de $\mathbb{R}$. On a en plus une information de signe qui permet de savoir quel infini répondre lorsque l'on considère par exemple l'opération $1/0$. Les deux nombres flottants correspondent à des vecteurs de bits différents mais doivent être considérés comme égaux arithmétiquement.

La norme IEEE-754 régit la façon dont toutes ces valeurs spéciales se comportent lors des opérations. Ces obligations sont raisonnables : si l'un des opérandes est un *NaN*, le résultat est un *NaN* et les infinis se comportent comme des infinis mathématiques, ainsi $3+\infty = +\infty$. Quant aux zéros, la norme définit exactement quel zéro doit être retourné selon l'opération et le mode d'arrondi : par exemple $x - x$ renvoie -0 en arrondi vers $-\infty$ et $+0$ dans tous les autres modes d'arrondi.

5.1.3 Interprétation

Cette suite de bits doit être ensuite comprise soit comme un nombre, c'est-à-dire comme une valeur réelle, soit comme une valeur spéciale. On considère un nombre flottant machine composé de son signe s, son exposant e (non biaisé) et sa fraction f. Il appartient à un format comme défini plus haut (N bits d'exposant et $p-1$ de fraction).

On note $1 \bullet f$ (respectivement $0 \bullet f$) la valeur du nombre à virgule fixe $1{,}b_1b_2\dots b_{p-1}$ (respectivement $0{,}b_1b_2\dots b_{p-1}$) soit

$$1 \bullet f = 1 + \sum_{i=1}^{p-1} b_i 2^{-i} \qquad\qquad 0 \bullet f = \sum_{i=1}^{p-1} b_i 2^{-i}.$$

Ce 1 ou ce 0 ne sont en pratique pas stockés dans le nombre flottant de façon à gagner un bit. Ce 1 est alors appelé implicite. Lorsqu'il fait systématiquement partie de la suite de bits, il est dit explicite.

Selon la valeur de e et de f, le tableau de la figure 5.2 décrit l'interprétation des nombres flottants machine. Il ajoute aussi suivant les cas la façon dont de tels nombres sont appelés.

Exposant	Fraction	Valeur	Qualificatif
$e = 2^N - 1$	$f \neq 0$	*NaN*	Not-a-Number
$e = 2^N - 1$	$f = 0$	$(-1)^s\infty$	Infini
$0 < e < 2^N - 1$		$(-1)^s \times 2^{e-B} \times (1 \bullet f)$	Normalisé
$e = 0$	$f \neq 0$	$(-1)^s \times 2^{1-B} \times (0 \bullet f)$	Dénormalisé
$e = 0$	$f = 0$	$(-1)^s 0$	Zéro

FIG. 5.2 – Interprétation des nombres flottants machines en valeurs réelles ou spéciales.

5.2 Les nombres flottants IEEE et fonctions associées

Il a d'abord fallu définir ce qu'était un nombre flottant IEEE en Coq. Pour cela, j'ai utilisé les vecteurs de bits ➢ Bvector de la bibliothèque standard de Coq. Je note $\overline{v}$ un vecteur ➢ vector bool taille. La taille du vecteur est toujours connue implicitement dans la notation mathématique mais doit toujours être explicitée dans le code Coq. Je note $val(\overline{v})$ la valeur entière correspondant ➢ binary_value v au vecteur $\overline{v}$.

Un format IEEE ➢ IEEE_precision est un enregistrement ➢ Record composé de deux entiers positifs ➢ nat, le premier étant la précision p définie ci-dessus ➢ precision et le second est le nombre de bits alloués pour l'exposant N défini ci-dessus ➢ nbexp.

Un nombre flottant IEEE ➢ IEEE_Float est un enregistrement ➢ Record composé d'un booléen ➢ bool correspondant au signe ➢ Sign du nombre flottant machine, d'un vecteur de bits de taille $p-1$ ➢ vector bool (pred (precision p)) correspondant à la fraction ➢ IEEE_Frac du nombre flottant machine et enfin d'un vecteur de bits de taille N ➢ vector bool (nbexp p) correspondant à l'exposant non biaisé ➢ IEEE_Exp du nombre flottant machine.

Énoncé Coq
```
Record IEEE_precision : Set := Precision {
  precision : nat;
  nbexp     : nat}.

Record IEEE_Float : Set := IEEE_construct {
   Sign      : bool;
   IEEE_Frac : vector bool (pred (precision p));
   IEEE_Exp  : vector bool (nbexp p)}.
```

Une partie des fonctions décrites ensuite sont reprises dans la figure 5.3 qui indique leurs types d'entrée et de sortie.

5.2.1 Des nombres flottants IEEE aux réels

J'utilise en plus du biais normal $B = 2^{N-1} - 1$ défini par la norme un biais correspondant au fait que l'évaluation d'un vecteur de bits est un entier et pas un nombre à virgule fixe $(1 \bullet f)$ et que cette transformation s'est faite en multipliant/divisant par la valeur 2^{p-1}.

Je définis la fonction ItoR ➢ IEEE_To_R des nombres flottants IEEE ➢ IEEE_Float vers $\mathbb{R}$. Si l'exposant du nombre flottant IEEE est zéro, alors je calcule la valeur dans le cas dénormalisé et sinon, je calcule la valeur dans le cas normalisé. Je considère le nombre flottant IEEE $(s, \overline{m}, \overline{e})$, où s est un booléen et où $\overline{m}$ et $\overline{e}$ sont des vecteurs de bits de tailles respectives $p-1$ et N. La valeur réelle associée à x est décrite selon la valeur de s et de $\overline{e}$ dans le tableau de la figure 5.4.

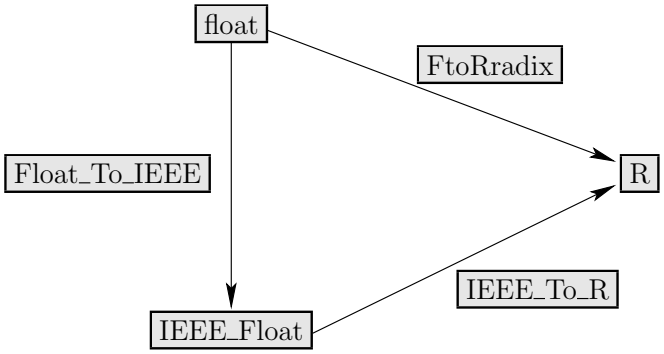


FIG. 5.3 – Diagramme paires/nombres flottants IEEE/Réels.

s	$val(\overline{e})$	valeur
false	$\neq 0$	$\left(2^{p-1} + val(\overline{m})\right) 2^{val(\overline{e})-B-p+1}$
true	$\neq 0$	$-\left(2^{p-1} + val(\overline{m})\right) 2^{val(\overline{e})-B-p+1}$
false	0	$(val(\overline{m}))\, 2^{2-B-p}$
true	0	$-(val(\overline{m}))\, 2^{2-B-p}$

FIG. 5.4 – Différents cas pour le passage des nombres flottants IEEE aux réels.

Mais je ne dois considérer cette fonction que comme une fonction partielle : si l'exposant vaut la valeur maximale, alors la valeur « théorique » du nombre flottant IEEE est soit NaN soit $\pm\infty$, et aucune valeur de $\mathbb{R}$ ne peut représenter cela. J'ai donc choisi de faire une fonction qui convient sur une partie des entrées mais renvoie une valeur réelle utilisable

5.2.2 Des paires aux nombres flottants IEEE

Il s'agit maintenant d'écrire une fonction bijective FtoR ➢ Float_To_IEEE transformant les paires en nombres flottants IEEE. Comme précédemment, j'utilise en plus du biais normal $B = 2^{N-1} - 1$ défini par la norme le biais mantisse entière/virgule fixe de $p - 1$.

J'ai utilisé la fonction $\overline{z}^{\ell}$ ➢ Z_to_binary qui transforme un entier z en un vecteur binaire de taille ℓ le représentant. Bien sûr, cette fonction donne un résultat correct lorsque le nombre à représenter est à la fois positif et plus petit que $2^{\ell} - 1$.

Soit f une paire (n_f, e_f) et soit $\tilde{f} = (n_{\tilde{f}}, e_{\tilde{f}})$ la paire canonique associée à f. La valeur des différents champs du nombre flottant IEEE associé est décrite selon les cas (normalisé ou pas et positif ou pas) dans le tableau de la figure 5.5 et est implantée par la fonction FtoI ➢ Float_To_IEEE.

Cette fonction renvoie toujours $+0$ lorsque la paire est un des zéros possibles. Il n'y a aucun moyen de renvoyer -0 vu que l'ensemble de départ ne contient qu'un seul zéro.

$\tilde{f}$ est	$\tilde{f}$ est tel que	signe	exposant	fraction
normalisé	$n_{\tilde{f}} > 0$	false	$\overline{e_{\tilde{f}} + B + p - 1}^{N}$	$\overline{n_{\tilde{f}} - 2^{p-1}}^{p-1}$
normalisé	$n_{\tilde{f}} < 0$	true	$\overline{e_{\tilde{f}} + B + p - 1}^{N}$	$\overline{2^{p-1} - n_{\tilde{f}}}^{p-1}$
dénormalisé	$n_{\tilde{f}} \geq 0$	false	$\overline{0}^{N}$	$\overline{n_{\tilde{f}}}^{p-1}$
dénormalisé	$n_{\tilde{f}} < 0$	true	$\overline{0}^{N}$	$\overline{-n_{\tilde{f}}}^{p-1}$

FIG. 5.5 – Différents cas pour le passage des paires aux nombres flottants IEEE.

5.2.3 Des nombres flottants IEEE aux paires

De façon symétrique, j'ai défini une fonction qui transforme un nombre flottant IEEE $x = (s, \overline{m}, \overline{e})$ en une paire. La valeur des différents champs de la paire associée est décrite selon la valeur de s et de $\overline{e}$ dans le tableau de la figure 5.6 et est implantée par la fonction ItoF ➢ IEEE_To_Float.

s	$val(\overline{e})$	mantisse	exposant
false	0	$val(\overline{m})$	$2 - B - p$
true	0	$-val(\overline{m})$	$2 - B - p$
false	$\neq 0$	$2^{p-1} + val(\overline{m})$	$val(\overline{e}) - B - p + 1$
true	$\neq 0$	$-2^{p-1} - val(\overline{m})$	$val(\overline{e}) - B - p + 1$

FIG. 5.6 – Différents cas pour le passage des nombres flottants IEEE aux paires.

5.3 Liens entre les nombres flottants

5.3.1 Hypothèses

Je suppose désormais que $p \geq 2$ et $N \geq 1$, ce qui est raisonnable pour un vrai format de nombres flottants.

Je ne peux pas considérer n'importe quelles paires. En effet, pour que je puisse les transformer en nombres flottants IEEE, elles doivent être représentables dans le format de donnée IEEE fourni. Cela signifie que la mantisse et l'exposant doivent être dans certains intervalles d'entiers. Pour permettre la compatibilité avec le reste de la bibliothèque, j'ai décidé de définir cela par l'intermédiaire de la propriété de représentabilité des paires Coq ➢ Fbounded. Je définis donc le format flottant comme $\mathcal{B} = (p, B + p - 2)$ qui correspond aux nombres flottants représentables dans mon format IEEE.

En effet, le plus petit nombre flottant strictement positif est 2^{-E_i}, mais est également le nombre machine dénormalisé dont l'exposant non biaisé est zéro et dont la mantisse est composée de zéros sauf le dernier bit à 1. Sa valeur est alors $2^{1-B} 0 \bullet 0 \cdots 1 = 2^{1-B} 2^{1-p}$, ce

qui explique le choix de $\mathcal{B}$.

Cela ne suffit pas car il faut ajouter une borne maximale pour l'exposant. Je définis alors un nombre flottant f comme restreint ➢ ayant la propriété InDomain par :

$$f \text{ est représentable dans le format}(p, B+p-2) \quad \wedge \quad e_{\tilde{f}} \leq B-p+1$$

où $\tilde{f}$ est le canonique associé à f. Cette propriété sur la paire f borne donc en fait l'exposant du canonique associé à f, qui est l'exposant minimal. Cette propriété est donc la plus faible possible pour garantir que la paire f est représentable dans le format donné.

Elle est équivalente à l'inégalité $|f| \leq \mathit{Maxfloat}$ avec *Maxfloat* le plus grand nombre flottant autorisé, soit $\mathit{Maxfloat} = (2^p - 1, B - p + 1)$. Néanmoins, l'assertion choisie est plus facile à manipuler, ce qui explique le choix fait ici.

5.3.2 Fonction et valeur identique

Je prouve tout d'abord que FtoI ➢ Float_To_IEEE est une fonction, relativement à la relation d'équivalence $=_{\mathbb{R}}$.

Lemme 17 (`Float_To_IEEE_fn` provenant de `IEEE`)
Soient deux paires représentables f et g telles que f et g ont la même valeur réelle, alors ils ont la même représentation IEEE.

Je prouve ensuite que le diagramme 5.3 page 49 commute : considérant une paire, sa valeur réelle est la même que si je la transforme en nombre flottant IEEE et que j'en prend la valeur.

Lemme 18 (`float_equals_IEEE` provenant de `IEEE`)
Soit une paire f restreinte, alors le diagramme 5.3 commute, c'est-à-dire

$$f =_{\mathbb{R}} \text{ItoR}\left(\text{FtoI}(f)\right).$$

5.3.3 Bijection

Pour prouver que FtoI ➢ Float_To_IEEE est une bijection, j'utilise son inverse définie ci-dessus ItoF ➢ IEEE_To_Float et je montre que le diagramme de la figure 5.7 commute.

J'ai tout d'abord eu besoin du lemme suivant.

Lemme 19 (`IEEE_To_Float_Bounded` provenant de `IEEE`)
Quelques soient les variables d'entrée, tout résultat de la fonction ItoF est une paire représentable.

J'ai ensuite voulu prouver que, pour tout nombre flottant IEEE, si je lui applique successivement ItoF ➢ IEEE_To_Float et FtoI ➢ Float_To_IEEE, je retombe sur le nombre

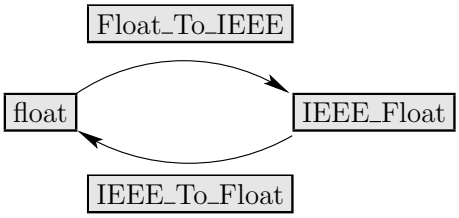


FIG. 5.7 – Diagramme paires/nombres flottants IEEE.

flottant IEEE initial. Or ceci est faux pour un unique nombre flottant IEEE correspondant à -0, c'est-à-dire le nombre flottant IEEE dont le signe est à 1 et dont la fraction et l'exposant sont des tableaux de zéros. En effet, ce nombre flottant IEEE est transformé en une paire nulle, qui est ensuite transformée en $+0$. À part ce cas précis, tous les autres nombres flottants IEEE sont conservés.

Lemme 20 (`IEEE_To_Float_To_IEEE` provenant de `IEEE`)
Soit un nombre flottant IEEE x tel que $x \neq -0$. Alors

$$\mathrm{FtoI}\left(\mathrm{ItoF}(x)\right) = x.$$

L'autre sens n'a pas une telle restriction, puisque le problème du -0 n'existe pas. Par contre, le résultat n'est pas forcément le nombre flottant initial mais le canonique associé.

Lemme 21 (`Float_To_IEEE_To_Float` provenant de `IEEE`)
Soit une paire f restreinte et soit $\tilde{f}$ le canonique associé à f. Alors

$$\mathrm{ItoF}\left(\mathrm{FtoI}(f)\right) = \tilde{f}.$$

5.4 Fonctions d'arrondi

Comme expliqué au chapitre 2, les arrondis sont des propriétés liant un réel et un nombre flottant (étant donnés une base et un format). Ceci permet de considérer *un* arrondi d'un réel plutôt que *l'*arrondi d'un réel. Néanmoins, pour l'extension de la bibliothèque et son utilisation par des non-spécialistes, il était nécessaire de définir de vraies fonctions d'arrondi. Dans cette section, la base est générique.

Ces fonctions sont donc des applications de $\mathbb{R}$ sur les nombres flottants représentables. Elles sont basées sur le fait que $\mathbb{R}$ est un corps archimédien, ce qui signifie qu'il existe une fonction up $\succ$ up de $\mathbb{R}$ dans $\mathbb{Z}$ telle que $\forall r \in \mathbb{R},\ \mathrm{up}(r) > r$ et $\mathrm{up}(r) - r \leq 1$.

Je définis alors la fonction $\lfloor\cdot\rfloor \succ$ IRNDD de $\mathbb{R}$ dans $\mathbb{Z}$ comme $\lfloor r \rfloor = \mathrm{up}(r) - 1$. J'en déduis immédiatement que

Lemme 22 (`IRNDD_correct1` et `IRNDD_correct2` provenant de `RND`)
Pour tout réel r, j'ai

$$\lfloor r \rfloor \leq r < \lfloor r \rfloor + 1.$$

J'ai donc défini une sorte d'arrondi vers le bas d'un réel vers un entier.

Je note F le plus petit nombre flottant normalisé sous sa forme canonique. La valeur de F ne dépend que du format flottant utilisé : $F = (\beta^{p-1}, -E_i) =_{\mathbb{R}} \beta^{p-1-E_i}$. À partir de cette valeur et de la fonction précédente, je peux définir la fonction $\bigtriangledown_p$ ➢ RND_Min_Pos des réels vers les nombres flottants qui correspond à un arrondi vers le bas pour les réels positifs :

$$\bigtriangledown_p(r) = \begin{cases} (\lfloor r \times \beta^{-e} \rfloor, e) & \text{si } r \geq F, \quad \text{avec } e = \left\lfloor \frac{\ln(r)}{\ln(\beta)} + 1 - p \right\rfloor \\ \left(\lfloor r \times \beta^{E_i} \rfloor, -E_i\right) & \text{si } r < F. \end{cases}$$

Pour valider cette fonction dont toutes les autres sont déduites, j'ai prouvé les lemmes suivants :

Lemme 23 (`RND_Min_Pos_canonic` provenant de `RND`)
Pour tout réel r tel que $0 \leq r$, le nombre flottant $\bigtriangledown_p(r)$ est canonique.

Il suffit pour le prouver d'étudier les différents cas et d'en déduire les inégalités utiles.

Lemme 24 (`RND_Min_Pos_correct` provenant de `RND`)
Pour tout réel r tel que $0 \leq r$, le nombre flottant $\bigtriangledown_p(r)$ est un arrondi vers le bas de r.

Pour cela, il suffit de prouver que $\bigtriangledown_p(r) \leq r$, ce qui est assuré par construction. J'ai ensuite prouvé que la fonction $\bigtriangledown_p$ est croissante et enfin qu'elle est un projecteur sur les nombres flottants représentables, c'est-à-dire que si f est un nombre flottant représentable, alors $\bigtriangledown_p(f) =_{\mathbb{R}} f$.

À partir de ces définitions et résultats, il est facile de définir et de valider les fonctions suivantes. Tout d'abord, $\bigtriangleup_p$ ➢ RND_Max_Pos est la fonction d'arrondi vers le haut pour les réels positifs :

$$\bigtriangleup_p(r) = \begin{cases} \bigtriangledown_p(r) & \text{si } r =_{\mathbb{R}} \bigtriangledown_p(r) \\ \bigtriangledown_p(r)^+ & \text{sinon.} \end{cases}$$

où, pour un nombre flottant représentable f, le nombre flottant f^+ est le successeur de f, c'est-à-dire le plus petit nombre flottant représentable strictement plus grand que f (voir également le chapitre 9).

Je peux maintenant définir l'arrondi vers le haut (vers $+\infty$) ➢ RND_Max et vers le bas (vers $-\infty$) ➢ RND_Min de tout réel :

$$\bigtriangledown(r) = \begin{cases} \bigtriangledown_p(r) & \text{si } r \geq 0 \\ -\bigtriangleup_p(-r) & \text{sinon.} \end{cases} \qquad \bigtriangleup(r) = \begin{cases} \bigtriangleup_p(r) & \text{si } r \geq 0 \\ -\bigtriangledown_p(-r) & \text{sinon.} \end{cases}$$

Au vu des définitions et des lemmes 23 et 24, il est facile de montrer que pour tout réel r, $\triangle(r)$ et $\triangledown(r)$ sont des nombres flottants canoniques ainsi que les lemmes suivants.

Lemme 25 (`RND_Min_correct` provenant de `RND`)
Pour tout réel r, le nombre flottant $\triangledown(r)$ est un arrondi vers le bas de r.

Lemme 26 (`RND_Max_correct` provenant de `RND`)
Pour tout réel r, le nombre flottant $\triangle(r)$ est un arrondi vers le haut de r.

Pour l'arrondi au plus proche, plusieurs choix sont possibles lorsque le réel est à mi-distance de deux nombres flottants. La norme IEEE-754 impose de renvoyer celui qui est pair (dont la mantisse finit par un zéro). Je propose ici deux alternatives.

La première possibilité est l'arrondi au plus proche supérieur ➢ RND_ClosestUp, noté $\circ_{up}$ qui est l'arrondi au plus proche où je choisis la valeur la plus grande lors du cas d'égalité. Cet arrondi est défini dans la révision de la norme IEEE-754 [IEE04] car il intéresse au plus haut point les opérateurs téléphoniques. Il était de plus l'arrondi légal du passage à l'euro.

$$\circ_{up}(r) = \begin{cases} \triangle(r) & \text{si } |\triangle(r) - r| \leq |\triangledown(r) - r| \\ \triangledown(r) & \text{sinon.} \end{cases}$$

Il est facile de montrer que pour tout réel r, le nombre flottant $\circ_{up}(r)$ est canonique. De plus,

Lemme 27 (`RND_ClosestUp_correct` provenant de `RND`)
Pour tout réel r, le nombre flottant $\circ_{up}(r)$ est un arrondi au plus proche de r.

La seconde possibilité est l'arrondi au plus proche pair ➢ RND_EvenClosest, noté $\circ_{even}$ qui est l'arrondi défini par la norme IEEE-754 et l'arrondi par défaut.

$$\circ_{even}(r) = \begin{cases} \triangledown(r) & \text{si } |\triangledown(r) - r| < |\triangle(r) - r| \\ \triangledown(r) & \text{si } |\triangledown(r) - r| = |\triangle(r) - r| \text{ et que la mantisse de } \triangledown(r) \text{ est paire} \\ \triangle(r) & \text{dans les autres cas.} \end{cases}$$

Il est aisé de montrer que pour tout réel r, le nombre flottant $\circ_{even}(r)$ est canonique. Par contre, la correction de l'arrondi a été plus difficile :

Lemme 28 (`RND_EvenClosest_correct` provenant de `RND`)
Pour tout réel r, le nombre flottant $\circ_{even}(r)$ est l'arrondi au plus proche pair de r.

la démonstration se base sur le fait que si $\triangledown(r)$ et $\triangle(r)$ sont différents, alors j'ai $\triangle(r) = \triangledown(r)^+$ et si l'un a une mantisse paire, alors l'autre a une mantisse impaire. De ce fait, des lemmes précédents et de l'étude de tous les cas, j'obtiens le résultat.

5.5 Conclusion

Les résultats obtenus concernent l'équivalence entre les valeurs représentées par la norme IEEE-754 et celles de notre formalisation. Ces résultats imposent évidemment que la paire soit représentable avec un format flottant pré-défini, mais aussi que son exposant soit majoré, or les développements excluent ce cas. La raison est que les dépassements de capacité sont facilement détectables : ils créent des infinis et des *NaN* que l'on peut récupérer et identifier en fin de calcul. De plus, une borne supérieure sur l'exposant créerait de nombreuses difficultés techniques : l'étude d'une suite de calcul nécessiterait que chaque calcul soit étudié pour savoir s'il peut ou pas créer un dépassement de capacité. Cela multiplierait le nombre de sous-cas sans intérêt à étudier avant de pouvoir énoncer un théorème. Nous ne sommes d'ailleurs pas les seuls à commencer par ignorer formellement ces cas [Jac01, Jac02].

Les preuves présentées sont longues et calculatoires car il faut étudier chaque cas (normalisé/dénormalisé, positif/négatif) et en tirer à chaque fois les inégalités utiles pour justifier que chaque valeur stockée dans un vecteur loge bien dans ce vecteur de taille donnée et qu'aucune information n'a été perdue pour cause de vecteur trop petit.

À propos des arrondis, une question justifiée est de se demander quelle est la réalité de ces fonctions. Ma réponse est que ces fonctions sont aussi constructives que le sont les réels utilisés. J'ai utilisé les réels de Coq qui sont axiomatisés [May01], je ne peux donc pas toucher vraiment ces réels et leurs valeurs ne sont pas calculables. Ces réels n'étant pas constructifs, mes fonctions d'arrondi ne sont pas constructives : par exemple, il est impossible à Coq de calculer l'arrondi de π sur 3 bits. Une solution pourrait être d'utiliser la bibliothèque des réels constructifs C-CoRN[1] [GPWZ02]. Elle possède en effet la propriété de corps archimédien (axiome archimed dans la bibliothèque standard). À part cet axiome, je n'ai utilisé dans mes autres développements que des propriétés simples des réels (calculs, manipulation d'inégalité...). La bibliothèque pourrait donc utiliser C-CoRN, mais je préfère les nombres réels standards pour des raisons de compatibilité.

Une extension possible de ce travail est de formaliser également la norme IEEE-854 [CK+84], l'équivalent de la norme IEEE-754 en base quelconque. Notre formalisation est générique vis-à-vis de la base mais je ne l'ai pas fait ici pour une raison très simple : je souhaite utiliser des définitions de la bibliothèque standard pour représenter les vecteurs de chiffres de façon à m'assurer une bibliothèque pré-existante de faits prouvés et à garantir la compatibilité avec d'autres projets. La seule possibilité était le type Bvector qui n'est malheureusement qu'un vecteur de bits true/false. Dès qu'un type plus général, avec son lot de théorèmes associés, sera disponible pour représenter les entiers en base β, en représentation usuelle ou redondante, je pourrai envisager d'étendre ce travail.

À la section précédente, je n'ai pas précisé une définition, celle de mantisse paire. Or, selon le choix de cette définition implicite, ce que je viens d'affirmer peut être considéré comme faux en base impaire. En effet, en base 2, avoir une mantisse paire correspond à avoir une mantisse qui finit par un zéro. Mais en base 3, qu'est-ce qu'une mantisse paire ? Les mantisses en base 3 1112 et 1120 se suivent et finissent toutes deux par un chiffre pair. Sont-elles toutes deux paires ? Si non, alors quelle définition cohérente peut-on donner ? Si oui, pour un exposant et un signe donné, si je considère le réel à mi-chemin entre les deux nombres flottants ainsi construits, quel est l'arrondi pair de ce réel ?

[1] C-CoRN est le sigle de « Constructive Coq Repository at Nijmegen ».
Voir `http://vacuumcleaner.cs.kun.nl/c-corn/`.

En fait, nous n'avons pas eu ce problème avec notre formalisation, mais probablement par un heureux hasard : une mantisse n'est pas un tableau de chiffres, mais un entier relatif. Or un entier relatif est toujours pair ou impair, et cela indépendamment de sa représentation dans la base utilisée. Le choix fait ici est donc du nombre flottant dont l'interprétation de la mantisse comme un entier est paire. Ce choix est arbitraire et ne dépend pas de la base utilisée pour effectivement représenter la mantisse, mais il a le mérite d'exister et d'être cohérent. Cette définition choisie par L. Théry lui avait permis de prouver l'existence et l'unicité de l'arrondi au plus proche pair, ce qui aurait été impossible en cas de définition ambiguë ou incohérente. Ce choix fait que dans notre exemple, il faut choisir $\boxed{1120}$: la réponse est 42.

Chapitre 6

Autres systèmes – Complément à 2

Planes that fly and bridges that stand are impressive evidence of belief in theorems.
Richard A. DeMillo, Richard J. Lipton et Alan J. Perlis [DLP77]

La formalisation utilisée est uniquement basée sur une représentation symétrique des mantisses conforme à la norme IEEE-754. Pourtant, nous avons voulu tester une extension à des systèmes plus génériques où la mantisse n'est plus symétrique. Les conséquences ont été rapides et concluantes, montrant des résultats surprenants et même quelques extensions utiles au système de départ.

La plupart des processeurs existants ne sont pas des processeurs génériques, comme ceux des ordinateurs, et compatibles avec la norme IEEE-754, ce sont des processeurs dédiés faisant fonctionner les téléphones portables, les machines à laver, les lecteurs de DVD mais aussi les voitures et les avions en tant que contrôleurs de vol [LMW01]. Leur unité flottante (quand elle existe) est rarement compatible IEEE-754 et utilise quelquefois une notation en complément à 2 (voir §6.1) comme le fait le TMS 320 C3x [Tex97] effectivement utilisé en avionique.

J'ai généralisé notre formalisation de façon à prendre en compte ces processeurs. Après avoir relâché la notion de nombre flottant représentable, je recouvre beaucoup de formats flottants, bien plus que ceux raisonnablement envisageables. Mais dès que j'ai imposé quelques conditions basiques sur les nombres flottants obtenus, je me suis rendue compte que cela oblige tant de restrictions sur le format flottant que mon type générique s'est réduit comme peau de chagrin [Bal31] : il ne reste que les formats du type IEEE-754 de notre formalisation initiale et les formats du type complément à 2, soit les deux formats effectivement utilisés. J'ai de plus prouvé quelques propriétés bien connues en format symétrique mais pas forcément évidentes en complément à 2.

6.1 Représentations des entiers

La représentation directe en base 2 sur les chiffres $\{0, 1\}$ (non redondante) s'est imposée pour représenter les entiers non signés. Par contre, dès qu'il faut représenter dans le

même formalisme des entiers positifs et négatifs, plusieurs choix sont possibles : les systèmes non redondants contiennent la représentation signe-valeur absolue, le complément à 1 et le complément à la base [Dau01] ; les systèmes redondants se retrouvent généralement dans le formalisme d'Avižienis [Avi61] : « borrow save » et « carry save » par exemple.

La représentation en complément à 2 est très souvent préférée par les concepteurs de circuits pour les entiers signés, mais la question reste ouverte pour l'arithmétique à virgule flottante dans le cadre de processeurs dédiés. La représentation signe-valeur absolue correspond aux *desiderata* de la norme IEEE-754 et représente un entier signé par un bit s de signe et une valeur absolue n, le nombre valant alors

$$(-1)^s \times n.$$

La représentation en complément à 2 représente un entier signé par un bit s accolé à un vecteur de $p-1$ bits (de valeur n), le nombre valant alors n si $s = 0$ et $-2^p + n$ sinon, c'est-à-dire dans tous les cas

$$-s \times 2^p + n.$$

Pour de plus amples considérations sur le complément à la base en base quelconque, voir [DM97a, Dau01].

6.2 Représentation des nombres flottants

6.2.1 Représentabilité

J'utilise les nombres flottants définis au chapitre 2. Par contre, je modifie la définition de « nombre flottant représentable » comme suit :

Énoncé Coq
```
Record FboundI : Set := BoundI {
   vNumInf : nat;
   vNumSup : nat;
   dExpI   : nat}.

Definition FboundedI (b : FboundI) (d : float) :=
  (- vNumInf b <= Fnum d)%Z
    /\ (Fnum d <= vNumSup b)%Z
    /\ (- dExpI b <= Fexp d)%Z.
```

ce qui signifie que les formats flottants sont désormais définis par un triplet de valeurs entières noté $\mathbb{B} = (N_i, N_s, E_i)$ et la définition de représentable devient :

$$(n, e) \text{ représentable} \quad \Leftrightarrow \quad n \in \{-N_i, \cdots, N_s\} \quad \text{et} \quad e \geq -E_i.$$

Si $N_i = N_s = \beta^p - 1$, je retrouve exactement la formalisation du chapitre 2. J'ai donc bien sûr comme précédemment une cohorte de nombres flottants possibles correspondant à certaines valeurs réelles.

6.2.2 Représentations multiples

Je définis alors comme au chapitre 2 un représentant canonique unique correspondant à chaque cohorte. J'appelle (dans ce chapitre uniquement) nombre flottant normalisé un nombre flottant représentable (selon la définition de ce chapitre) dont l'exposant ne peut être réduit en modifiant la mantisse, c'est-à-dire tel que

$$n \times \beta \notin \{-N_i, \cdots, N_s\}.$$

J'appelle (dans ce chapitre uniquement) nombre flottant dénormalisé un nombre flottant représentable (selon la définition de ce chapitre) dont l'exposant est le minimum possible et dont la mantisse pourrait être multipliée par la base tout en restant dans l'intervalle autorisé, c'est-à-dire tel que

$$n \times \beta \in \{-N_i, \cdots, N_s\} \qquad \text{et} \qquad e = -E_i.$$

Un nombre flottant canonique est alors soit un nombre flottant normalisé, soit un nombre flottant dénormalisé. J'ai ensuite prouvé les propriétés suivantes associées à ces définitions :

Lemme 29 (`FcanonicIUnique` provenant de `FboundI`)
Soient f et g deux nombres flottants canoniques tels que $f =_{\mathbb{R}} g$, alors $f = g$.

J'ai ensuite pu définir la fonction FnormalizeI qui décrit un algorithme pour transformer un nombre flottant représentable en flottant canonique. Sa définition étant longue et assez compliquée, je ne l'ai pas retranscrite ici mais elle se trouve dans le fichier `FboundI.v`. Ses propriétés ont été validées par les lemmes suivants où je note $\mathcal{N}$ cette fonction (la base et le format utilisés sont implicites) :

Lemme 30 (`FnormalizeICorrect` provenant de `FboundI`)
Soit f un nombre flottant représentable, alors $\mathcal{N}(f) =_{\mathbb{R}} f$.

Lemme 31 (`FnormalizeIFcanonicI` provenant de `FboundI`)
Soit f un nombre flottant représentable, alors $\mathcal{N}(f)$ est canonique.

Nous nous sommes ensuite intéressés à un certain nombre de propriétés de base que l'on peut attendre d'un système flottant.

6.3 Opposé d'un nombre flottant

Un nombre flottant représentable f admet un opposé s'il existe un nombre flottant g représentable tel que $g =_{\mathbb{R}} -f$. Cela peut sembler naturel pour tout système raisonnable mais en fait cela limite grandement les systèmes autorisés.

6.3.1 Garantir que tout nombre flottant admette un opposé

Sur les systèmes du type IEEE-754 où $N_i = N_s$, tout nombre flottant admet un opposé : il suffit de considérer le nombre flottant dont la mantisse est opposée, il est également représentable et sa valeur est bien l'opposée de la valeur initiale.

Sur les autres systèmes, nous pouvons également répondre à cette question :

Théorème 32 (`FoppBoundedI` provenant de `FnormI`)
Si le format est tel qu'il existe un entier m tel que $N_i = \beta \times m$ et $N_s = \beta \times m - 1$, alors tout nombre flottant représentable admet un opposé.

Énoncé Coq
```
Theorem FoppBoundedI :
 forall (b : FboundI) (x : float) (m : nat),
 Z_of_nat (vNumSup b) = (radix * m - 1%nat)%Z ->
 Z_of_nat (vNumInf b) = (radix * m)%Z ->
 FboundedI b x ->
 exists y : float,
   FtoRradix y = (- x)%R /\ FboundedI b y.
```

Théorème 33 (`FoppBoundedIInv` provenant de `FnormI`)
Si le format est tel que $N_s < N_i$ et que tout nombre flottant représentable admet un opposé, alors il existe un entier m tel que $N_i = \beta \times m$ et $N_s = \beta \times m - 1$.

Énoncé Coq
```
Theorem FoppBoundedIInv :
 (vNumSup b < vNumInf b)%Z ->
 (forall x : float,  FboundedI b x ->
     exists y : float, FtoRradix y = (- x)%R /\ FboundedI b y) ->
 (exists m : Z,
   Z_of_nat (vNumInf b) = (radix * m)%Z) /\
   Z_of_nat (vNumInf b) = (vNumSup b + 1%nat)%Z.
```

En effet, on a alors que tout nombre flottant (n, e) avec $n \in \{-N_s, \cdots, N_s\}$ admet un opposé en manipulant le signe de la mantisse. Mais les nombres flottants tels que $n \in \{N_s + 1, \cdots, N_i\}$ admettent un opposé seulement si on peut manipuler leur exposant. Donc β doit diviser tous les $n \in \{N_s + 1, \cdots, N_i\}$ ce qui n'est possible que si $N_s + 1 = N_i$ et β divise N_i.

Par symétrie, si $N_i \neq N_s$, on a que « tout nombre flottant représentable admet un opposé » est équivalent à

$$|N_i - N_s| = 1 \qquad \text{et} \qquad \beta \mid \max(N_i, N_s).$$

Cela signifie qu'il faut se restreindre à une notation du type complément à deux ou du type signe-magnitude si l'on veut que tout nombre flottant admette un opposé !

6.3.2 Opposé et dépassements de capacité

En fait, en notation en complément à deux comme celle du TMS 320 C3x, on a $N_i = 2^p$ et $N_s = 2^p - 1$. De plus, seuls les normalisés sont implantés, tout nombre trop petit est arrondi à zéro.

Il faut se souvenir que nous ne traitons pas les dépassements de capacité supérieurs en Coq. En particulier dans le cas du TMS 320 C3x, deux nombres flottants n'admettent pas d'opposé :

- le plus petit nombre représentable (*i.e.* le négatif à la valeur absolue la plus grande) $(-N_i, E_{max})$ a un opposé plus grand que le plus grand nombre et l'opposer crée donc un dépassement de capacité ;
- le plus petit nombre représentable strictement positif $(2^{p-1}, -E_i)$ a un opposé qui n'est pas normalisé et n'est donc pas représentable par le TMS 320 C3x. Ce cas n'arrive pas sur les systèmes qui pourraient représenter les dénormalisés.

Le lemme suivant garantit que seuls les cas précédents peuvent poser problème :

Lemme 34 (`FoppBoundedIExp` provenant de `FnormI`)
Si le format est tel qu'il existe un entier m tel que $N_i = \beta \times m$ et $N_s = \beta \times m - 1$, alors tout nombre flottant représentable f admet un opposé g tel que

$$e_g = e_f \qquad \text{ou} \qquad e_g = e_f + 1.$$

6.3.3 Opposé et négation

La négation est le seul opposé signifie que si $x \boxplus y = 0$, alors y est un opposé de x. J'ai prouvé la validité de ce fait sur un système représentant les dénormalisés. Je reprends la preuve de Kulisch [Kul00] : si deux nombres flottants sont différents, alors leur distance est d'au moins β^{-E_i} et donc l'arrondi de leur somme n'est pas nul.

Lemme 35 (`OppositeIUnique` provenant de `FnormI`)
Soient x et y deux nombres flottants représentables. Si $y \neq_{\mathbb{R}} -x$ et $z = x \boxplus y$ selon un arrondi quelconque, alors

$$|z| \geq \beta^{-E_i}.$$

6.4 Ordre lexicographique

De nombreux auteurs dont Coonen [Coo78] ont présenté qu'il est souhaitable que l'ordre lexicographique des nombres flottants coïncide avec l'ordre des valeurs réelles représentées, du moins sur les positifs. Ce fait n'est pas évident pour un système quelconque comme je l'ai défini, j'ai établi le lemme suivant quelles que soient les valeurs de N_i, N_s et E_i :

Lemme 36 (`LexicoPosCanI` provenant de `FnormI`)
Soit $x = (n_x, e_x)$ un nombre flottant canonique et soit $y = (n_y, e_y)$ un nombre flottant représentable. Alors

$$0 \leq x \leq y \qquad \text{implique} \qquad e_x \leq e_y.$$

Par contre, pour pouvoir comparer positifs et négatifs, il faut absolument ajouter une condition supplémentaire, mais celle-ci est moins forte que celle définie au paragraphe 6.3.1 :

Lemme 37 (`LexicoCanI` provenant de `FnormI`)
On suppose que $|N_i - N_s| \leq 1$.
Soit $x = (n_x, e_x)$ un nombre flottant canonique et soit $y = (n_y, e_y)$ un nombre flottant représentable. Alors

$$|x| < |y| \qquad \text{implique} \qquad e_x \leq e_y.$$

Le résultat équivalent en système du type IEEE-754 est le théorème Fcanonic_Rle_Zle qui dit que pour deux canoniques x et y, alors $|x| \leq |y|$ implique $e_x \leq e_y$. La différence entre les deux résultats est due au fait que pour une même valeur absolue, les mantisses d'un canonique positif ou négatif peuvent être différentes.

La contraposée du résultat précédent est également intéressante :
On suppose que $|N_i - N_s| \leq 1$. Soit $x = (n_x, e_x)$ un nombre flottant canonique et soit $y = (n_y, e_y)$ un nombre flottant représentable. Alors

$$e_x < e_y \qquad \text{implique} \qquad |x| \leq |y|.$$

Donc à condition que $|N_i - N_s| \leq 1$, notre système flottant se comporte presque comme un système IEEE au point de vue de l'ordre lexicographique. Une condition comme précédemment du type $\beta \mid \max(N_i, N_s)$ est ici inutile.

Par contre, dès que $|N_i - N_s| > 1$, le comportement du système est bien différent. Voici un exemple justifiant également que la borne sur $|N_i - N_s|$ est la meilleure possible. Je considère le système en base 2 où les mantisses doivent être comprises entre -1001_2 et 111_2. Les nombres flottants $(100_2, 1)_2$ et $(-1001_2, 0)_2$ sont canoniques, mais leurs valeurs absolues et leurs exposants sont en ordre inverse. Heureusement, ce genre de comportement aberrant ne peut arriver ni dans les systèmes du type IEEE-754, ni dans les systèmes du type complément à 2.

6.5 Ensemble représentable

Les lemmes suivants montrent que l'ensemble représenté dans un processeur du type TMS 320 C3x où la mantisse est en complément à 2 est quasi-identique à celui représenté dans un processeur compatible IEEE-754.

Lemme 38 (`ReductRangeI` provenant de `FnormI`)
Si $N_s > 1$ et $\beta \mid N_s$, tout nombre flottant représentable dans le format (N_i, N_s, E_i) a un nombre flottant équivalent (en valeur réelle) représentable dans le format $(N_i, N_s - 1, E_i)$.

Lemme 39 (`ReductRangeIInv` provenant de `FnormI`)
Si $N_s > 1$ et que tout nombre flottant représentable dans le format (N_i, N_s, E_i) a un nombre flottant équivalent (en valeur réelle) représentable dans le format $(N_i, N_s - 1, E_i)$, alors $\beta \mid N_s$.

Il manque néanmoins les dénormalisés au TMS 320 C3x pour que l'équivalence fonctionnelle puisse être envisagée. Il ne faut pas non plus oublier le problème de dépassement de capacité qui permet à la notation en complément à 2 de représenter une unique valeur (la plus grande en valeur absolue) de plus que la notation signe-valeur absolue.

6.6 Théorème de Sterbenz

Le théorème de Sterbenz [Ste74] est bien connu (voir aussi le chapitre 4 page 35 et son préambule sur la paternité du théorème) : si x et y sont des nombres flottants IEEE-754 représentables tels que $\frac{y}{2} \leq x \leq 2 \times y$, alors $x - y$ est représentable. En toute logique, j'aimerais pouvoir démontrer que :

> *« Pour toute base et tout format flottant, si x et y sont représentables et tels que $\frac{y}{2} \leq x \leq 2 \times y$, alors $x - y$ est représentable. »*

Malheureusement, cet énoncé est faux. Considérons $\beta = 2$ et un format flottant qui autorise des mantisses entre -1100_2 et 1111_2. Soit $x = (1111_2, 0)$ et $y = (1110_2, 1)$ qui sont tous deux représentables et qui vérifient $\frac{y}{2} \leq x \leq y \leq 2 \times y$. On a alors $x - y =_{\mathbb{R}} -1101_2$, valeur qui ne peut être représentée exactement dans ce format flottant. Je peux néanmoins énoncer d'autres résultats de ce type intéressants et corrects.

6.6.1 Cas positif

Le lemme suivant est très général : il est valable pour toute base et tout format flottant, mais ne correspond qu'aux cas où $x - y$ est positif (du même signe que x et y) :

Lemme 40 (`SterbenzIAux1` provenant de `FnormI`)
Soient deux nombres flottants x et y représentables tels que

$$y \leq x \leq 2 \times y,$$

alors il existe un nombre flottant représentable z tel que $z =_{\mathbb{R}} x - y$.

De plus, la preuve Coq de ce lemme est en fait exactement la preuve du théorème équivalent utilisant la formalisation du format IEEE-754 : SterbenzAux de [DRT01] qui n'était censée marcher que pour $N_i = N_s$.

Je déduis facilement le théorème suivant, en appliquant deux fois le résultat précédent. La première condition de ce théorème a été explicitée dans le paragraphe 6.3.1.

Théorème 41 (`SterbenzOppI` provenant de `FnormI`)
On suppose que le format est tel que tout nombre flottant admette un opposé.
Soient deux nombres flottants x et y représentables tels que

$$\frac{y}{2} \leq x \leq 2 \times y,$$

alors il existe un nombre flottant représentable z tel que $z =_{\mathbb{R}} x - y$.

Énoncé Coq

```
Theorem SterbenzOppI :
 forall x y : float,
 (forall u : float, FboundedI b u ->
    exists v : float, FtoRradix v = (- u)%R /\ FboundedI b v) ->
 FboundedI b x -> FboundedI b y ->
 (/ 2%nat * y <= x)%R -> (x <= 2%nat * y)%R ->
 exists z : float,
   FtoRradix z = (x - y)%R /\ FboundedI b z.
```

6.6.2 Autres cas

Bien que je puisse espérer que tous les systèmes utilisables ont les propriétés naturelles décrites dans les paragraphes 6.3 et 6.4, notre formalisation très générale m'a permis de prouver un théorème s'appliquant à tous les formats couverts.

Lemme 42 (`SterbenzI` provenant de `FnormI`)
On suppose que le format est tel que $|N_i - N_s| \leq \delta$.
Soient deux nombres flottants $x = (n_x, e_x)$ canonique et $y = (n_y, e_y)$ représentable tels que

$$\frac{y + \delta\beta^{\min(e_x, e_y)}}{2} \leq x \leq 2y,$$

alors il existe un nombre flottant représentable z tel que $z =_{\mathbb{R}} x - y$.

La preuve est assez longue et nécessite trois lemmes pour tenir compte de tous les cas possibles : $x \leq y$ ou pas et $e_x \leq e_y$ ou pas puisque l'ordre lexicographique n'est pas garanti dans ce système générique.

6.7 Salmigondis 3 : la risée des machines

Tous les processeurs génériques et la plupart des processeurs dédiés proposent et utilisent plusieurs précisions. Ainsi, la norme IEEE-754 demande au moins la simple (32 bits) et la double (64 bits) précision mais les processeurs offrent souvent également une précision supérieure. Le but ici est de profiter de ces deux (au moins) formats : si x et y sont précis et proches l'un de l'autre, alors la différence entre x et y peut être représentée dans une précision moindre.

Théorème 43 (`SterbenzApproxI` provenant de `FnormI`)
Soient $\mathbb{B}_1$ et $\mathbb{B}_2$ deux formats flottants tels que $-{E_i}^2 \leq -{E_i}^1$. On suppose que tout nombre flottant représentable dans le format $\mathbb{B}_1$ (resp. $\mathbb{B}_2$) admet un opposé représentable dans son format. Soit

$$\xi = \frac{\min\left({N_i}^2, {N_s}^2\right) + 1}{\max\left({N_i}^1, {N_s}^1\right) + 1}.$$

Soient x et y deux nombres flottants représentables dans $\mathbb{B}_1$, de même signe et tels que

$$\frac{|y|}{1+\xi} \leq |x| \leq (1+\xi)\,|y|,$$

alors il existe un nombre flottant z représentable dans le format $\mathbb{B}_2$ tel que $z =_{\mathbb{R}} x - y$.

Énoncé Coq

```
Theorem SterbenzApproxI :
 forall (rho : R) (x y : float),
 (0 < rho)%R ->
 rho = Rmin
   (Rmin (Zsucc (vNumInf b1) / Zsucc (vNumInf b2))
      (Zsucc (vNumSup b1) / Zsucc (vNumSup b2)))
   (Rmin (Zsucc (vNumInf b1) / Zsucc (vNumSup b2))
      (Zsucc (vNumSup b1) / Zsucc (vNumInf b2))) ->
 (- dExpI b2 <= - dExpI b1)%Z ->
 (forall x : float,
   FboundedI b2 x -> exists y : float, y = (- x)%R :>R /\ FboundedI b2 y)->
 (forall x : float,
   FboundedI b1 x -> exists y : float, y = (- x)%R :>R /\ FboundedI b1 y)->
  FboundedI b1 x -> FboundedI b1 y -> (0 <= x * y)%R ->
  (/ (1 + / rho) * Rabs y <= Rabs x)%R ->
  (Rabs x <= (1 + / rho) * Rabs y)%R ->
    exists u : float, u = (x - y)%R :>R /\ FboundedI b2 u.
```

La preuve est ici aussi très longue et utilise 7 lemmes intermédiaires : il faut en effet séparer les cas selon que $x - y$ est positif ou négatif, selon la valeur de $\rho = \frac{1}{\xi}$ et utiliser les symétries à bon escient.

6.8 Validité des résultats précédents

La plupart des résultats des paragraphes précédents assurent l'existence d'un nombre flottant représentable ayant telle ou telle propriété. Le problème restant est de savoir si ce nombre flottant est effectivement le résultat renvoyé par le calcul de l'unité flottante. Pour cela, nous nous sommes penchés sur la façon dont le TMS 320 C3x exécute une addition (ou une soustraction). L'algorithme extrait de [Tex97] est repris dans la figure 6.1.

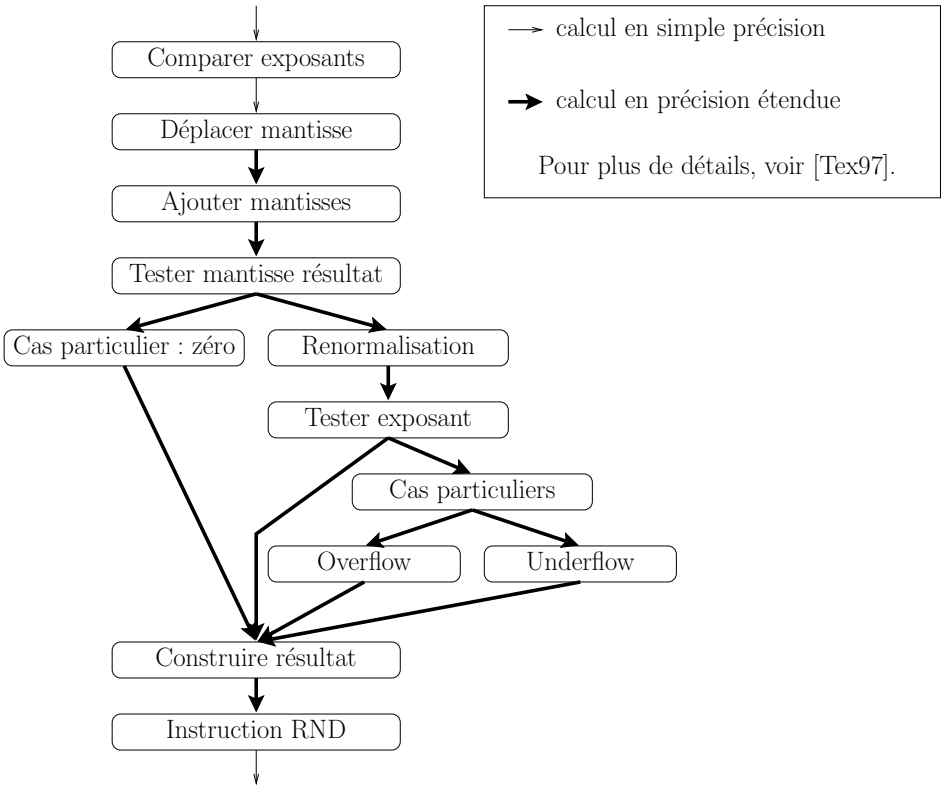


FIG. 6.1 – Schéma d'exécution d'une addition flottante sur le TMS 320 C3x.

L'opération est tout d'abord exécutée en précision étendue (flèche grasse), puis le résultat est arrondi. Cela signifie que 8 bits supplémentaires sont utilisés pour les calculs intermédiaires, ce qui suffit à garantir, pour une addition/soustraction simple précision, que si le résultat est représentable, alors il est renvoyé par l'unité flottante. Par contre, si l'utilisateur fait une soustraction en précision étendue sans l'arrondir ensuite en simple précision, il n'y a pas de bit de garde et le résultat représentable pourrait ne pas être renvoyé.

Dans les conditions de Sterbenz, on a $x - y \in [-y/2; y]$, d'où l'impossibilité d'un dépassement de capacité supérieur. Par contre, un dépassement de capacité inférieur est possible et le TMS 320 C3x renverra alors 0 puisque les dénormalisés ne sont pas implantés. À part cette absence et en simple précision, le TMS 320 C3x fonctionne donc comme décrit par les théorèmes précédents.

Chapitre 7

Arrondi fidèle

Mathematics alone proves, and its proofs are held to be of universal and absolute validity, independant of position, temperature or pressure.
Philip J. Davis [Dav72]

Bien que plus faible que celle d'arrondi, la notion d'arrondi fidèle me sera très utile au chapitre 9. Un arrondi fidèle est soit l'arrondi vers $+\infty$, soit l'arrondi vers $-\infty$, sans que ce choix puisse être connu. Cette définition semble simple mais cache en fait quelques subtilités aux alentours des puissances de la base.

L'arrondi fidèle est un précurseur de l'arrondi correct [Pri92] : il est en effet bien plus facile à obtenir. À cause du dilemme du fabricant de tables [Gol91, Lef00, LM01], il est difficile d'arrondir correctement et un arrondi fidèle accompagné d'une borne sur l'erreur relative, est une solution acceptable. Il semble ironique de revenir à l'arrondi fidèle aujourd'hui mais le résultat d'une séquence de calculs flottants arrondis correctement peut (sous certaines conditions) être un arrondi fidèle de la valeur mathématique (voir le chapitre 9).

De plus, il permet de généraliser non seulement tous les arrondis IEEE (au plus proche, vers 0 et vers $\pm\infty$), mais également l'arrondi stochastique [LV74, Che95] ainsi que d'autres arrondis exotiques [PV93]. Je vais donc détailler un peu les propriétés et les exigences de l'arrondi fidèle.

7.1 Définition

Je définis un nombre flottant comme étant un arrondi fidèle d'une valeur réelle si ce nombre flottant est soit l'arrondi vers $-\infty$, soit l'arrondi vers $+\infty$ de cette valeur réelle. Sur la figure 7.1, le réel z peut être arrondi fidèlement par l'un quelconque des deux nombres flottants l'entourant. Je note $\square(z)$ *un* arrondi fidèle du réel z.

Énoncé Coq
```
Definition MinOrMax :=
   [z : R] [f : float]  (isMin b radix z f) \/ (isMax b radix z f).
```

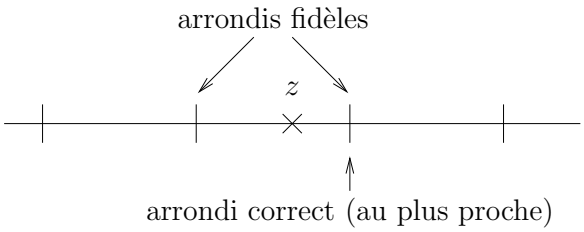


Fig. 7.1 – Définition d'un arrondi fidèle.

Bien sûr, il n'y a aucun moyen de savoir quel va être, ou quel a été le mode d'arrondi choisi lorsqu'un résultat fidèle est rendu. Comme l'arrondi, la fidélité est une relation liant un réel à un nombre flottant (ainsi que la base et le format flottant). C'est particulièrement nécessaire ici étant donné qu'il y a quasiment toujours deux nombres flottants vérifiant cette propriété. Il n'y en a qu'un lorsque le réel est un nombre flottant représentable, la propriété impose alors que ce nombre flottant soit retourné. Néanmoins, pour la simplicité des notations et de façon similaire aux arrondis classiques, l'arrondi fidèle est ici noté comme une fonction.

7.2 Propriétés de base

7.2.1 Fidélité et opposition

Il est aisé de prouver la propriété suivante. En effet, si $-f$ est l'arrondi vers $+\infty$ de $-z$, alors f est l'arrondi vers $-\infty$ de z et *vice versa*.

Lemme 44 (`MinOrMax_Fopp` provenant de `MinOrMax`)
Pour tout nombre flottant représentable f et tout réel z, si $(-f) = \square(-z)$ alors $f = \square(z)$.

7.2.2 Arrondi fidèle → ulp

Je déduis facilement de la définition le résultat suivant :

Lemme 45 (`MinOrMax_Rlt` provenant de `MinOrMax`)
Si z est un réel et f un nombre flottant représentable tel que $f = \square(z)$, alors

$$|z - f| < \mathrm{ulp}(f).$$

En effet, $f = \square(z)$ est soit l'arrondi vers $+\infty$, soit vers $-\infty$. Dans tous les cas, f est à strictement moins d'un ulp de la valeur exacte d'où le résultat.

7.2.3 Ulp → arrondi fidèle ?

On pourrait croire que si un nombre flottant f vérifie $|z - f| < \text{ulp}(f)$, alors f est un arrondi fidèle de z. C'est vrai le plus souvent comme sur la partie gauche de la figure 7.2. Sur cette figure et les suivantes, la partie grisée correspond aux réels z tels que $f = \square(z)$ et la partie noire hachurée aux z tels qu'une inégalité précisée est vérifiée, ici $|z - f| < \text{ulp}(f)$.

Les deux ensembles sont égaux sur la partie gauche. Malheureusement, si f est une puissance de la base, ce n'est plus vrai comme le montre la partie droite de la figure 7.2. Cela correspond aux cas où $f = \beta^n$. En effet, le nombre flottant prédécesseur n'est alors plus à distance 1 ulp(f) de f comme dans les autres cas, mais à distance $\text{ulp}(f^-) = \frac{1}{\beta}\,\text{ulp}(f)$.

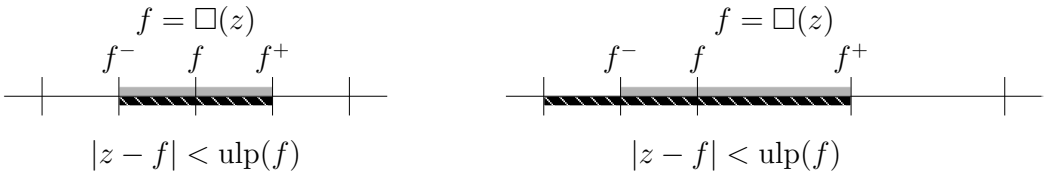


FIG. 7.2 – Ulp et arrondi fidèle.

7.3 Comment garantir la fidélité ?

La condition simple espérée ne suffit pas à prouver la fidélité du résultat et la condition $|z - f| \leq \frac{\text{ulp}(f)}{2}$ non plus. Je dois énoncer deux propriétés pour couvrir tous les cas et j'utilise pour cela le prédécesseur d'un nombre flottant f noté f^-. Si f est représentable, f^- est le nombre flottant représentable le plus grand strictement inférieur à f.

La première propriété est vraie dans tous les cas comme le montre la figure 7.3.

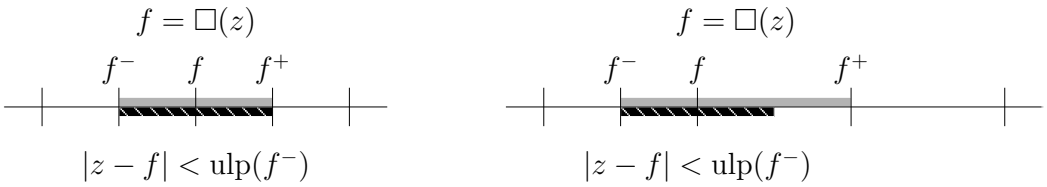


FIG. 7.3 – Première façon de prouver la fidélité.

Lemme 46 (`MinOrMax1` provenant de `MinOrMax`)
Soit f est un nombre flottant représentable. Si $f > 0$ et $|z - f| < \text{ulp}(f^-)$, alors f est un arrondi fidèle de z.

La seconde propriété est également vraie dans tous les cas comme le montre la figure 7.4.

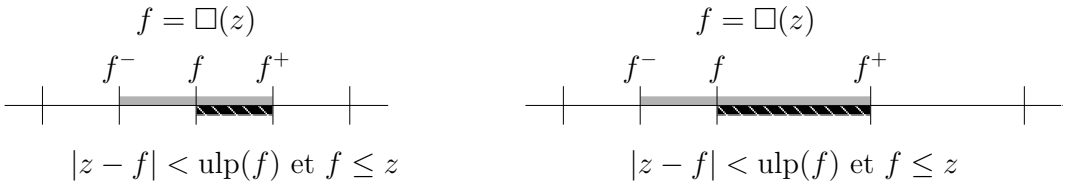


FIG. 7.4 – Seconde façon de prouver la fidélité.

Lemme 47 (`MinOrMax2` provenant de `MinOrMax`)
Soit f est un nombre flottant représentable. Si $f > 0$ et $|z - f| < \text{ulp}(f)$ et $f \leq z$, alors f est un arrondi fidèle de z.

Pour tenir compte du cas où le nombre flottant est nul, cas simple mais où les résultats précédents ne sont pas applicables, j'ai également prouvé le lemme suivant. Même si f est connu, il est plus facile pour la suite de continuer à utiliser f^- par homogénéité avec les résultats précédents.

Lemme 48 (`MinOrMax3` provenant de `MinOrMax`)
Soit f un nombre flottant représentable tel que $f = 0$. Si $|z - f| < \text{ulp}(f^-)$ alors f est un arrondi fidèle de z.

Dans ce cas, $\text{ulp}(f^-) = \text{ulp}\left(-\beta^{-E_i}\right) = \beta^{-E_i}$.

7.4 Stabilité par double arrondi

L'arrondi fidèle est stable relativement au double arrondi : si on arrondit fidèlement un réel z en un nombre flottant u et que l'on arrondit ensuite ce flottant u sur un format moins précis en un flottant v, alors v est un arrondi fidèle de z sur le format le moins précis, comme le montre la figure 7.5.

Je note $\mathcal{B}_1 = (p_1, -E_i^1)$ le premier format, le moins précis, et $\mathcal{B}_2 = (p_2, -E_i^2)$ le second format, le plus précis.

Lemme 49 (`BoundedBounded` provenant de `DoubleRound`)
Si $p_2 \leq p_1$ et $p_1 - p_2 \leq E_i^1 - E_i^2$, alors tout nombre flottant représentable dans le format $\mathcal{B}_2$ l'est aussi dans le format $\mathcal{B}_1$.

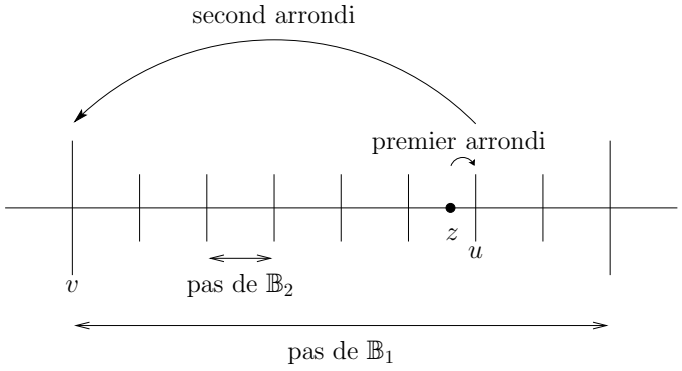


FIG. 7.5 – Double arrondi.

La propriété de double arrondi peut s'énoncer ainsi :

Théorème 50 (`DblRndStable` provenant de `DoubleRound`)
Je suppose $p_2 \leq p_1$ et $p_1 - p_2 \leq E_i^1 - E_i^2$.
Si z est un réel, alors tout nombre flottant $\square_1(\square_2(z))$ est également un arrondi fidèle de z selon $\mathcal{B}_1$.

ÉNONCÉ COQ
```
Theorem DblRndStable :
 forall (z : R) (p q : float),
 prec2 <= prec1 ->
 (prec1 - prec2)%Z = (dExp b1 - dExp b2)%Z ->
 Fbounded b1 p -> Fbounded b2 q ->
 MinOrMax radix b1 z p -> MinOrMax radix b2 p q ->
     MinOrMax radix b2 z q.
```

Cela signifie en particulier que tout arrondi (au plus proche, dirigé ou fidèle) suivi d'un autre arrondi (au plus proche, dirigé ou fidèle) moins précis donne un résultat qui est un arrondi fidèle de la valeur initiale. Ainsi, pour calculer en précision normale le TMS 320 C3x applique l'algorithme décrit en figure 6.1 page 66 [Tex97] :

- calcul et arrondi en précision étendue
- arrondi en précision normale du résultat précédent.

Je garantis donc que le résultat est toujours un arrondi fidèle de la valeur exacte. Il est impossible de garantir que l'arrondi est correct : en effet les bits perdus lors du premier arrondi peuvent être déterminants dans le choix de l'arrondi final dans le cas où le résultat exact est très proche d'un nombre machine (arrondi dirigé) ou du milieu de deux nombres machines (arrondi au plus proche). Pour de tels exemples, voir les paragraphes 3.1 et 3.4 du chapitre 3.

Enfin, même si le double arrondi n'est qu'un arrondi fidèle, il est souvent bien meilleur qu'un arrondi au hasard : si le second arrondi est au plus proche, la distance entre le résultat et la valeur exacte est très proche du demi-ulp.

Lemme 51 (`DoubleRound2` provenant de `DoubleRound`)
Je suppose $p_2 \leq p_1$ et $p_1 - p_2 = E_i^1 - E_i^2$.
Soit un réel z, si $u = \square_1(z)$ et $v = \circ_2(u)$, alors

$$|z - v| < \mathrm{ulp}_2(v) \left(\frac{1}{2} + 2^{1+p_2-p_1} \right).$$

7.5 Conclusion

Les résultats des chapitres de cette partie montrent bien la souplesse du modèle : on peut le tordre et le malmener, il reste utilisable et des résultats surprenants et utiles en sortent.

L'extension correspondant à la norme IEEE-754 peut sembler artificielle et peu utilisable. Ce développement a néanmoins le mérite de permettre la validation du modèle par rapport au principal document de référence du domaine. Quant à l'utilisabilité, elle est surtout limitée par la malléabilité des tableaux de bits Coq : ce ne sont que des listes de taille imposée et il est difficile de les manipuler aussi facilement et intuitivement qu'on le souhaiterait. Cela rend pour l'instant l'utilisation de cette extension périlleuse et compliquée.

L'extension correspondant à l'arrondi fidèle n'a pas pour l'instant beaucoup d'intérêt à part le résultat de double arrondi. Mais ces résultats préliminaires sont nécessaires aux résultats du chapitre 9 qui correspond à une application très tangible.

L'extension principale est celle correspondant aux autres systèmes et au complément à la base. Les résultats obtenus sont originaux et inattendus. Ils concordent souvent entre systèmes mais diffèrent parfois sur les propriétés les plus basiques. La généricité initialement voulue s'est en fait vite évaporée au profit de résultats concernant quasi-uniquement le complément à la base. Mais ces résultats sont tout de même importants car les théorèmes habituels et évidents dans le système IEEE-754 ne sont pas forcément vérifiés dans un autre système. Il est donc difficile de faire confiance à une preuve papier faite par un humain habitué à la norme IEEE-754 : comment être sûr que son intuition habituellement correcte n'a pas dépassé la vérité dans ce système différent ?

Troisième partie

Application du modèle à deux bibliothèques

Chapitre 8

Expansions

It is complexity of design and process that got us (and Murphy's Law!).
John R. Garman [Gar81]

À la suite des développements du chapitre 3, il est possible de calculer avec plus de précision juste en utilisant l'arithmétique flottante du processeur avec des expansions. Bien qu'elles puissent sembler simples, les opérations de base sont assez complexes si l'efficacité est prise en compte. Nous avons donc cherché non seulement à prouver ces algorithmes, mais aussi à les améliorer.

8.1 Introduction

Après la publication des résultats rappelés au chapitre 3, les expansions ont été introduites par Priest [Pri91] à partir de la synthèse de Knuth [Knu73]. Puis Bailey [Bai93] et d'autres sur les BLAS [LDB+02] ainsi que Shewchuk [She96, She97] développèrent des bibliothèques disponibles sur Internet avec pour ce dernier une application à la géométrie algorithmique.

Les « expansions » sont un format de donnée permettant de calculer avec une précision supérieure à celle de la machine tout en utilisant l'unité flottante du processeur. Cette unité est en effet très optimisée dans les processeurs actuels et donc les temps de calcul sont assez bons. De plus, la conversion avec le format interne est immédiate ce qui permet de faire facilement varier la précision au cours du programme. Les expansions sont utilisées à plusieurs fins.

- De la précision étendue : on calcule avec autant de bits qu'on le peut. On n'est alors limité que par le format de donnée initial. En double précision, on ne peut utiliser des nombres que nuls ou compris entre $2^{1024} - 2^{971} \approx 1{,}80\ 10^{308}$ et $2^{-1074} \approx 4{,}94\ 10^{-324}$, ce qui fait au plus environ 2098 bits ou environ 633 chiffres décimaux.
- De la précision étendue locale : au lieu d'avoir 53 bits de précision, on en a 100 ou 200 localement. Cela permet d'avoir plus de précision à un endroit critique [HLB01].
- De la précision adaptable : on ne calcule que les composantes dont on a besoin pour arriver à la précision finale désirée. Les calculs intermédiaires se font alors de plus en plus précis jusqu'à ce que le dernier opérateur fournisse une réponse assez précise [MF01].

8.2 Définitions

Je me place dans ce chapitre uniquement en base 2. Une expansion x de taille n est un tableau de nombres machines usuels (en général double précision) $x_0, x_1, \ldots, x_{n-1}$. La valeur de x est alors

$$\sum_{i=0}^{n-1} x_i.$$

Pour des questions techniques, notamment pour pouvoir faire du calcul adaptatif, le concept de pseudo-expansion a été introduit par Daumas et Finot-Moreau [Dau99, DF98, DF99] : on autorise les composantes du résultat d'une opération à se chevaucher de quelques bits, ce qui signifie que l'on oblige

$$\forall i \qquad |x_{i+i}| \leq 2^{\zeta+1-p} \times |x_i|$$

avec ζ valant le nombre de bits de chevauchement autorisés, typiquement 3 ou 4, comme le montre la figure 8.1 pour $\zeta = 4$.

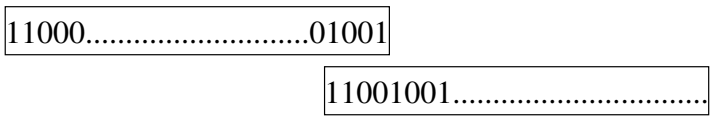


Fig. 8.1 – Composantes chevauchantes d'une expansion avec $\zeta = 4$.

Cela permet le calcul en-ligne [TE77, Erc84, Mul89] : les opérateurs acceptent en entrée des pseudo-expansions et renvoient des pseudo-expansions. Nous faisons du en-ligne logiciel : on fait entrer les composantes dans les opérateurs les unes après les autres, celle de poids le plus fort en tête. Après avoir fourni les premières composantes, l'opérateur donne alors au fur et à mesure les composantes du résultat qui peuvent être utilisées par un autre opérateur.

Si je veux un résultat final qui est une vraie expansion non chevauchante, je peux compresser une pseudo-expansion en une expansion, mais cela nécessite de connaître toutes les composantes et donc d'attendre la fin du calcul. Or ce n'est pas forcément utile de connaître la valeur exacte du résultat : en géométrie algorithmique, on s'intéresse par exemple non pas à la valeur, mais au signe d'un déterminant pour savoir si un point est à droite ou à gauche d'une droite, à l'intérieur ou à l'extérieur d'un cercle [She97].

8.3 Opérateur Σ_3

Le principal opérateur de calcul dont j'ai besoin est l'opérateur Σ_3 décrit en figure 8.2. Il se base sur l'opérateur d'addition conditionnelle qui permet de calculer l'arrondi d'une somme de nombres flottants et l'erreur commise [Dek71, Knu97]. Cet opérateur est conditionnel dans le sens où il présuppose que l'entrée « du haut » (celle avec une flèche entrante) est telle que son exposant est supérieur à celui de l'autre entrée. En contrepartie, il ne coûte que 3 additions au lieu de 6 dans le cas de l'opérateur non conditionnel (sans flèche entrante).

Intuitivement, l'opérateur Σ_3 rend a', b' et c' tels que $a'+b'+c' = a+b+c$ et $a' \gg b' \gg c'$. Il existe malheureusement des cas dégénérés où $a' = c' = 0$ et $b' \neq 0$.

J'ai donc montré que :

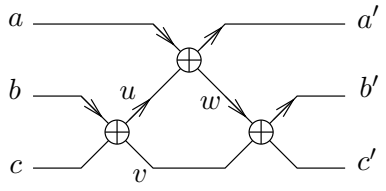


FIG. 8.2 – Opérateur Σ_3.

Lemme 52 (`bound3Sum` et `OutSum3` provenant de `ThreeSum2`)
Soient a, b et c des nombres flottants représentables tels que les sommes conditionnelles soient valides. Soient a', b' et c' les résultats de l'opérateur Σ_3. Si $c' \neq 0$, alors

$$|b' + c'| \leq \frac{6}{2^p - 1}|a'|$$

$$(2^p - 1)\, 2^{e_c} \leq \frac{3}{2^{p-1} - 1}|a'|.$$

Lemme 53 (`ThreeSumLoop` provenant de `ThreeSumProps`)
Si $p \geq 4$, soient a, b et c des nombres flottants représentables avec $e_a \geq e_b \geq e_c$ et soient a', b' et c' les résultats de l'opérateur Σ_3. On suppose que $a = 0 \vee (2^p - 1)\, 2^{e_c} + |b| \leq (2^p - 1)\, 2^{e_a}$. Alors l'opérateur est valide et il existe des représentants a'', b'' et c'' de a', b' et c' tels que

$$\begin{aligned} & e_{a''} \geq e_{b''} \geq e_{c''} \geq e_c \\ c'' = 0 \quad \rightarrow \quad & a'' = 0 \ \vee \ (2^p - 1)\, 2^{e_c} + |b''| \leq (2^p - 1)\, 2^{e_{a''}} \\ c'' \neq 0 \quad \rightarrow \quad & (2^p - 1)\, 2^{e_c} + |c''| \leq (2^p - 1)\, 2^{e_{b''}}. \end{aligned}$$

8.4 Addition

Avec l'opérateur précédent et l'opérateur MQ (Merge Queue), je peux immédiatement construire un opérateur décrit en figure 8.3 qui ajoute deux pseudo-expansions. A et B sont les pseudo-expansions triées par exposant que je veux ajouter et C est la sortie de l'opérateur. Je fais entrer les composantes de A et B au fur et à mesure de leur disponibilité dans la Merge Queue de celle de plus grand exposant à celle de plus petit exposant. La MQ transmet alors au Σ_3 la composante d'exposant le plus grand entre les deux qui lui sont proposées. Le fait que ce soit des pseudo-expansions garantit que les composantes suivantes seront plus petites.

L'algorithme est simple : au début a et b sont nuls. Après chaque calcul effectué par l'opérateur Σ_3, on regarde la valeur de c'. Si c' est nul, alors a' et b' sont les nouvelles valeurs de a et b, sinon, a' est la composante suivante de C et b' et c' sont les nouvelles valeurs de a et b. On itère jusqu'à ce qu'on aie épuisé toutes les composantes de A et B. Il ne reste alors que les valeurs de a et b. On fait alors une addition exacte conditionnelle et on ajoute à C les composantes $a \oplus b$ et $a + b - (a \oplus b)$.

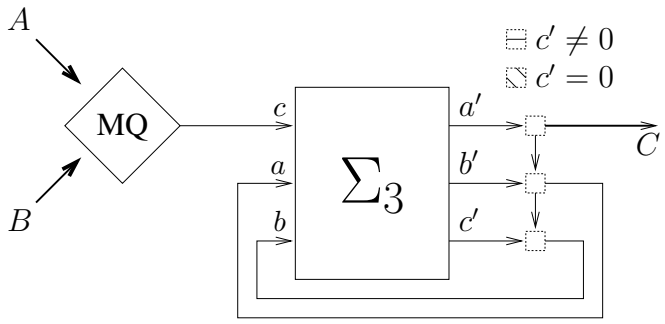


FIG. 8.3 – Opérateur d'addition d'expansions.

Théorème 54 (`FexpAdd_main` provenant de `FexpAdd`)
Soit une liste L triée par exposants, soit l_L la longueur de L, c'est-à-dire la somme des longueurs de A et de B. On suppose

$$l_L < \frac{2^{p-1}-1}{3}.$$

On a alors les trois propriétés suivantes :
- Toutes les additions exactes conditionnelles sont valides.
- La somme des éléments de L est égale à la somme des éléments de C.
- Pour tout i,

$$|c_{i+1}| \leq \frac{3l_L+3}{2^{p-1}-1-3l_L}|c_i|.$$

ÉNONCÉ COQ
```
Theorem FexpAdd_main :
 (IsExp b input) ->
 (6%nat * length input * / (pPred (vNum b) - 1%nat) < 1)%R ->
  exists output : list float,
    (input <> nil -> length output <= S (length input)) /\
    sum input = sum output :>R /\
    IsRleEpsExp output.
```

Ce chevauchement correspond au maximum à 7 bits pour la simple précision et 9 pour la double précision (pour A et B de longueurs maximales). En effet, le nombre maximum de composantes est limité par la plage d'exposants disponibles. Cela peut sembler beaucoup, surtout qu'en pratique le chevauchement n'est que de 2 ou 3 bits en double précision, mais cette borne supérieure est prouvée.

Cette preuve est particulièrement forte car elle montre la validité des trois additions exactes conditionnelles dans le Σ_3. Jusqu'à maintenant, l'addition d'expansions se basait sur un opérateur Σ_3 composé de deux additions conditionnelles et une usuelle entre a et u. Étant donné

qu'il est possible de remplacer cette addition usuelle par une addition conditionnelle, on économise trois additions à chaque étape de l'algorithme ce qui le rend bien plus efficace.

La seule hypothèse est que la liste L, fusion de A et B, est triée par exposants et n'est pas trop longue. La preuve est très générale et sera réutilisée pour la multiplication.

8.5 Multiplication

L'algorithme de multiplication est présenté en figure 8.4. Il est basé sur l'opérateur Σ_3 présenté en 8.3 et les opérateurs suivants : PP génère les produits partiels $a_i \times b_j$ (comme somme de deux nombres flottants) au fur et à mesure, PQ est une queue de priorité et MQ est un opérateur de tri.

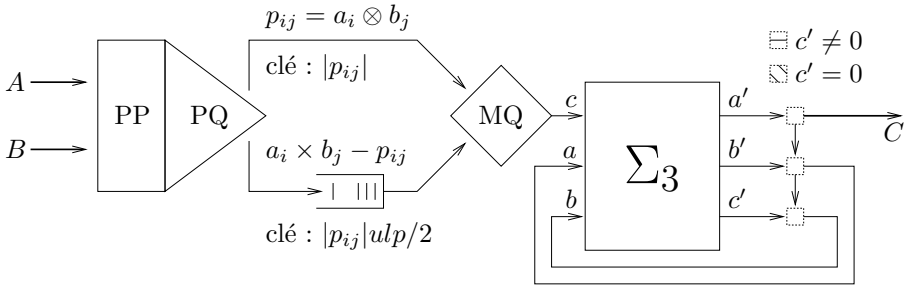


FIG. 8.4 – Opérateur de multiplication d'expansions.

La queue de priorité PQ utilise la valeur de $|a_i \otimes b_j|$ pour trier les produits partiels. Par contre, MQ utilise la clé indiquée sur la figure 8.4 pour trier les valeurs qui lui sont fournies. Cela permet d'être sûr que la liste obtenue en sortie de MQ est bien triée par exposants. Ainsi, en utilisant le résultat précédent, je prouve que la pseudo-expansion obtenue à la fin de l'algorithme est bien le produit des deux expansions fournies et le chevauchement est borné par la même inégalité que celle du théorème 54.

8.6 Division

La division est basée sur une partie des opérateurs de la multiplication présentés ci-dessus. L'opérateur supplémentaire est noté Σ_3' et est décrit en figure 8.5. Il ressemble beaucoup à l'opérateur Σ_3 de la figure 8.2. La première différence est que l'addition exacte entre a et u n'est plus conditionnelle. La seconde différence est que l'addition entre w et v est exacte : on aura toujours $c' = 0$. Cet opérateur a donc 3 entrées pour 2 sorties et est composé de 10 additions.

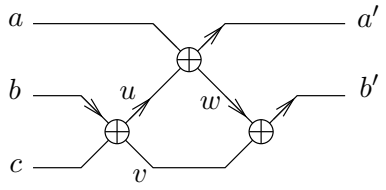


FIG. 8.5 – Opérateur Σ_3'.

L'algorithme est présenté en figure 8.6 où on divise l'expansion R par l'expansion D, le résultat étant l'expansion Q. La division est une sorte de multiplication/accumulation où on calcule au fur et à mesure $R + Q \times (-D)$, sachant que Q est au début inconnu et que le résultat idéal de ce calcul est zéro. On va donc calculer les composantes q_i de Q comme étant le meilleur choix pour rendre ce calcul nul, dans la mesure du connu, c'est-à-dire des composantes déjà examinées.

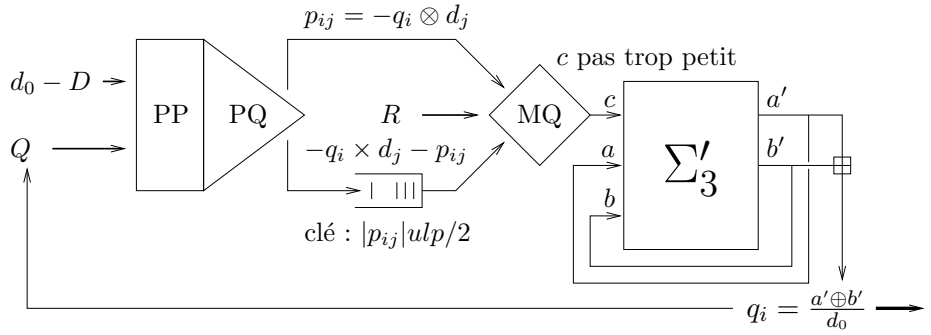


FIG. 8.6 – Opérateur de division d'expansions.

Pour fonctionner, il faut d'une part garantir que les valeurs successives de c sont triées par exposant. On ne laisse donc passer une valeur c que si elle n'est pas trop petite.

La condition précise que doit respecter c est

$$c \geq 2^{2-p}|a \oplus b| \ \wedge \ c \geq \frac{|a \oplus b|\,\varepsilon_D}{(1-2^{-p})^2}$$

où $\varepsilon_D = \max_j(|d_{j+1}|/|d_j|)$. Ceci signifie que ε_D vaut $\zeta 2^{-p}$, mais que si D est compressée, alors cette valeur peut descendre à 2^{-p}.

Si c est trop petit, on fournit alors une nouvelle composante de Q qui est $q_i = (a' \oplus b') \oslash d_0$. Les nouvelles valeurs de a et b sont alors l'arrondi de $a' + b' - d_0 q_i$ et l'erreur commise.

Cet algorithme et cette condition garantissent que $a + b + c$ est toujours représentable comme la somme de deux nombres flottants, que les futurs produits partiels $q_i \otimes d_j$ sont plus petits que les composantes déjà entrées dans l'opérateur Σ'_3 et que Q est une pseudo-expansion. Je peux borner le nombre de bits de chevauchement par une formule complexe qui conduit à 10 bits en double précision et 9 en simple précision IEEE-754 au maximum si D est quasi-compressée.

Ces affirmations de correction et de chevauchement n'ont pas été démontrées en Coq mais une preuve papier se trouve en annexe B.

Chapitre 9

Évaluation fidèle par la méthode de Horner

Mathematicians give symposium and colloquium talks which attempt to convince doubting (sometimes hostile) audiences of their arguments, and they scribble on napkins in university cafeterias and expensive restaurants.
Richard A. DeMillo, Richard J. Lipton et Alan J. Perlis [DLP77]

L'évaluation de fonctions élémentaires passe souvent par l'évaluation de polynômes avec un argument petit. Dans ces conditions, la méthode de Horner est stable et le résultat est très bon numériquement. Nous avons voulu quantifier la façon dont le résultat est bon ainsi que des conditions suffisantes pour le garantir.

L'évaluation de fonctions élémentaires est souvent basée sur deux briques distinctes : la réduction d'arguments (voir le chapitre 10) et une évaluation polynomiale. Je m'intéresse ici à la seconde partie, beaucoup d'auteurs [Stu80, PFTV89, BP94, Mul97, Mar00, Epp01, Hig02] reconnaissent que l'évaluation de Horner se comporte bien à moins que l'indéterminée ne soit proche d'une des racines du polynôme, mais je cherche un critère précis justifiant ce bon comportement pour un polynôme donné, ce bon comportement étant que l'arrondi est fidèle en plus de la borne sur l'erreur relative obtenue par la récurrence classique [Hig02].

L'étape de réduction d'argument a eu lieu et j'en déduis que l'argument sur lequel l'évaluation polynomiale va s'appliquer est relativement petit, typiquement inférieur à 2^{-4}. Or certaines évaluations polynomiales, dont l'exponentielle et le cosinus, sont du type $a_0 + a_1 x + a_2 x^2 + \dots$ avec a_0 de l'ordre de grandeur de 1 et les $|a_i|$ décroissant rapidement vers zéro. En utilisant la méthode de Horner :

$$\begin{aligned} w_n &= a_n \\ w_{n-1} &= a_{n-1} + w_n \times x \\ w_{n-2} &= a_{n-2} + w_{n-1} \times x \\ &\vdots \\ w_0 &= a_0 + w_1 \times x = P(x). \end{aligned}$$

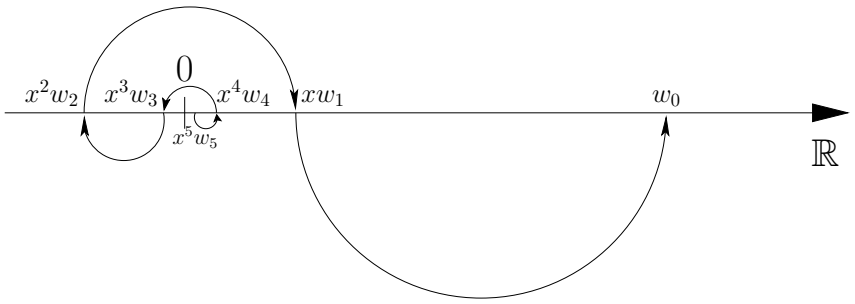


FIG. 9.1 – Évaluation de Horner avec argument réduit : valeurs des $x^i w_i$.

J'ajoute à chaque fois un terme correctif petit à une grandeur de l'ordre de 1. À chaque étape, comme le montre la figure 9.1, la valeur de $x^i w_i$, qui est environ la contribution de w_i au résultat final, est bien plus grande que celle de $x^{i+1}w_{i+1}$.

Il est donc raisonnable de penser que les erreurs commises lors de l'évaluation des termes correctifs les plus petits sont faibles et sont négligeables devant la principale erreur commise : celle du dernier calcul $a_0 + w_1 x$. J'ai donc travaillé sur l'évaluation flottante de $ax + y$ sous la condition que $|ax| \ll |y|$. Cela permet de couvrir le dernier pas d'une évaluation polynomiale quelconque et tous les pas d'une évaluation de type Horner. Malheureusement, il est impossible de garantir un arrondi correct du résultat final. Nous garantissons donc le meilleur possible : un arrondi fidèle (voir le chapitre 7).

Au cours d'une évaluation de Horner, les dépassements de capacités inférieurs sont possibles et il est impossible de l'empêcher et difficile de le prévoir [Dem84]. J'ai donc voulu que mes résultats soient valides quelles que soient les entrées, en présence ou non de nombres dénormalisés. Malgré tout, pour être valable, un tel résultat ne doit pas en pâtir dans le cas général et j'essaie d'obtenir les résultats les plus serrés possible. Seule la base 2 est envisagée ici.

9.1 Inégalités préliminaires et prédécesseur

J'ai par la suite besoin des inégalités suivantes. Elles rendent compte du fait que l'arrondi est très proche de la valeur réelle, et cela que l'arrondi soit un nombre flottant normalisé ou dénormalisé.

Lemme 55 (`FulpLeGeneral` provenant de `Axpy`)
Soit f un nombre flottant représentable, alors

$$\mathrm{ulp}(f) \leq |f| \times 2^{1-p} + 2^{-E_i}.$$

En effet, si le représentant canonique de f est normalisé, alors $\mathrm{ulp}(f) \leq |f| \times 2^{1-p}$ et sinon, $\mathrm{ulp}(f) = 2^{-E_i}$. J'aurais pu aussi aisément prouver que $\mathrm{ulp}(f) \leq \max\left(|f| \times 2^{1-p}, 2^{-E_i}\right)$, mais la présence d'un maximum rend tout calcul suivant extrêmement compliqué et lourd à manipuler. La suite contient en effet beaucoup de calculs sur les réels et l'addition a donc été préférée pour une simple raison de simplicité de la preuve Coq.

J'en déduis alors le lemme suivant.

Lemme 56 (`RoundLeGeneral` et `RoundGeGeneral` provenant de `Axpy`)
Soit z un réel et $f = \circ(z)$,

$$\frac{|z|}{1+2^{-p}} - \frac{2^{-E_i-1}}{1+2^{-p}} \leq |f| \leq \frac{|z|}{1-2^{-p}} + \frac{2^{-E_i-1}}{1-2^{-p}}.$$

En effet, je sais que $|z - f| \leq \text{ulp}(f)/2$ puisque f est un arrondi au plus proche de z. En utilisant le lemme 55, j'obtiens les inégalités précédentes.

Les résultats suivants utilisent la notion de prédécesseur flottant. Pour un nombre flottant donné représentable f, il s'agit du nombre flottant représentable f^- le plus grand strictement inférieur à f. Les valeurs de la mantisse et de l'exposant de f^- se déduisent facilement de celles de f. L'inégalité $f^- < f$ est valide par construction, mais j'ai également :

Lemme 57 (`FpredUlpPos` provenant de `ClosestProp`)
Soit f un nombre flottant représentable avec $f > 0$, alors

$$f^- + \text{ulp}\left(f^-\right) = f.$$

Lemme 58 (`FulpFPredGePos` provenant de `FroundProp`)
Soit f un nombre flottant représentable avec $f > 0$, alors

$$\text{ulp}\left(f^-\right) \leq \text{ulp}(f).$$

Lemme 59 (`FulpFPredLe` provenant de `ClosestProp`)
Soit f un nombre flottant représentable, alors

$$\text{ulp}(f) \leq 2\ \text{ulp}\left(f^-\right).$$

Les preuves sont assez simples, il suffit de séparer les valeurs possibles pour la mantisse et l'exposant de f, d'en déduire celles de f^- et de vérifier dans chaque cas que le résultat est exact.

De façon symétrique, je définis le nombre flottant successeur f^+ qui est le plus petit nombre flottant représentable strictement supérieur à f. Je définis ensuite f^* comme étant le nombre flottant proche de f, mais en direction de zéro, c'est-à-dire $f^* = f^-$ si $f \geq 0$ et f^+ sinon.

9.2 Premières conditions

Je veux donc évaluer $a \times x + y$ sachant que les nombres flottants ne sont pas exacts et que j'évalue d'abord $a \times x$ en calcul flottant puis j'ajoute y, présumé grand par rapport à $a \times x$.

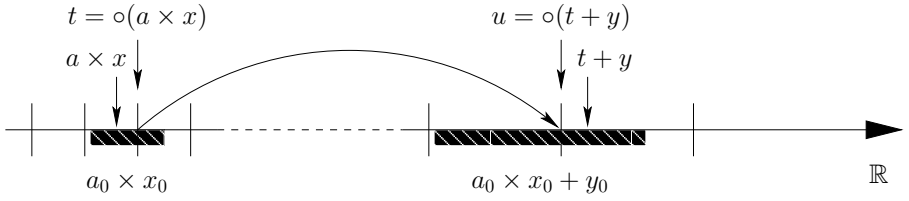


FIG. 9.2 – Exemple de calcul fidèle de $a \times x + y$.

La figure 9.2 montre les notations et les intervalles autorisés pour les valeurs réelles exactes, qui sont notées avec l'indice 0.

Je vais maintenant présenter les théorèmes principaux. Le lemme 60 correspond à la condition la plus stricte mais il dépend des sorties de l'algorithme, ce qui ne peut être testé *a priori*. De plus, il se limite au cas positif.

Lemme 60 (`AxpyPos` provenant de `Axpy`)
Soient les réels a_0, x_0 et y_0. Soient les nombres flottants représentables a, x et y. On définit les nombres flottants représentables t et u comme $t = \circ(a \times x)$ et $u = \circ(t + y)$. Si $u > 0$ et

$$4|t| \leq |u| \qquad \text{et} \qquad |y_0 - y| + |a_0 \times x_0 - a \times x| < \frac{\text{ulp}\,(u^-)}{4},$$

alors $u = \square(a_0 \times x_0 + y_0)$.

Je cherche à borner :

$$\begin{aligned} |u - (a_0 \times x_0 + y_0)| &= |u - (t + y) + t - a \times x + y - y_0 + a \times x - a_0 \times x_0| \\ &< |u - (t + y)| + \frac{\text{ulp}(t)}{2} + \frac{\text{ulp}\,(u^-)}{4} \end{aligned}$$

puis je distingue deux cas.

1. Si $t + y \leq u$, l'arrondi de u s'est fait vers le haut et donc $t + y$ est plus proche de u que de u^- donc
$$|u - (t + y)| \leq \frac{\text{ulp}\,(u^-)}{2}.$$
Comme $4|t| \leq |u|$, j'ai dans la plupart des cas
$$4\ \text{ulp}(t) \leq \text{ulp}(u).$$
Les cas spéciaux sont traités indépendamment avec Coq : ils correspondent à des valeurs dénormalisées pour u et/ou u^- que je regarde séparément.
De plus, d'après le lemme 59, j'ai $\text{ulp}(u) \leq 2\text{ulp}\,(u^-)$. J'en déduis immédiatement $|u - (a_0 \times x_0 + y_0)| < \text{ulp}\,(u^-)$ et je peux conclure par le lemme 46 du chapitre 7.
2. Sinon, j'ai $t + y \geq u$ et donc
$$|u - (a_0 \times x_0 + y_0)| < \frac{\text{ulp}(u)}{2} + \frac{\text{ulp}(u)}{8} + \frac{\text{ulp}(u)}{4} < \text{ulp}(u).$$
et je peux achever la preuve en utilisant le lemme 47 du chapitre 7.

Le lemme suivant correspond à la condition la plus stricte mais il dépend des sorties de l'algorithme et s'applique à tous les cas.

Lemme 61 (`Axpy_tFlessu` provenant de `Axpy`)
Soient les réels a_0, x_0 et y_0. Soient les nombres flottants représentables a, x et y. On définit les nombres flottants représentables t et u comme $t = \circ(a \times x)$ et $u = \circ(t + y)$. Si

$$4|t| \leq |u| \quad \text{et} \quad |y_0 - y| + |a_0 \times x_0 - a \times x| < \frac{\text{ulp}\,(u^*)}{4},$$

alors $u = \square(a_0 \times x_0 + y_0)$.

Ce lemme est exactement le lemme 60 quand $u > 0$. Quand $u < 0$, je réutilise le lemme 60 en prenant comme valeurs $-a$ pour a et $-y$ pour y. Comme l'arrondi au plus proche est stable par passage à l'opposé, les valeurs $-t$ pour t et $-u$ pour u conviennent puisque $-u > 0$. J'en déduis le résultat car l'arrondi fidèle est également stable par passage à l'opposé : c'est le lemme 44 du chapitre 7.

Je dois gérer indépendamment le cas où $u = 0$. Cela implique en particulier, comme $4|t| \leq |u|$, que $t = 0$. Comme u est l'arrondi de $t + y$, cela signifie que $u = t = y = 0$. Donc

$$\begin{aligned} |u - (a_0 \times x_0 + y_0)| &= |a_0 \times x_0 + y_0| \\ &\leq |a \times x - t| + |a \times x - a_0 \times x_0| + |y_0 - y| \\ &< \frac{\text{ulp}(t)}{2} + \frac{\text{ulp}\,(u^*)}{4} \\ &< \text{ulp}(u^-) \end{aligned}$$

et je peux achever la preuve en utilisant le lemme 48 du chapitre 7.

9.3 Conditions suffisantes pour garantir la fidélité

Le théorème précédent n'est pas satisfaisant car les conditions dépendent des sorties de l'algorithme et non pas des entrées contrairement au théorème suivant.

Théorème 62 (`Axpy_opt` provenant de `Axpy`)
Soient les réels a_0, x_0 et y_0. Soient les nombres flottants représentables a, x et y. On définit les nombres flottants représentables t et u comme $t = \circ(a \times x)$ et $u = \circ(t + y)$. Si

$$\frac{5 + 5 \times 2^{-p}}{1 - 2^{-p}} \times \left(|a \times x| + 2^{-E_i - 1}\right) \leq |y|, \text{ et}$$

$$|y_0 - y| + |a_0 \times x_0 - a \times x| \leq 2^{-p-2} \times \left(\left(1 - 2^{1-p}\right) \times |y| - |a \times x|\right) - 2^{-E_i - 2},$$

alors $u = \square(a_0 \times x_0 + y_0)$.

Énoncé Coq

```
Theorem Axpy_opt :
 forall (a1 x1 y1 : R) (a x y t u : float),
 (Fbounded b a) -> (Fbounded b x) -> (Fbounded b y) ->
 (Fbounded b t) -> (Fbounded b u) ->
 (Closest b radix (a * x) t) ->
 (Closest b radix (t + y) u) ->
 (Fcanonic radix b u) -> (Fcanonic radix b t) ->
 ((5%nat + 4%nat * (powerRZ radix (- precision))) *
    / (1 - powerRZ radix (- precision)) *
    (Rabs (a * x) + (powerRZ radix (Zpred (- dExp b)))) <= Rabs y)%R ->
 (Rabs (y1 - y) + Rabs (a1 * x1 - a * x) <=
    (powerRZ radix (Zpred (Zpred (- precision)))) *
    (1 - powerRZ radix (Zsucc (- precision))) * Rabs y +
    - (powerRZ radix (Zpred (Zpred (- precision))) * Rabs (a * x)) +
    - powerRZ radix (Zpred (Zpred (- dExp b))))%R ->
        (MinOrMax radix b (a1 * x1 + y1) u).
```

La preuve de ce théorème est sans grand intérêt : elle se base sur le lemme 61 et une utilisation intensive des inégalités du lemme 56 pour évaluer les valeurs de t, u et u^* à partir de celles de a, x et y.

Ces conditions sont un peu compliquées. Je propose deux versions simplifiées et affaiblies du théorème précédent, à condition que la précision ne soit pas ridiculement faible :

Lemme 63 (`Axpy_Simpl1` provenant de `Axpy`)
Soient les réels a_0, x_0 et y_0. Soient les nombres flottants représentables a, x et y. On définit les nombres flottants représentables t et u comme $t = \circ(a \times x)$ et $u = \circ(t + y)$. Si $p \geq 4$,

$$6 \left(|a \times x| + 2^{-E_i - 1}\right) \leq |y|, \text{ et}$$

$$|y_0 - y| + |a_0 \times x_0 - a \times x| \leq 2^{-p-2} \times \left(\left(1 - 2^{1-p}\right) \times |y| - |a \times x|\right) - 2^{-E_i - 2},$$

alors $u = \square(a_0 \times x_0 + y_0)$.

Lemme 64 (`Axpy_Simpl2` provenant de `Axpy`)
Soient les réels a_0, x_0 et y_0. Soient les nombres flottants représentables a, x et y. On définit les nombres flottants représentables t et u comme $t = \circ(a \times x)$ et $u = \circ(t + y)$. Si $p \geq 4$,

$$6 \left(|a \times x| + 2^{-E_i - 1}\right) \leq |y|, \text{ et}$$

$$|y_0 - y| + |a_0 \times x_0 - a \times x| \leq \frac{2^{-p-1}}{3} \times |y| - 2^{-E_i - 2},$$

alors $u = \square(a_0 \times x_0 + y_0)$.

Les deux conditions correspondent aux deux conditions du théorème 62. Les inégalités ont juste été affaiblies pour être plus lisibles. Pour la seconde simplification, j'ai utilisé la première condition de façon à simplifier la seconde et ne la faire dépendre que de y.

9.4 Utilisation d'un FMA

Le FMA ou Fused-Multiply-and-Add est une nouvelle fonctionnalité de certains processeurs comme l'IA64 d'Intel [Mar00] ou le PowerPC. Elle sera ajoutée à la norme IEEE-754 dans la prochaine révision de cette norme [IEE04]. Le FMA permet de calculer $a \times x + y$ ou plus généralement $\pm a \times x \pm y$ avec un seul arrondi. Il est donc particulièrement intéressant de l'utiliser pour calculer un pas de la méthode de Horner.

Je montre ici que les résultats précédents s'appliquent encore dans ce cas et qu'ils peuvent être simplifiés.

Lemme 65 (`Axpy_FLessu_Fmac` provenant de `Axpy`)
Soient les réels a_0, x_0 et y_0. Soient les nombres flottants représentables a, x et y. On définit le nombre flottant représentable u comme $u = \circ(a \times x + y)$. Si

$$|y_0 - y| + |a_0 \times x_0 - a \times x| < \frac{\mathrm{ulp}\,(u^*)}{2},$$

alors $u = \square(a_0 \times x_0 + y_0)$.

La preuve est très similaire à celle de 60. Contrairement à celui-ci, ce théorème ne comporte qu'une unique condition : c'est logique car il est ici inutile de compenser l'erreur faite dans le calcul de $\circ(a \times x)$. Seules les erreurs précédentes sur les entrées sont importantes pour garantir la fidélité, les valeurs respectives comparées de $a \times x$ et y n'ont pas d'importance. Je ne peux tout de même pas garantir un arrondi correct à cause précisément de ces petites erreurs. L'utilisation d'un FMA est ici décisive car elle permet de se débarrasser de l'hypothèse principale des théorèmes précédents.

De façon similaire, je souhaite un théorème dépendant des entrées a, x et y et non de la sortie u.

Théorème 66 (`Axpy_opt_Fmac` provenant de `Axpy`)
Soient les réels a_0, x_0 et y_0. Soient les nombres flottants représentables a, x et y. On définit le nombre flottant représentable u comme $u = \circ(a \times x + y)$. Si

$$|y_0 - y| + |a_0 \times x_0 - a \times x| \leq 2^{-1-p}\left(1 - 2^{1-p}\right) \times |a \times x + y| - \frac{3 \times 2^{-E_i - p - 2}}{1 - 2^{-2p}},$$

alors $u = \square(a_0 \times x_0 + y_0)$.

Énoncé Coq
```
Theorem Axpy_opt_Fmac :
 forall (a1 x1 y1 : R) (a x y u : float),
 (Fbounded b a) -> (Fbounded b x) -> (Fbounded b y) -> (Fbounded b u) ->
 (Closest b radix (a * x + y) u) ->
 (Fcanonic radix b u) ->
 (Rabs (y1 - y) + Rabs (a1 * x1 - a * x) <
   (Rabs (a * x + y)) * (powerRZ radix (Zpred (- precision)) *
   (1 - powerRZ radix (Zsucc (- precision)))) -
   powerRZ radix (Zpred (Zpred (- dExp b))) *
   (3 * / (powerRZ radix precision - powerRZ radix (- precision))))%R ->
      (MinOrMax radix b (a1 * x1 + y1) u).
```

9.5 Application à l'évaluation polynomiale de Horner

Les critères définis dans le paragraphe précédent peuvent être utilisés pour tester automatiquement la fidélité du calcul d'un polynôme qu'il soit exact ou une approximation d'une fonction plus complexe : les conditions des théorèmes 62 et 66 peuvent être facilement vérifiées.

M. Daumas a écrit des telles routines en Maple, C et Java. Elles sont disponibles sur Internet sous la licence LGPL à l'adresse `http://perso.ens-lyon.fr/marc.daumas/SoftArith/`. Elles permettent à tout programme de savoir à l'avance que des évaluations polynomiales sont fidèles. Ces programmes sont très simples, à part celui en Java, car Java ne permet pas d'accéder aux modes d'arrondi, il a donc fallu changer les formules en ajoutant des ulps ici et là pour garantir que les conditions testées sont suffisantes [KD98].

9.6 Exemples

Après la présentation des conditions pour obtenir une implantation fidèle de la méthode de Horner, il reste à voir si ces conditions sont applicables en réalité.

9.6.1 Approximation polynomiale de l'exponentielle

Dans cet exemple, je recherche une approximation l'exponentielle de x avec x dans l'intervalle $\left[-2^{-4};2^{-4}\right]$. Je force le polynôme à commencer par $1+x+x^2/2$ et les paramètres sont des nombres flottants double précision fournis par Maple :

$$P(x) = 1 + x + \frac{1}{2}\,x^2 + \frac{6004799503158175}{36028797018963968}\,x^3 + \frac{6004799503152767}{144115188075855872}\,x^4$$

$$+ \frac{4803839629518891}{576460752303423488}\,x^5 + \frac{1601279914265145}{1152921504606846976}\,x^6$$

$$+ \frac{3660108472028231}{18446744073709551616}\,x^7 + \frac{7318367436494265}{295147905179352825856}\,x^8.$$

L'erreur de méthode $|\exp(x) - P(x)|/|x|$ est alors majorée par

$$\frac{5509901405496691}{81129638414606681695789005144064}.$$

Un appel aux fonctions Maple garantit alors que l'évaluation de ce polynôme est fidèle sur tout l'intervalle donné. Je peux alors facilement vérifier que l'évaluation est fidèle sur $\left[-2^{-3}; 2^{-3}\right]$ mais le critère échoue sur $\left[-2^{-2}; 2^{-2}\right]$.

9.6.2 Un polynôme dû à Fike

Le polynôme suivant apparaît dans [Fik67] où il est utilisé comme approximation en simple précision de 2^x sur l'intervalle $[-1/16; 0]$. Les vrais coefficients, après arrondi en simple précision, sont :

$$P(x) = 1 + \frac{1453635}{2097152}\,x + \frac{1007583}{4194304}\,x^2 + \frac{14899271}{268435456}\,x^3 + \frac{10327375}{1073741824}\,x^4$$

$$+ \frac{11451013}{8589934592}\,x^5 + \frac{5179279}{34359738368}\,x^6.$$

L'erreur de méthode $|2^x - P(x)|/|x|$ est alors majorée par

$$\frac{8577801}{4503599627370496}$$

et un appel aux fonctions Maple garantit que l'évaluation de ce polynôme est fidèle sur l'intervalle $[-1/16; 0]$.

Ainsi, j'ai démontré des conditions garantissant une évaluation fidèle d'une multiplication puis addition et par extension une évaluation fidèle d'un polynôme appliqué à un petit argument. L'intérêt réside d'une part dans la simplicité des conditions, et d'autre part dans leur applicabilité : les polynômes réels utilisés pour l'évaluation de fonctions élémentaires remplissent ces conditions.

Chapitre 10

Réduction d'arguments

There will always be a small but steady demand for error-analysts to...
expose bad algorithms' big errors and, more important,
supplant bad algorithms with provably good ones.
William Kahan, Interval Arithmetic Options in the "Proposed IEEE Floating Point Arithmetic Standard" (1980)

La réduction d'arguments consiste, pour un argument x, à obtenir $x^* = x - k \times C$ où C est une constante connue telle que π ou $\ln(2)$ et k un entier. Le but est d'obtenir un $|x^*|$ petit de façon à pouvoir calculer préférentiellement sur x^* que sur x. Ce chapitre montre une méthode usuelle, une nouvelle méthode et ses avantages, ainsi que les preuves formelles de ceux-ci et des conditions d'applications des algorithmes.

Les méthodes existantes pour calculer les fonctions élémentaires se basent pour la plupart sur une réduction d'argument, ce qui explique la constante recherche d'une méthode efficace et précise [CW80, Mar90, Ng92, Mul97, Mar00]. Étant donné un argument x, on veut calculer un argument réduit x^* qui est dans un intervalle donné. Après avoir calculé à partir de x^*, il faut pouvoir reconstruire le résultat de la fonction sur x.

L'argument réduit x^* vaut en général $x - kC$ où C est une constante donnée, typiquement $\ln(2)$ ou un multiple de π. L'entier k est choisi de façon à minimiser x^* pour qu'il soit dans un intervalle cible, typiquement légèrement supérieur à $\left[-\frac{\ln(2)}{2}; \frac{\ln(2)}{2}\right]$ ou $\left[-\frac{\pi}{2}; \frac{\pi}{2}\right]$. Le calcul complet de $f(x)$ suit alors les trois étapes suivantes :

- réduction d'argument : il faut calculer k puis x^*.
- évaluation : il faut calculer effectivement le résultat de la fonction cible g sur x^* : souvent $f = g$, mais le calcul du cosinus se fait parfois par l'intermédiaire du sinus ou *vice versa*. Cela peut se faire par l'intermédiaire d'une évaluation polynomiale (voir le chapitre 9 ainsi que [Lin76]), d'une méthode à base de tables précalculées ou par une méthode mixte [Tan91, WG95, Mul97] ou en passant par une autre fonction [Bri28, Vol59, PRN80, Val88, Ahr99].

– reconstruction de la fonction initiale sur l'argument initial : ce calcul est généralement simple et exact : il s'agit de multiplier par une puissance de 2, d'opposer le résultat...

Voici les trois étapes décrites par deux exemples. Ainsi, pour le cosinus de 3 :

Réduction d'argument :
$$\begin{cases} 3 \approx \pi - 0{,}141\ 59 = \pi - u \\ \text{avec } u \in \left[0; \frac{\pi}{4}\right] \end{cases}$$

Évaluation polynomiale :
$$\begin{cases} P(u) \text{ polynôme approchant } \cos(u) \text{ pour } u \in \left[0; \frac{\pi}{4}\right] \\ P(0{,}141\ 59) \approx 0{,}989\ 993 \end{cases}$$

Reconstruction :
$$\begin{cases} \cos(\pi - u) = -\cos(u) \\ \begin{array}{rcl} \cos(3) & \approx & \cos(\pi - 0{,}141\ 59) \\ & = & -\cos(0{,}141\ 59) \\ & \approx & -P(0{,}141\ 59) \\ & \approx & -0{,}989\ 993 \end{array} \end{cases}$$

De même pour calculer l'exponentielle de 3 :

Réduction d'argument :
$$\begin{cases} 3 \approx 4 \times \ln(2) + 0{,}227\ 41 = 4 \times \ln(2) + u \\ \text{avec } u \in [0; \ln(2)] \end{cases}$$

Évaluation polynomiale :
$$\begin{cases} P(u) \text{ polynôme approchant } \exp(u) \text{ pour } u \in [0; \ln(2)] \\ P(0{,}227\ 41) \approx 1{,}255\ 34 \end{cases}$$

Reconstruction :
$$\begin{cases} \exp(k \times \ln(2) + u) = 2^k \times \exp(u) \\ \begin{array}{rcl} \exp(3) & \approx & \exp(4\ln(2) + 0{,}227\ 41) \\ & = & 2^4 \exp(0{,}227\ 41) \\ & \approx & 2^4 \times P(0{,}227\ 41) \\ & \approx & 2^4 \times 1{,}255\ 34 \approx 20{,}085 \end{array} \end{cases}$$

Bien sûr, ces exemples sont simplifiés : on fait le plus souvent une étape supplémentaire dans la réduction d'arguments de façon à réduire x en une petite valeur qui vérifiera les hypothèses du chapitre 9. Nous considérons désormais un nombre flottant x et une constante réelle C. Nous voulons obtenir $x - k \times C$ dans un petit intervalle cible du type $[0; \varepsilon]$ ou $[-\varepsilon; \varepsilon]$.

10.1 Méthode de Cody & Waite

La méthode de Cody & Waite [CW80, Cod82] consiste à prendre une première approximation C_1 de C sur moins de bits que la précision disponible. Le nombre flottant C_2 est alors une approximation de $C - C_1$. On calcule k, l'entier le plus proche de $\frac{x}{C}$ puis x^* par :

$$\begin{array}{rcl} k & = & \left\lceil \dfrac{x}{C} \right\rfloor \\ x^* & = & ((x \ominus (k \otimes C_1)) \ominus (k \otimes C_2)) \\ & = & \circ(x - k \times C_1 - \circ(k \times C_2)) \end{array}$$

Si x est suffisamment petit, k l'est aussi et $k \times C_1$ est représentable : la multiplication est alors exacte. De plus, la soustraction $x - \circ(k \times C_1)$ est alors également exacte puisque x et $k \times C_1$ sont proches. La seule erreur commise est alors celle faite lors du dernier calcul $(x - k \times C_1) - k \times C_2$ car l'erreur faite sur le calcul de $k \times C_2$ est négligeable. L'algorithme complet dans ce cas est représenté par la figure 10.1 : la partie hachurée correspond à l'annulation des

premiers chiffres de x par $k \times C_1$ lors du calcul exact. Les parties grisées correspondent aux erreurs d'arrondi.

Malheureusement, ce scénario idyllique n'est pas toujours vérifié. En effet, lorsque l'argument x est trop grand, k occupe beaucoup de bits. Pour garantir que $k \times C_1$ est toujours représentable, il faut alors imposer que C_1 soit sur très peu de bits, comme le montre la figure 10.2. La conséquence est alors une annulation moindre que dans le scénario précédent. Le résultat final est alors bien moins précis car $k \times C_2$ est bien plus grand et l'erreur relative sur le résultat final est donc bien plus importante.

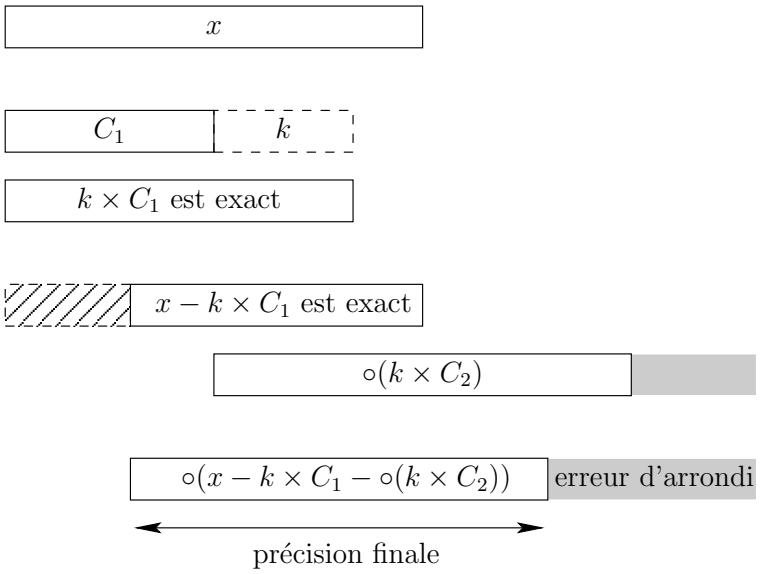


FIG. 10.1 – Réduction de Cody & Waite – cas usuel.

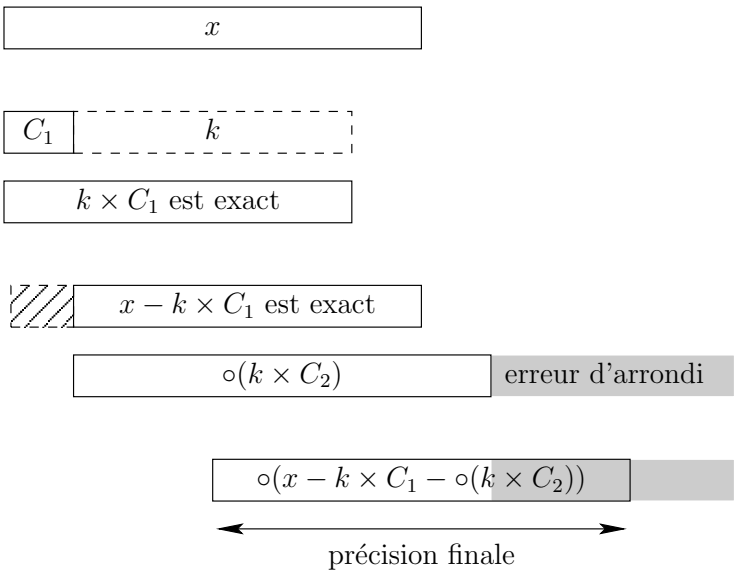


FIG. 10.2 – Réduction de Cody & Waite – cas à problème.

10.2 Notre méthode

Le but ici est de pouvoir garantir que même les grands x sont correctement réduits. Pour cela, nous tirons parti du FMA (voir la section 9.4).

Nous considérons C_1 d'une précision légèrement inférieure à celle avec laquelle nous travaillons (typiquement 2 ou 3 bits de moins). Le nombre flottant C_2 est alors une approximation de $C - C_1$. Étant donné x, on calcule :

$$\begin{aligned} k &= \left\lceil \frac{x}{C} \right\rfloor \\ x^* &= \circ(\circ(x - k \times C_1) - k \times C_2) \\ &= \circ(x - k \times C_1 - k \times C_2) \end{aligned}$$

Le calcul de $x - k \times C_1$ est en effet exact. Il n'est pas exact car nous avons choisi un C_1 sur peu de bits, mais parce que x et $k \times C_1$ sont proches et que la valeur $x - k \times C_1$ est exactement représentable dans le format flottant utilisé à cause de l'annulation importante des chiffres de x en soustrayant $k \times C_1$. Or, dans ce cas, le FMA renvoie cette valeur exacte. L'utilisation du FMA est décisive : en effet, le calcul de $x - \circ(k \times C_1)$ donnerait un résultat tout autre. Cet algorithme est décrit par la figure 10.3. Le fait que $x - k \times C_1$ soit représentable avait été noté dans [HKST99] sans plus de précisions.

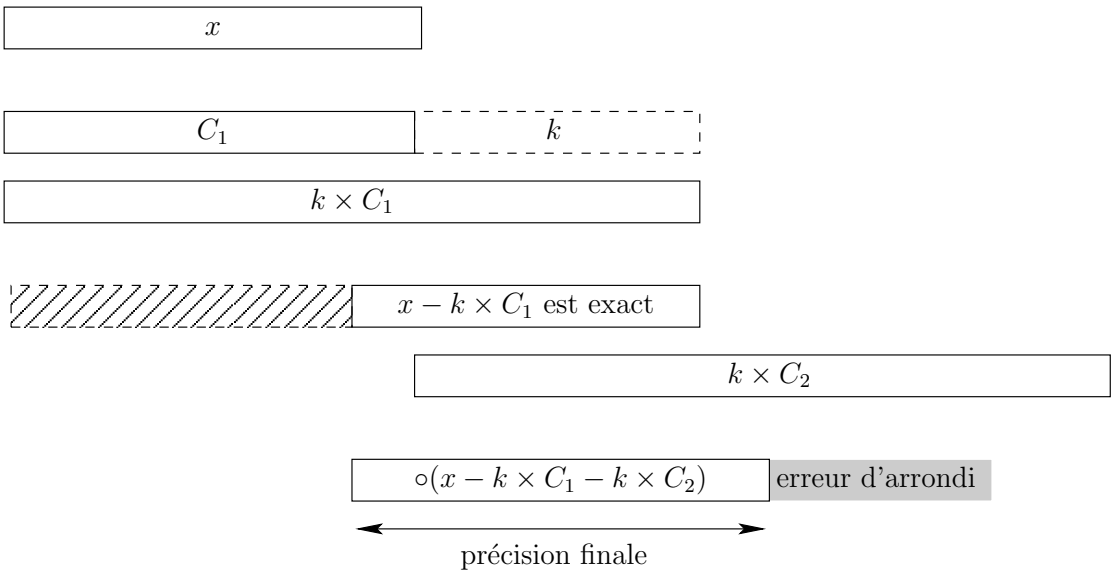


Fig. 10.3 – Notre réduction d'argument.

Il reste à trouver des hypothèses suffisant à garantir que la valeur $x - k \times C_1$ est représentable. Nous commençons par résoudre le problème du calcul de k où nous garantissons un algorithme classique dans la section 10.3. Les sections suivantes sont consacrées aux possibilités pour garantir la représentabilité de $x - kC_1$. Il faut pour cela calculer et choisir C_1 de façon judicieuse. Nous nous plaçons ici uniquement en base 2.

10.3 Calcul de k/z

Jusqu'à maintenant, nous avions défini k comme étant $k = \left\lceil \frac{x}{C} \right\rfloor$. Si k était calculé ainsi, on serait sûr que $x - kC_1$ est représentable. Mais ce calcul coûte cher à cause de la division. C'est pourquoi ce calcul se fait souvent ainsi : si R est une approximation de l'inverse de C, nous définissons k comme $\lceil xR \rfloor$.

Mais la définition de k est ici plus générale : pour un N donné, nous utilisons z, le nombre flottant le plus proche de xR dont l'exposant est $-N$. Nous pourrions également affirmer de façon équivalente l'égalité suivante sur les entiers : $z2^N = \left\lceil xR2^N \right\rfloor$. Nous nous ramenons donc au cas et au k usuels en prenant $N = 0$.

Le calcul de ce z peut se faire de façon efficace grâce à un FMA par

$$z = \circ(\circ(xR + \sigma) - \sigma) = \circ(xR + \sigma) - \sigma$$

où σ est une constante connue pré-calculée, ici $3 \cdot 2^{p-N-2}$. Cette technique est adaptée de [Mar00], qui attribue cette idée à C. Roothaan lors de son travail pour la bibliothèque mathématique sur les vecteurs pour l'Itanium. Nous voulons donc appliquer cette technique pour obtenir un z qui remplit les hypothèses nécessaires à l'application des théorèmes suivants.

Lemme 67 (`arg_reduct_exists_k_zH` provenant de `FArgReduct2`)
Soient x, R et z des nombres flottants et N un entier relatif avec $N \leq E_i$. On suppose que

1. $p > 3$,
2. x est un nombre flottant positif représentable sur p bits,
3. R est un nombre flottant strictement positif normalisé sur p bits,
4. $z = \circ\left(\circ\left(3 \cdot 2^{p-N-2} + xR\right) - 3 \cdot 2^{p-N-2}\right)$,
5. $z \geq 2^{1-N}$,
6. $xR \leq 2^{p-N-2} - 2^{-N}$,

alors il existe un entier ℓ tel que $2 \leq \ell \leq p - 2$, que $z2^N$ est un entier positif sur ℓ bits supérieur à $2^{\ell-1}$ et enfin que $|xR - z| \leq 2^{-N-1}$.

Les hypothèses ont été groupées par « thème ». L'introduction et les hypothèses 1 à 3 concernent les données du problème : il faut une précision pas trop limitée, un argument à réduire x et l'inverse de la constante R. L'hypothèse 4 est l'algorithme de calcul de z qui est exactement celui expliqué avec une constante $\sigma = 3 \cdot 2^{p-N-2}$. Les hypothèses 5 et 6 sont des inégalités nécessaires à la preuve pour des raisons techniques mais dont l'explication est assez simple. Pour garantir que z est sur au moins 2 bits, c'est-à-dire que $\ell \geq 2$, il faut que $z \geq 2^{1-N}$. Les cas où cette hypothèse n'est pas remplie sont discutés aux sections suivantes. La dernière hypothèse correspond au fait que x ne doit pas être trop grand, sinon la technique ne fonctionne pas : il faut que σ soit prépondérant par rapport à xR. La technique est également reprise et expliquée par la figure 10.4.

En résumé, si z est calculé comme expliqué et que x n'est pas trop grand, alors ce calcul donne un z convenable. Par exemple, pour les fonctions trigonométriques en double précision, pour $C = 2\pi$ et $N = 0$, nous pouvons aller jusqu'à $|x| \leq 1{,}5 \cdot 2^{53}$. En précision double

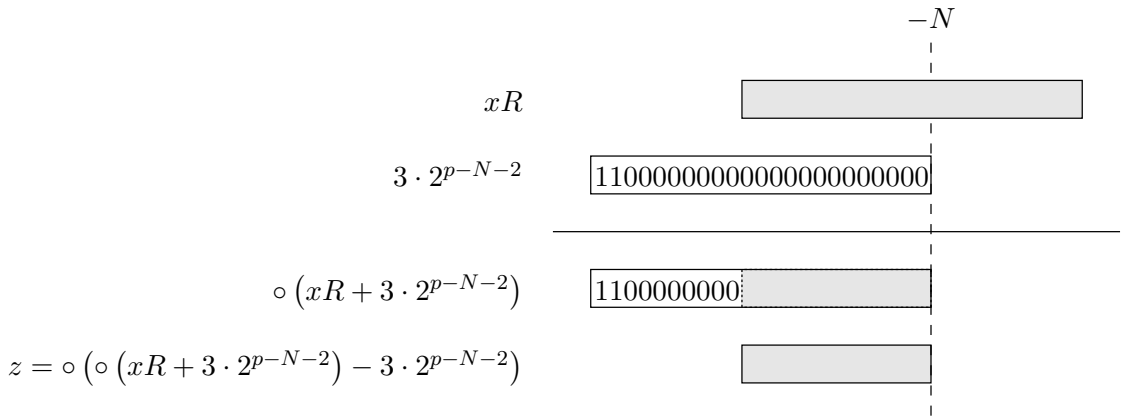


FIG. 10.4 – Algorithme de calcul de k/z.

étendue, nous pouvons donc largement aller jusqu'à $|x| \leq 2^{63}$ qui est la limite de l'argument autorisé pour les instructions Intel `FSIN`, `FCOS`, `FPTAN` et `FSINCOS` [Int96]. Pour l'exponentielle en double précision, pour $C = \ln 2$ et $N = 0$, nous pouvons aller jusqu'à $|x| \leq 1{,}3 \cdot 2^{50}$, ce qui dépasse largement la valeur maximale de x qui crée un dépassement de capacité.

10.4 Premier lot d'hypothèses

L'idée est d'utiliser le théorème 9 page 36. En effet, x et $k \times C_1$ peuvent être considérés comme des nombres flottants sur une précision étendue et s'ils sont suffisamment proches, alors la soustraction tient sur une précision réduite, ici la précision de travail. C'est pourquoi nous choisissons C_1 sur un peu moins de bits que la précision de travail de façon à limiter la précision étendue. Toute ces idées permettent de valider le lemme suivant.

Lemme 68 (`Fmac_arg_reduct_correct1` provenant de `FArgReduct2`)
Soient x, R, C_1 et z des nombres flottants et q, ℓ et N des entiers relatifs avec $N \leq E_i$. On suppose que

1. $p > 3$,
2. x est représentable sur p bits,
3. R est un nombre flottant normalisé sur p bits,
4. C_1 est un nombre flottant strictement positif, normalisé sur $p - q$ bits, qui n'est pas une puissance de 2 et tel que $C_1 = \circ_{p-q}(\frac{1}{R})$,
5. $z2^N$ est un entier positif sur ℓ bits supérieur à $2^{\ell-1}$,
6. $2 \leq \ell \leq p - 1$,
7. $|xR - z| \leq 2^{-N-1}$,
8. $R \leq 2^{q-p+E_i}$ et $C_1 \geq 2^{N+p-E_i}$,
9. $2 \leq q < p - 1$,
10. $q \leq \ell$,

alors $x - zC_1$ est représentable sur p bits.

Ici aussi, les hypothèses ont été groupées par « thème ». L'introduction et les hypothèses 1 à 4 concernent les données du problème : il faut une précision pas trop limitée, un argument à réduire x, une constante C_1 et son inverse R qui vérifient des propriétés raisonnables (la constante est sur moins de p bits et n'est pas une puissance de 2).

Les hypothèses 5 à 7 concernent z. Je n'explique pas comment le calculer mais je donne les hypothèses à remplir pour être acceptable, qui sont exactement celles du lemme 67.

Les hypothèses 8 et 9 sont des inégalités nécessaires à la preuve pour des raisons techniques mais dont l'explication est assez simple. Il faut tout d'abord que la constante soit dans un domaine raisonnable de façon à ce qu'elle et son inverse soient assez loin du nombre flottant maximal et du seuil de dépassement de capacité inférieur. En pratique, ces inégalités sont très largement vérifiées par toutes les constantes usuelles. Il faut ensuite que q soit borné : $q < p - 1$ est nécessaire pour que la précision de C_1 qui est $p - q$ soit strictement supérieure à 1 ; le fait que $q \geq 2$ est nécessaire vu la façon dont la preuve est pensée, il faut réduire la précision de C_1 d'au moins 2 pour que la précision de zC_1 ne soit pas trop grande et que le théorème 9 soit applicable.

L'hypothèse 10 est bien plus intéressante. Elle est venue naturellement dans la preuve et pourtant sa signification n'est pas simple. En effet, cette hypothèse n'est pas toujours vérifiée : si on considère les valeurs naturelles $q = 2$ et $N = 0$, alors l'unique cas où $\ell = 1$ qui est $z = \lceil xR \rfloor = 1$ ne rentre pas dans les hypothèses du théorème 68. Nous avons en fait isolé le cas difficile de ce problème. En effet, dans le cas général, l'algorithme facile

$$R = \circ_p\left(\frac{1}{C}\right) \qquad \text{et} \qquad C_1 = \circ_{p-2}\left(\frac{1}{R}\right)$$

fonctionne sans problème. Mais les cas où $\ell < q$ correspondent aux cas compliqués où les arguments précédents ne fonctionnent plus. Il pourrait être intéressant de tester ce cas particulier sur les processeurs disposant du FMA et qui l'utilisent pour la réduction d'arguments. Le théorème précédent couvre la très grande majorité des cas et il ne reste que quelques cas particuliers à regarder dans le détail pour garantir l'algorithme sous des hypothèses additionnelles.

10.5 Étude des cas particuliers et solutions possibles

Nous avons proposé deux solutions différentes pour régler les cas particuliers non couverts par le théorème précédent.

10.5.1 Solution 1 : $RC_1 \leq 1$

La première solution consiste à obliger $RC_1 \leq 1$. Cette restriction couvre les cas particuliers difficiles. Cela peut sembler une hypothèse très restrictive mais en fait, elle est assez facile à remplir. Il suffit de modifier légèrement R, en lui ajoutant ou lui soustrayant 1 ou 2 ulps et de tester chaque valeur de C_1 correspondante jusqu'à en obtenir une qui vérifie cette hypothèse. Il n'y a en effet aucune obligation pour R d'être exactement l'arrondi de l'inverse de la constante. Une légère modification, à condition d'en tenir compte ensuite n'est pas gênante.

Ce théorème est une simple application des théorèmes 67 et 68 précédents. C'est pourquoi se retrouve l'union de leur hypothèses pour pouvoir valider ce résultat. L'ajout de l'hypothèse 9 permet de garantir les cas particuliers.

Lemme 69 (`Fmac_arg_reduct_correct2` provenant de `FArgReduct2`)
Soient x, R, C_1 et z des nombres flottants et q et N des entiers relatifs avec $N \leq E_i$. On suppose que

1. $p > 3$,
2. x est un nombre flottant positif représentable sur p bits,
3. R est un nombre flottant normalisé sur p bits,
4. C_1 est un nombre flottant strictement positif, normalisé sur $p-q$ bits, qui n'est pas une puissance de 2 et tel que $C_1 = \circ_{p-q}(\frac{1}{R})$,
5. $z = \circ\left(\circ\left(3 \cdot 2^{p-N-2} + xR\right) - 3 \cdot 2^{p-N-2}\right)$,
6. $R \leq 2^{q-p+E_i}$ et $C_1 \geq 2^{N+p-E_i}$,
7. $xR \leq 2^{p-N-2} - 2^{-N}$,
8. $2 \leq q < p-1$,
9. $RC_1 \leq 1$

alors $x - zC_1$ est représentable sur p bits.

10.5.2 Solution 2 : $q = 2$

La seconde solution consiste à étudier de très près le cas $q = 2$. En effet, nous avons alors un unique cas particulier qui est lorsque $z = 2^{-N}$. Or ce cas est plus facile que le cas général car on calcule $x - C_1 2^{-N}$ et je peux donc appliquer le théorème de Sterbenz classique puisque zC_1 a la précision de C_1. Alors, le seul cas à problème vient des valeurs de xR qui sont proches de 2^{-N-1}, les autres cas étant résolus facilement.

Le dernier argument vient du fait que C_1 est loin d'une puissance de 2 : en effet, c'est un nombre flottant sur $p-2$ bits qui n'est pas une puissance de 2, il est donc à au moins 4 ulps d'une puissance de la base. Je peux alors majorer la mantisse de C_1 et appliquer le théorème de Sterbenz classique.

Théorème 70 (`Fmac_arg_reduct_correct3` provenant de `FArgReduct2`)
Soient x, R, C_1 et z des nombres flottants et N un entier relatif avec $N \leq E_i$. On suppose que

1. $p > 3$,
2. x est un nombre flottant positif représentable sur p bits,
3. R est un nombre flottant normalisé sur p bits,
4. C_1 est un nombre flottant strictement positif, normalisé sur $p-2$ bits, qui n'est pas une puissance de 2 et tel que $C_1 = \circ_{p-2}(\frac{1}{R})$,
5. $z = \circ\left(\circ\left(3 \cdot 2^{p-N-2} + xR\right) - 3 \cdot 2^{p-N-2}\right)$,
6. $R \leq 2^{2-p+E_i}$ et $C_1 \geq 2^{p+1-E_i+\max(N+1,0)}$,
7. $xR \leq 2^{p-N-2} - 2^{-N}$,

alors $x - zC_1$ est représentable sur p bits.

Énoncé Coq

```
Hypothesis precMoreThanThree : 3 < prec.
Hypothesis xCanonic : Fcanonic radix b x.
Hypothesis xPos : (0 <= x)%R.

Hypothesis alphaNormal : Fnormal radix b alpha.
Hypothesis gammaNormal : Fnormal radix bmoinsq gamma.
Hypothesis alphaPos : (0 < alpha)%R.
Hypothesis gammaPos : (0 < gamma)%R.
Hypothesis gamma_not_pow_2: forall e:Z, FtoRradix gamma <> powerRZ radix e.
Hypothesis gammaDef : Closest bmoinsq radix (/ alpha) gamma.

Hypothesis uDef : Closest b radix
     (3%nat * powerRZ radix (Zpred (Zpred (prec - N))) + x * alpha) u.
Hypothesis zH1Def : Closest b radix
     (u - 3%nat * powerRZ radix (Zpred (Zpred (prec - N)))) zH1.

Hypothesis N_not_too_big : (N <= dExp b)%Z.
Hypothesis exp_alpha_enough: (- dExp b <= 3-(Fexp alpha + (prec+prec)))%Z.
Hypothesis exp_gamma_enough: (- dExp b <= - N + Fexp gamma)%Z.
Hypothesis exp_gamma_enough2: (- dExp b <= Zpred (Zpred (Fexp gamma)))%Z.
Hypothesis exp_gamma_enough3: (- dExp b <= Fexp gamma -3 - N)%Z.
Hypothesis xalpha_small : (x * alpha <=
     powerRZ radix (Zpred (Zpred (prec - N))) - powerRZ radix (- N))%R.

Theorem Fmac_arg_reduct_correct3 :
 ex (fun f : float => FtoRradix f = (x - zH1 * gamma)%R /\ Fbounded b f).
```

Ce théorème est exactement le théorème 68 avec $q = 2$. Mais il est de plus valable dans tous les cas. La preuve dans le cas particulier $z = 2^{-N}$ n'a rien à voir avec les preuves précédentes. La démonstration complète se base d'une part sur le cas général et d'autre part sur le fait qu'il y a exactement un seul cas particulier connu précisément et pour lequel une démonstration spécifique existe.

Le tableau suivant donne les valeurs de R et de C_1 qui vérifient les hypothèses du théorème 70, selon le format flottant utilisé pour $C = \pi$.

Format	Single	Double	Double extended	Quad
R	$\frac{10680707}{33554432}$	$\frac{5734161139222659}{18014398509481984}$	$\frac{5871781006564002453}{18446744073709551616}$	$\frac{330551884414534967184149864106936 5}{103845937170696552570609926584401 92}$
C_1	$\frac{3294199}{1048576}$	$\frac{884279719003555}{281474976710656}$	$\frac{3622009729038561421}{1152921504606846976}$	$\frac{1019505104126898525104217885171767}{3245185536584267267831560205762 56}$

Le tableau suivant donne les valeurs de R et de C_1 qui vérifient les hypothèses du théorème 70, selon le format flottant utilisé pour $C = \ln(2)$.

Format	Single	Double	Double extended	Quad
R	$\frac{12102203}{8388608}$	$\frac{3248660424278399}{2251799813685248}$	$\frac{3326628274461080623}{2305843009213693952}$	$\frac{37454504643157696971616313650977 57}{25961484292674138142652481646100 48}$
C_1	$\frac{1453635}{2097152}$	$\frac{390207173010335}{562949953421312}$	$\frac{3196577161300663915}{4611686018427387904}$	$\frac{89975648203091924338510186296 1405}{12980742146337069071326240823050 24}$

10.6 Conclusion

Les théorèmes précédents montrent la correction de la technique présentée. Elle est bien plus rapide et précise que les techniques existantes, mais n'est bien sûr applicable que sur les quelques processeurs munis d'un FMA. Les conditions d'application des théorèmes sont en fait assez légères : il faut mettre à zéro les deux derniers bits de C_1 et garantir que l'argument à réduire n'est pas trop grand. Quant aux hypothèses qui bornent la valeur des constantes, elles sont en pratique toujours vérifiées. Ainsi en simple précision où les hypothèses sont les plus fortes, il faut que $2^{-123} \leq C_1 \leq 2^{126}$. Dans tous les formats utilisés et pour toutes les constantes utilisées, ces hypothèses sur la valeur de C_1 sont largement remplies.

La façon de faire cette preuve est également intéressante, l'assistant de preuve m'a naturellement aidée dans ce travail : le cas général ne peut résoudre tous les cas et l'hypothèse additionnelle m'a permis de circonscrire les cas difficiles avant de les résoudre. Or ces cas sont si rares que si le théorème général avait été testé, il est peu probable que l'un de ces cas soit apparu et donc une preuve presque juste et des tests intensifs n'auraient pas suffi à mettre le doigt sur ce problème. D'autre part, la séparation des cas permet de faire une preuve plus simple, mais aussi meilleure puisque les optimisations se font surtout sur les cas particuliers et les hypothèses additionnelles qui s'y attachent.

J'ai donc démontré la validité de la technique proposée pour la première étape de la réduction d'arguments pour les cas réels, ainsi qu'un critère sur le polynôme approchant la fonction élémentaire pour garantir que le calcul par la méthode de Horner est fidèle. Cela n'est pas encore suffisant pour garantir le processus complet mais je m'en approche. En effet, il reste à ajouter la seconde étape de la réduction d'arguments, qui est un travail en cours avec J.-M. Muller. En effet, il faut ajouter à $x - kC_1$ non seulement la valeur $-kC_2$, mais également l'erreur commise lors du calcul de $\circ((x - kC_1) - kC_2)$. On peut alors envisager d'ajouter $-kC_3$ et une partie de l'erreur commise lors du dernier calcul. Tout cela vise à pouvoir évaluer et améliorer la précision finale obtenue à volonté.

Chapitre 11

Conclusions & perspectives

Given the pace of technology, I propose we leave math to the machines and go play outside.
Calvin, Calvin and Hobbes par Bill Watterson (1992)

La bibliothèque Coq complète est téléchargeable sur internet à l'adresse suivant :

`http://lipforge.ens-lyon.fr/www/pff/`

Tous les fichiers Coq, ainsi que le graphe de dépendances sont également accessibles en version Web sans tout télécharger.

11.1 Travail effectué

Au cours de cette étude, j'ai travaillé sur divers aspects de l'arithmétique des ordinateurs et j'ai certifié de nombreuses propriétés très différentes les unes des autres. Par rapport aux chapeaux mexicains des figures 1.3 et 1.4 de l'introduction, je pense que le résultat est concluant. Quelques contributions importantes ont été représentées en figure 11.1 : la bibliothèque de base est importante, et les résultats supplémentaires sont nombreux et variés.

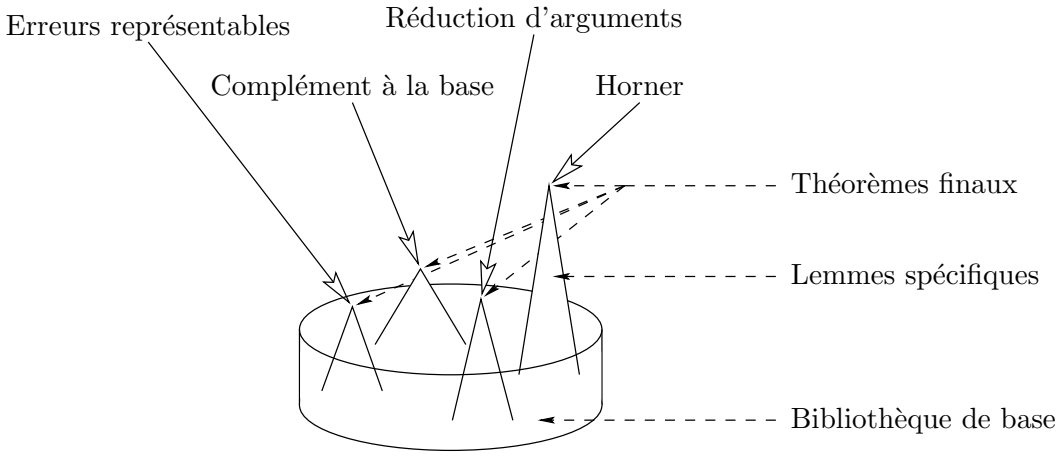


Fig. 11.1 – Mon chapeau mexicain.

Cette figure tente d'être réaliste : le pic des résultats sur les erreurs représentables (du chapitre 3) est bas car le nombre de lemmes intermédiaires est très faible, au contraire des

résultats concernant l'évaluation polynomiale par la méthode de Horner (du chapitre 9). Le pic des résultats correspondant au complément à la base et aux autres systèmes (du chapitre 6) n'est pas très haut mais est très large : il y a peu de lemmes intermédiaires mais beaucoup de résultats intéressants différents. J'avais d'ailleurs commencé par faire un trapèze plutôt qu'un triangle pour représenter ces résultats, mais j'ai tenu à garder une certaine homogénéité dans cette représentation.

Je trouve que les pics sont plus hauts que ce que je souhaite, c'est-à-dire que les lemmes spécifiques nécessaires avant de prouver un résultat donné sont souvent trop nombreux. Mais cela reste dans un domaine acceptable : la plupart des travaux sont sur un fichier Coq de moins de 1500 lignes. Mais ce reproche n'est peut-être qu'un souhait de facilité de ma part.

11.2 Caractéristiques des développements

Cette section analyse quelques données statistiques sur les différents développements en Coq. Ceux-ci sont séparés en trois catégories : la bibliothèque de base écrite surtout par L. Théry, mes développements présentés ici et la bibliothèque complète contenant l'ensemble des résultats. Cette bibliothèque complète contient également d'autres développements de L. Théry.

Le premier tableau en figure 11.2 donne les caractéristiques brutes de ces développements : le nombre de fichiers, le nombre de définitions, c'est-à-dire de `Definition`, `Record`, `Fixpoint` et `Inductive` le nombre de théorèmes, c'est-à-dire de `Theorem` et `Lemma`, le nombre de lignes et la taille mémoire en kilo-octets.

Développement	**Fichiers**	**Définitions**	**Théorèmes**	**Axiomes**	**Lignes**	**Taille**
Bibliothèque de base	32	87	750	0	15017	636 ko
Développements décrits ici	19	56	245	0	15582	692 ko
Bibliothèque complète	58	161	1127	0	35202	1508 ko

FIG. 11.2 – Tailles des développements.

La bibliothèque complète fait donc plus de 35000 lignes de Coq et comporte environ 1,5 méga-octet de données brutes (non compilées). Il faut insister sur le fait qu'aucun des développements ne comporte le moindre axiome. Il n'y a donc aucun risque de notre part de création de contradiction et d'obtenir une logique non consistante.

Le tableau suivant en figure 11.3 permet une première analyse de ces données. Une conclusion s'en dégage facilement : la bibliothèque de base et mes développements n'ont pas la même structure, même si la quantité de Coq est comparable. La bibliothèque de base est répartie sur plus de fichiers et contient plus de définitions et surtout bien plus de théorèmes. Les raisons en sont multiples et en voici quelques-unes :

– La bibliothèque de base contient les définitions initiales des structures de donnée et des opérateurs, d'où la proportion supérieure de définitions. Il est à noter que mes

Développement	Fichiers	Définitions	Théorèmes	Lignes	Taille
Bibliothèque de base	55%	54%	67%	43%	42%
Développements décrits ici	33%	35%	22%	44%	46%

FIG. 11.3 – Répartitions des tailles des développements.

développements ont tout de même nécessité une cinquantaine de définitions, ce qui n'est pas si faible en regard du total. Celles-ci me servent essentiellement à simplifier les énoncés de théorèmes.

– La bibliothèque de base doit contenir les premiers lemmes nécessaires à tout développement, elle doit donc contenir beaucoup de théorèmes qui sont censés être plus simples que ceux des développements suivants. Mes développements sont donc composés de moins de théorèmes, mais qui sont en comparaison plus « conséquents », c'est-à-dire que leur preuve est relativement plus longue en nombre de lignes : en moyenne 20 lignes pour la bibliothèque de base pour 64 pour mes développements.
– Ce n'est pas la même personne qui a écrit ces deux parties : la bibliothèque de base est surtout l'œuvre de L. Théry, avec l'aide de L. Rideau et de moi-même, et donc il n'est pas surprenant que le style de preuves soit différent. Mon style est de plutôt limiter le nombre de lemmes intermédiaires, quitte à avoir une preuve plus longue. Les chiffres montrent également que mes lignes de Coq sont en moyenne plus longues que celles de L. Théry.

Il faut ajouter à cela l'hétérogénéité des développements. Par exemple, pour les seuls résultats sur la réduction d'arguments, on trouve 41 théorèmes et aucune définition alors que pour les seuls résultats sur les systèmes génériques et le complément à 2, on trouve 51 théorèmes et 16 définitions. Les taux indiqués ne sont que des moyennes regroupant des réalités hétérogènes qui correspondent à des développements hétérogènes, alors que la bibliothèque de base est plus homogène.

11.3 Difficultés

Faire une preuve en Coq n'est jamais facile, mais je me suis heurtée à un certain nombre de difficultés spécifiques. Mon apprentissage initial de Coq a été rapide grâce à mon passage dans l'équipe Lemme à Sophia-Antipolis et à l'aide constante de L. Théry. Malgré cela, faire une preuve Coq reste une aventure semée d'embûches insoupçonnées, faite de beaucoup de sueur et de frustration et de quelques joies inattendues.

J'ai pu évaluer les points forts de Coq, notamment les inductions et l'utilisation de théorèmes déjà connus. Ces deux aspects sont exécutés par Coq bien mieux que par moi-même. Par contre, les calculs sur les réels, les coercions et changements de types entre entiers de Peano ➢ nat, entiers relatifs ➢ Z et réels ➢ R manquent beaucoup d'automatisation. De même pour les calculs sur les réels : ils sont en général bien gérés, mais dès que je rajoute les réels du type β^e, il faut indiquer et prouver chaque simplification comme par exemple $2^e + 2^e = 2^{e+1}$. Ces calculs devraient être simplifiés à l'avenir, mais ils restent pour l'instant difficiles et frustrants.

La conséquence immédiate est que certaines démonstrations ont été allégées : au lieu de prouver le résultat pour toute base, je ne l'ai prouvé que pour la base 2 (voir les chapitres 8, 9 et 10) bien que je sois convaincue de sa véracité pour toute base (après quelques modifications

légères). Mais les calculs sont trop lourds en toute base pour être faisable dans un temps raisonnable. En effet, la base 2 apporte des simplifications ; par exemple, le lemme 56 page 83 dit que, en base 2, si $f = \circ(z)$, alors

$$\frac{|z|}{1+2^{-p}} - \frac{2^{-E_i-1}}{1+2^{-p}} \leq |f| \leq \frac{|z|}{1-2^{-p}} + \frac{2^{-E_i-1}}{1-2^{-p}}.$$

Le théorème équivalent en base quelconque serait que si $f = \circ(z)$, alors

$$\frac{|z|}{1+\frac{\beta^{1-p}}{2}} - \frac{\beta^{-E_i}}{2+\beta^{1-p}} \leq |f| \leq \frac{|z|}{1-\frac{\beta^{1-p}}{2}} + \frac{\beta^{-E_i}}{2-\beta^{1-p}}.$$

La division par 2 correspondant au demi-ulp en cas d'arrondi au plus proche se simplifie en base 2, mais pas en base quelconque, ce qui ajoute une division à manipuler, ce qui est toujours désagréable en Coq.

Ces difficultés ont néanmoins été allégées par les différentes interfaces graphiques que j'ai utilisées, par ordre chronologique :

- `ctcoq`[1] [Ber99], basé sur Centaur, abandonné au profit de `pcoq`.
- `pcoq`[2] [ABRP01], écrit en Java et OCaml.
- `ProofGeneral`[3] [Asp00], qui est une sur-couche d'`emacs`.
- `coqide`[4], qui fait partie de la distribution de Coq depuis la version 8.0.

Une autre complication est l'utilisation des types dépendants pour représenter les tableaux de taille fixe du chapitre 5 concernant la norme IEEE-754. Cette notion complexe m'a semblée peu intuitive à utiliser et de plus elle alourdit nettement les énoncés de théorèmes.

Un autre obstacle important fut les nombreux changements de version. Il est en effet normal que faire une preuve Coq soit plus long et difficile que faire une preuve papier. Mais devoir régulièrement retoucher aux preuves Coq déjà faites semble inutile. La bibliothèque est en effet née sous la version 6.2 et a connu depuis les versions 6.3, 7.0, 7.1, 7.2, 7.3, 7.4 et est désormais en 8.0, soit 8 versions en moins de 4 ans.

Nous aurions pu mettre les théorèmes prouvés dans une version en axiomes de la bibliothèque utilisant la version suivante. Mais cela rendrait extrêmement dangereuse la retranscription des théorèmes d'une version à l'autre en cas de changement de syntaxe : la moindre erreur rendrait la logique inconsistante ! De plus, les nouvelles versions autorisent des preuves plus courtes et plus lisibles, permettant aux utilisateurs d'avoir des exemples simples dont ils peuvent s'inspirer.

Le problème n'est pas tant le changement de versions que l'incompatibilité entre versions ce qui le rend coûteux en temps. En effet, la tactique `auto` étant améliorée, elle résout plus de buts, ce qui peut faire échouer la preuve. La difficulté vient alors du fait que la preuve n'échoue pas à l'endroit où un but supplémentaire est résolu mais plus tard, entre deux preuves de sous-buts, ce qui nécessite une nouvelle étude et une nouvelle compréhension de la preuve de façon à supprimer les bonnes lignes. Le passage à la version 8.0 a ajouté de nombreux changements de noms sur mes théorèmes préférés sur les calculs réels. Cela rend harassantes les premières preuves utilisant la nouvelle version mais permet l'homogénéité de noms entre les théorèmes de calculs sur $\mathbb{R}$ et sur $\mathbb{Z}$.

[1] `http://www-sop.inria.fr/croap/ctcoq/ctcoq-fra.html`
[2] `http://www-sop.inria.fr/lemme/pcoq/`
[3] `http://proofgeneral.inf.ed.ac.uk/`
[4] `http://coq.inria.fr/`

11.4 Perspectives

11.4.1 Exceptions

La norme IEEE-754 définit 5 exceptions : opération invalide, division par zéro, dépassement de capacité supérieur, dépassement de capacité inférieur et inexactitude du résultat. Tout calcul qui lève une telle exception peut déclencher une trappe : le programme contenant le calcul ayant levé l'exception est suspendu et une autre fonction, qui dépend de l'exception et qui peut être redéfinie par l'utilisateur, va être exécutée avant que le programme initial ne continue.

Ce mécanisme oblige tout programme utilisant des calculs flottants à pouvoir être interrompu à tout moment et le flux d'exécution à pouvoir être redirigé en temps réel. On ne peut donc pas modéliser un calcul comme une simple fonction prenant l'opération, les arguments flottants et le mode d'arrondi et renvoyant un nombre flottant. Il faut aussi qu'un calcul flottant renvoie les exceptions levées et éventuellement exécute une fonction utilisateur selon l(es) exception(s) levée(s). L'utilisateur peut également tester les drapeaux à tout moment pour savoir si une exception vient d'être ou a été déclenchée.

Ce mécanisme est excessivement compliqué, même si certains en tirent profit pour faire des calculs [DL93]. Néanmoins, les exceptions sont en train d'être rediscutées dans la révision de la norme IEEE-754 [IEE04] de façon à changer ce fonctionnement compliqué et extrêmement contraignant pour les concepteurs de processeurs. Le mécanisme de drapeaux sera vraisemblablement modifié dans un proche avenir.

Au point de vue des preuves formelles, un certain nombre de travaux ont décrit ce mécanisme (voir le tableau de la figure 2.5 page 20). Mais aucune vraie preuve ne s'en sert : la formalisation existe, mais n'a jamais été vraiment utilisée. Ces raisons expliquent que la formalisation des exceptions ne soit pas faite ici : d'une part, il me manque une application réelle validant le modèle que je choisirais, d'autre part, j'attends la fin de la révision de façon à formaliser la nouvelle version des exceptions choisie par le comité de révision.

11.4.2 Automatisation

Une autre perspective est l'automatisation des preuves. Les travaux de ma thèse concernent des points particuliers difficiles identifiés comme tels. Il est donc logique d'y passer du temps. Mais on peut espérer que, lorsque le raisonnement suit un chemin préparé simple et que tout se passe bien, alors la preuve Coq puisse être écrite automatiquement par un programme. C'est un thème de la thèse de Guillaume Melquiond [DM04].

Un moyen est d'utiliser l'arithmétique d'intervalles de façon à avoir une garantie de l'erreur sur le résultat et de prouver en Coq chaque étape de calcul. Bien sûr, il est nécessaire d'être un peu plus fin que l'arithmétique d'intervalles et les améliorations actuelles contiennent la séparation en sous-intervalles par exemple. Les améliorations possibles sont très nombreuses mais très coûteuses à mettre en place, car il faut démontrer les résultats en Coq valides même lors de l'apparition de cas particuliers.

Pour l'instant, la version actuelle de l'« usine à preuves » s'intéresse à l'évaluation polynomiale par la méthode de Horner. Le fonctionnement est le suivant : on insère dans un formulaire Internet le format flottant utilisé, les valeurs de l'argument et des coefficients du polynôme avec des valeurs minimales et maximales et éventuellement des erreurs absolues ou relatives. Par exemple, je veux regarder $y = 1 + x + x^2/2 + x^3/6$ avec $x \in [0; 1]$ sachant que

l'erreur relative sur x et sur le calcul de 1/6 est inférieure à 1/2ulp. Le programme renvoie sur cet exemple la réponse sous la forme suivante (qui pourrait changer) :

```
Program:
s3 = a3
m2 = mul64(s3, x)
a2 = .5
s2 = add64(m2, a2)
m1 = mul64(s2, x)
a1 = 1
s1 = add64(m1, a1)
m0 = mul64(s1, x)
a0 = 1
s0 = add64(m0, a0)
y = s0

Property: REL(x, X)   in [-1b-53, 1b-53]  /\
          a3          in [.16666, .16667] /\
          REL(a3, A3) in [-1b-53, 1b-53]  /\
          x           in [0, 1] ->
             y in ? /\
             ABS(y, (((A3) * X + a2) * X + a1) * X + a0) in ? /\
             REL(y, (((A3) * X + a2) * X + a1) * X + a0) in ?

Result:
REL(y, ...) in [-1657107070431586193207586122165081320l5b-177 {-8.6505e-16},
                1657107070431586193207586122165081320l5b-177 {8.6505e-16}]
ABS(y, ...) in [-1657107070431586193207586122165081320l5b-177 {-8.6505e-16},
                1657107070431586193207586122165081320l5b-177 {8.6505e-16}]
y in [1 {1}, 750600876145005b-48 {2.66667}]
```

On a donc tout d'abord le programme en pseudo-assembleur, puis les hypothèses et les propriétés cherchées (avec des points d'interrogation). Ainsi, l'erreur sur x est inférieure à $1b-53 = 1 \times 2^{-53}$. Le résultat est alors un intervalle d'erreur sur le résultat y et un intervalle d'appartenance de y. Ici $y \in [1; 2{,}666\ 67]$ et son erreur relative est inférieure à $8{,}650\ 5\ 10^{-16}$. Il est possible de contraindre l'intervalle d'erreur du résultat et ainsi d'obtenir une bien meilleure erreur relative de 2 ulps soit 2^{-51}.

La nouvelle version améliorée de ce programme n'écrit pas encore la preuve Coq correspondante, mais ce n'est qu'une question de temps. L'intérêt est que l'on pourra ensuite vérifier cette preuve : même si le programme a fait une erreur, ce n'est pas grave car alors la preuve Coq échoue. Par contre, la génération et surtout la vérification de la preuve Coq seront plus longues que le simple appel au programme. Il faut donc un temps supplémentaire non négligeable pour vérifier la preuve conçue automatiquement. C'est évidemment une alternative bien plus sûre aux programmes en Maple/C/Java pour l'évaluation polynomiale par la méthode de Horner, grâce à l'interface accessible et aux preuves Coq générées.

11.4.3 Autres développements

Il faut évidemment continuer à faire des développements utilisant cette bibliothèque, car les preuves sur les nombres flottants doivent être les plus sûres possibles pour accroître la sécurité des calculs informatiques.

Un premier développement en cours est la seconde étape de la réduction d'arguments. Après avoir prouvé que $x - kC_1$ est exact, on calcule $\circ(x - kC_1 - kC_2)$ avec un FMA, mais celui-ci ne sera probablement pas exact. Je veux évaluer précisément l'erreur commise dans ce calcul, par exemple comme fait pour les autres opérations dans le chapitre 3. Cela permettrait d'obtenir une réduction d'argument plus précise en calculant par exemple $x - kC_1$ puis $x' = \circ(x - kC_1 - kC2)$ et enfin $x^* = x' + erreur(x') - kC_3$. Pouvoir calculer l'erreur d'un FMA permettrait avec un coût en nombre d'opérations supérieur, de faire une réduction d'arguments aussi précise que l'on veut. Ceci peut être intéressant en certains points précis, par exemple savoir si le sinus d'un nombre très proche d'un multiple de π est positif ou négatif. Bien sûr, il faut que l'algorithme de calcul de l'erreur soit efficace pour ne pas trop pénaliser ces cas particuliers.

Je cherche également d'autres théorèmes très généraux et originaux comme celui qui affirme que l'exposant du canonique est l'exposant le plus petit possible parmi les nombres flottants représentables d'une cohorte. Cette propriété est nouvelle car elle se base sur l'existence d'une cohorte ; elle est simple puisqu'il est clair que la normalisation déplace la mantisse à la maximiser et donc de façon à minimiser l'exposant ; elle est utile dans certaines preuves (voir notamment le chapitre 3).

D'autres développements sont envisagés ou en cours sur des algorithmes courts, mais dont la preuve est particulièrement fine ou technique. Je m'intéresse également à l'amélioration certaines bornes d'erreur où tout gain se traduit par une accélération effective en moyenne de l'algorithme.

Ensuite, plus la quantité de théorèmes disponible augmente, plus les preuves suivantes sont faciles. Cela aide d'autres utilisateurs à utiliser la bibliothèque. Or le fait que d'autres personnes utilisent la bibliothèque renforce la confiance qu'on peut lui accorder, en plus d'en démontrer l'intérêt et la généricité. C'est ce que font L. Fousse et P. Zimmermann [FZ03] pour prouver un algorithme difficile de sommation [DH03].

11.4.4 Accessibilité

Il faut continuer à prouver formellement des résultats d'arithmétique des ordinateurs de façon à ancrer cette pratique dans les habitudes de façon à garantir les algorithmes proposés. Il n'y a en effet plus de frein culturel puisque la principale différence avec la norme IEEE-754 est l'existence de cohortes de nombres flottants différents de valeurs égales. Or cette notion va bientôt être normalisée dans [IEE04] pour la base 10 et le vocabulaire associé (probablement cohorte) va être fixé.

D'autre part, les preuves formelles tentent de se rapprocher des utilisateurs potentiels par divers moyens. L'un est CYP (Color Your Proof) [Thé03] qui permet de transformer une preuve papier en preuve formelle. Pour que la preuve formelle n'échoue pas, il faut bien sûr que la preuve papier soit juste et bien détaillée, mais aussi que l'utilisateur ait colorié sa preuve, c'est-à-dire qu'il ait indiqué au logiciel le contenu sémantique de la preuve (séparer les sous-preuves, isoler les hypothèses et les conclusions, préciser les noms des théorèmes appliqués...)

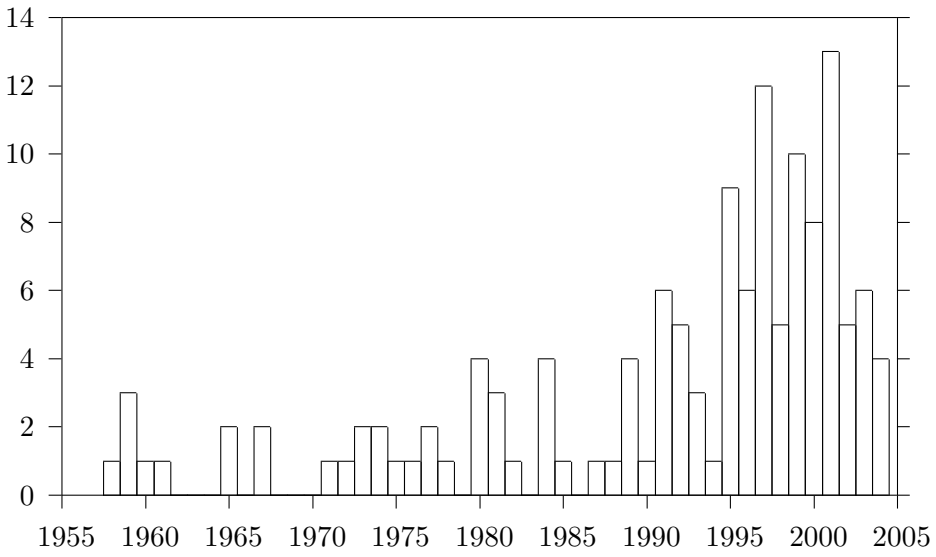
Cela permet une interaction entre l'assistant de preuve et une preuve compréhensible par tous et permet également de garantir que la preuve papier fournie est correcte et correspond bien à la démonstration faite à l'intérieur de l'assistant de preuves.

Enfin, le domaine est si sensible et si compliqué que les preuves formelles sont considérées comme un bienfait, même si elles semblent encore hors de portée à la plupart. L'automatisation d'une partie des preuves et l'augmentation du nombres de résultats disponibles devraient permettre de mettre les preuves formelles à la portée de ceux qui le souhaitent.

Bibliographie

The Fundamental Theorem of Algebra asserts that every polynomial equation over the complex field has a root. It is almost beneath the dignity of such a majestic theorem to mention that in fact it has precisely n roots.
J. H. Wilkinson, The Perfidious Polynomial (1984)

La figure suivante décrit le nombre de références bibliographiques en fonction de leur année de publication (les références [Bri28] et [Bal31] en sont exclues pour des raisons de lisibilité).



[ABRP01] Ahmed Amerkad, Yves Bertot, Laurence Rideau et Loïc Pottier : Mathematics and proof presentation in Pcoq. Dans *Proceedings of Proof Transformation and Presentation and Proof Complexities*, Siena, Italy, 2001. URL `http://www-sop.inria.fr/lemme/Laurence.Rideau/proof-pcoq.ps.gz`. [Page 104]

[Ahr99] Timm Ahrendt : Algorithms and data structures 3 - fast computations of the exponential function. *Lecture Notes in Computer Science*, 0(1563):302–312, 1999. [Page 91]

[AM59] Robert L. Ashenhurst et Nicholas Metropolis : Unnormalized floating point arithmetic. *Journal of the ACM*, 6(3):415–428, 1959. URL `http://delivery.acm.org/10.1145/330000/320996/p415-ashenhurst.pdf`. [Page 18]

[Asp00] David Aspinall : Software and formal methods tools - Proof General : A generic tool for proof development. *Lecture Notes in Computer Science*, 0(1785):38–42, 2000. [Page 104]

[Avi61] Algirdas Avižienis : Signed digit number representations for fast parallel arithmetic. *IRE Transactions on Electronic Computers*, 10:389–400, 1961. [Pages 11 et 58]

[Bai93] David H. Bailey : Algorithm 719, multiprecision translation and execution of Fortran programs. *ACM Transactions on Mathematical Software*, 19(3):288–319, 1993. URL `http://www.acm.org/pubs/articles/journals/toms/1993-19-3/p288-bailey/p288-bailey.pdf`. [Page 75]

[Bal31] Honoré de Balzac : *La Peau de chagrin*. Le Livre de Poche, 1831. [Pages 57 et 109]

[Bar84] Hendrik Pieter Barendregt : *The Lambda Calculus – Its Syntax and Semantics*, volume 103 de *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984. [Page 4]

[Bar89] Geoff Barrett : Formal methods applied to a floating-point number system. *IEEE Transactions on Software Engineering*, 15(5):611–621, 1989. URL `http://csdl.computer.org/dl/trans/ts/1989/05/e0611.pdf`. [Pages 5 et 45]

[BC04] Yves Bertot et Pierre Casteran : *Interactive Theorem Proving and Program Development. Coq'Art : the Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer Verlag, 2004. URL `http://www.labri.fr/Perso/~casteran/CoqArt/index.html`. [Page 4]

[Ber99] Yves Bertot : The ctcoq system : Design and architecture. *Formal Aspects of Computing*, 11(3):225–243, 1999. URL `http://www-sop.inria.fr/lemme/Yves.Bertot/hdr/RR-3540.ps.gz`. [Page 104]

[BP94] Dario Bini et Victor Y. Pan : *Polynomial and Matrix Computations : Fundamental Algorithms*. Birkhäuser, 1994. [Page 81]

[Bri28] Henry Briggs : *Arithmetica logarithmica*. Goudae, Excudebat Petrus Rammasenius, 1628. URL `http://gallica.bnf.fr/scripts/ConsultationTout.exe?O=N021044&E=0`. [Pages 91 et 109]

[BWKM91] Gerd Bohlender, Wolfgang Walter, Peter Kornerup et David W. Matula : Semantics for exact floating point operations. Dans Peter Kornerup et David Matula, éditeurs : *Proceedings of the 10th Symposium on Computer Arithmetic*, pages 22–26, Grenoble, France, 1991. [Pages 23, 24 et 32]

[Car59] John W. Carr : Error analysis in floating point arithmetic. *Communications of the ACM*, 2(5):10–16, 1959. URL `http://delivery.acm.org/10.1145/370000/368329/p10-carr.pdf`. [Page 18]

[Car95] Victor A. Carreño : Interpretation of IEEE-854 floating-point standard and definition in the HOL system. Rapport technique Technical Memorandum 110189, NASA Langley Research Center, 1995. URL `http://techreports.larc.nasa.gov/ltrs/PDF/NASA-95-tm110189.pdf`. [Pages 5, 32 et 45]

[CF58] Haskell B. Curry et R. Feys : *Combinatory Logic*, volume 1. North Holland, 1958. [Page 4]

[Che95] Jean-Marie Chesneaux : *L'arithmétique stochastique et le logiciel CADNA*. Habilitation à diriger des recherche, Université Paris VI, Paris, France, 1995. [Page 67]

[CK+84] William J. Cody, Richard Karpinski *et al.* : A proposed radix and word-length independent standard for floating point arithmetic. *IEEE Micro*, 4(4):86–100, 1984. [Pages 9, 32 et 55]

[CM95] Victor A. Carreño et Paul S. Miner : Specification of the IEEE-854 floating-point standard in HOL and PVS. Dans *1995 International Workshop on Higher Order Logic Theorem Proving and its Applications*, Aspen Grove, Utah, 1995. URL `http://shemesh.larc.nasa.gov/people/vac/publications/hug95.ps`. supplemental proceedings. [Page 5]

[Cod73] William J. Cody : Static and dynamic numerical characteristics of floating point arithmetic. *IEEE Transactions on Computers*, 22(6):598–601, 1973. [Page 11]

[Cod82] William J. Cody : Implementation and testing of function software. Dans P. C. Messina et A. Murli, éditeurs : *Problems and Methodologies in Mathematical Software Production*, numéro 42 de Lecture Notes in Computer Science, Berlin, 1982. Springer-Verlag. [Page 92]

[Coe95] Tim Coe : Inside the Pentium FDIV bug. *Dr. Dobb's Journal*, 20(4):129–135, 148, 1995. [Page 1]

[Col97] Robert R. Collins : Inside the Pentium II math bug. *Dr. Dobb's Journal*, 22(8): 52, 55–57, 1997. URL `http://www.ddj.com/articles/1997/9708/`. [Page 1]

[Coo78] Jerome T. Coonen : Specification for a proposed standard for floating point arithmetic. Memorandum ERL M78/72, University of California, Berkeley, 1978. [Pages 46 et 61]

[Coq04] The Coq Development Team, LogiCal Project, INRIA. *The Coq Proof Assistant : Reference Manual, Version 8.0*, 2004. URL `http://coq.inria.fr/doc/main.html`. [Page 4]

[Cow03] Michael F. Cowlishaw : decimal floating point : algorism for computers. Dans Jean-Claude Bajard et Michael Schulte, éditeurs : *Proceedings of the 16th Symposium on Computer Arithmetic*, pages 104–111, Santiago de Compostela, Spain, 2003. URL `http://csdl.computer.org/comp/proceedings/arith/2003/1894/00/1894toc.htm`. [Page 11]

[CSSW01] Michael F. Cowlishaw, Eric M. Schwarz, Ronald M. Smith et Charles F. Webb : A decimal floating point specification. Dans Neil Burgess et Luigi Ciminiera, éditeurs : *Proceedings of the 15th Symposium on Computer Arithmetic*, pages 147–154, Vail, Colorado, 2001. URL `http://csdl.computer.org/comp/proceedings/arith/2001/1150/00/1150toc.htm`. [Page 11]

[CVSG97] Michael P. Collins, Franck J. Vecchio, Robert G. Selby et Pawan R. Gupta : The failure of an offshore platform. *Concrete International*, 19(8):159–192, 1997. URL `http://www.concreteinternational.com/pages/featured_article.asp?ID=56`. [Page 1]

[CW80] William J. Cody et William Waite : *Software manual for elementary functions*. Prentice Hall, 1980. [Pages 91 et 92]

[Dau99] Marc Daumas : Multiplications of floating point expansions. Dans Israel Koren et Peter Kornerup, éditeurs : *Proceedings of the 14th Symposium on Computer Arithmetic*, pages 250–257, Adelaide, Australia, 1999. URL `http://perso.ens-lyon.fr/marc.daumas/SoftArith/Dau99.pdf`. [Page 76]

[Dau01] Marc Daumas : *Écrire les nombres autrement pour calculer plus vite*. Habilitation à diriger des recherches, Université Claude Bernard, Lyon, France, 2001. URL `ftp://ftp.ens-lyon.fr/pub/LIP/Rapports/HDR/HDR2001/HDR2001-02.ps.gz`. [Page 58]

[Dav72] Philip J. Davis : Fidelity in mathematical discourse : Is one and one really two ? *Amer. Math. Monthly*, 79:252–263, 1972. [Pages 1, 2 et 67]

[Dek71] Theodorus J. Dekker : A floating point technique for extending the available precision. *Numerische Mathematik*, 18(3):224–242, 1971. [Pages 23–25 et 76]

[Dem84] James Demmel : Underflow and the reliability of numerical software. *SIAM Journal on Scientific and Statistical Computing*, 5(4):887–919, 1984. [Pages 28 et 82]

[DF98] Marc Daumas et Claire Finot : Division of floating point expansions. Dans *IMACS-GAMM International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics*, pages 45–46, Budapest, Hungaria, 1998. URL `ftp://ftp.ens-lyon.fr/pub/LIP/Rapports/RR/RR1999/RR1999-03.ps.Z`. [Page 76]

[DF99] Marc Daumas et Claire Finot : Algorithm, proof and performances of a new division of floating point expansions. Research report 3771, Institut National de Recherche en Informatique et en Automatique, Le Chesnay, France, 1999. URL `ftp://ftp.inria.fr/INRIA/publication/publi-pdf/RR/RR-3771.pdf`. [Page 76]

[DH03] James Demmel et Yozo Hida : Accurate and efficient floating point summation. *SIAM Journal on Scientific Computing*, 25(4):1214–1248, décembre 2003. URL `http://epubs.siam.org/sam-bin/dbq/article/40762`. [Page 107]

[DL93] James W. Demmel et Xiaoye Li : Faster numerical algorithms via exception handling. Dans E. E. Swartzlander, M. J. Irwin et J. Jullien, éditeurs : *Proceedings of the 11th IEEE Symposium on Computer Arithmetic*, pages 234–241, Windsor, Canada, 1993. IEEE Computer Society Press, Los Alamitos, CA. URL `http://http.cs.berkeley.edu/~xiaoye/ieee.ps.gz`. [Page 105]

[DLP77] Richard A. DeMillo, Richard J. Lipton et Alan J. Perlis : Social processes and proofs of theorems and programs. Dans *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 206–214. ACM Press, 1977. URL `http://doi.acm.org/10.1145/512950.512970`. [Pages 2, 57 et 81]

[DM97a] Marc Daumas et David W. Matula : Recoders for partial compression and rounding. Research report 97-01, Laboratoire de l'Informatique du Parallélisme, Lyon, France, 1997. URL `ftp://ftp.ens-lyon.fr/pub/LIP/Rapports/RR/RR97/RR97-01.ps.Z`. [Page 58]

[DM97b] Marc Daumas et Jean-Michel Muller, éditeurs. *Qualité des calculs sur ordinateur : vers des arithmétiques plus fiables*. Masson, 1997. URL `http://perso.ens-lyon.fr/jean-michel.muller/livre_masson.html`. [Page 3]

[DM04] Marc Daumas et Guillaume Melquiond : Generating formally certified bounds on values and round-off errors. Dans *Real Numbers and computers*, Dagstuhl, Germany, 2004. [Page 105]

[Dow02] Gilles Dowek : Théories des types, 2002. URL `http://www.lix.polytechnique.fr/~dowek/Cours/theories_des_types.ps.gz`. Notes de cours du DEA : Sémantique, preuves et programmation. [Page 4]

[DRT01] Marc Daumas, Laurence Rideau et Laurent Théry : A generic library of floating-point numbers and its application to exact computing. Dans *14th International Conference on Theorem Proving in Higher Order Logics*, pages 169–184, Edinburgh, Scotland, 2001. URL `http://perso.ens-lyon.fr/marc.daumas/SoftArith/DauRidThe01.ps`. [Pages 17, 24, 25 et 64]

[Epp01] James F. Epperson : *An Introduction to Numerical Methods and Analysis*. John Wiley & Sons, 2001. [Page 81]

[Erc84] Miloš D. Ercegovac : On-line arithmetic : an overview. Dans *SPIE, Real Time Signal Processing VII*, pages 86–93, 1984. [Page 76]

[Fik67] C. T. Fike : Methods of evaluating polynomial approximations in function evaluation routines. *Communications of the ACM*, 10(3):175–178, 1967. URL `http://delivery.acm.org/10.1145/370000/363200/p175-fike.pdf`. [Page 89]

[FZ03] Laurent Fousse et Paul Zimmermann : Accurate summation : Towards a simpler and formal proof. Dans *Real Numbers and computers*, pages 97–108, Lyon, France, septembre 2003. URL `http://www.komite.net/laurent/pro/rnc5.pdf`. [Page 107]

[Gar81] John R. Garman : The "bug" heard 'round the world. *ACM SIGSOFT Software Engineering Notes*, 6(5):3–10, 1981. [Pages 1 et 75]

[Gol67] I. Bennett Goldberg : 27 bits are not enough for 8 digit accuracy. *Communications of the ACM*, 10(2):105–106, 1967. URL `http://delivery.acm.org/10.1145/370000/363112/p105-goldberg.pdf`. [Page 17]

[Gol91] David Goldberg : What every computer scientist should know about floating point arithmetic. *ACM Computing Surveys*, 23(1):5–47, 1991. URL `http://www.acm.org/pubs/articles/journals/surveys/1991-23-1/p5-goldberg/p5-goldberg.pdf`. [Pages 3, 17 et 67]

[GPWZ02] Herman Geuvers, Randy Pollack, Freek Wiedijk et Jan Zwanenburg : A constructive algebraic hierarchy in coq. *Journal of Symbolic Computation*, 34(4):271–286, 2002. [Pages 19 et 55]

[Har96] John Harrison : HOL light : a tutorial introduction. Dans Mandayam Srivas et Albert Camilleri, éditeurs : *Proceedings of the First International Conference on Formal Methods in Computer-Aided Design (FMCAD'96)*, volume 1166 de *LNCS*, pages 265–269, 1996. URL `http://www.cl.cam.ac.uk/users/jrh/papers/demo.ps.gz`. [Page 4]

[Har97a] John Harrison : Floating point verification in HOL light : the exponential function. Technical Report 428, University of Cambridge Computer Laboratory, 1997. URL `http://www.cl.cam.ac.uk/users/jrh/papers/tang.ps.gz`. [Page 6]

[Har97b] John Harrison : Verifying the accuracy of polynomial approximations in HOL. Dans *Proceedings of the 10th International Conference on Theorem Proving in*

Higher Order Logics, pages 137–152, Murray Hill, New Jersey, 1997. URL `http://www.cl.cam.ac.uk/users/jrh/papers/poly.ps.gz`. [Page 6]

[Har99] John Harrison : A machine-checked theory of floating point arithmetic. Dans Yves Bertot, Gilles Dowek, André Hirschowitz, Christine Paulin et Laurent Théry, éditeurs : *12th International Conference on Theorem Proving in Higher Order Logics*, pages 113–130, Nice, France, 1999. URL `http://www.cl.cam.ac.uk/users/jrh/papers/fparith.ps.gz`. [Page 6]

[Har00] John Harrison : Formal verification of floating point trigonometric functions. Dans Warren A. Hunt et Steven D. Johnson, éditeurs : *Proceedings of the Third International Conference on Formal Methods in Computer-Aided Design*, pages 217–233, Austin, Texas, 2000. URL `http://www.link.springer.de/link/service/series/0558/papers/1954/19540217.pdf`. [Page 6]

[Hay98] Alden M. Hayashi : Rough sailing for smart ships. *Scientific American*, 279(5): 26, novembre 1998. [Page 1]

[Hig02] Nicholas J. Higham : *Accuracy and stability of numerical algorithms*. SIAM, 2002. URL `http://www.ma.man.ac.uk/~higham/asna/`. Seconde édition. [Pages 3, 35 et 81]

[HKST99] John Harrison, Ted Kubaska, Shane Story et Ping Tak Peter Tang : The computation of transcendental functions on the IA-64 architecture. *Intel Technology Journal*, 1999 Q4, 1999. URL `http://developer.intel.com/technology/itj/q41999/articles/art_5.htm`. [Page 94]

[HLB01] Yozo Hida, Xiaoye S. Li et David H. Bailey : Algorithms for quad-double precision floating point arithmetic. Dans Neil Burgess et Luigi Ciminiera, éditeurs : *Proceedings of the 15th Symposium on Computer Arithmetic*, pages 155–162, Vail, Colorado, 2001. URL `http://csdl.computer.org/comp/proceedings/arith/2001/1150/00/1150toc.htm`. [Page 75]

[How80] William A. Howard : The formulae-as-types notion of construction. Dans Jonathan Paul Seldin et James Roger Hindley, éditeurs : *To H. B. Curry : Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, 1980. [Page 4]

[IEC89] Binary floating-point arithmetic for microprocessor systems. ISO / IEC Standard 60559, International Electrotechnical Commission, 1989. URL `http://www.iec.ch/webstore/`. [Page 9]

[IEC01] Language independent arithmetic, part 2 : Elementary numerical functions. ISO / IEC Standard 10967-2, International Standard, 2001. URL `http://www.open-std.org/JTC1/SC22/WG11/docs/n462.pdf`. [Page 17]

[IEE04] IEEE standard for floating-point arithmetic. Floating-Point Working Group of the Microprocessor Standards Subcommittee of the Standards Committee of the IEEE Computer Society, 2004. URL `http://www.validlab.com/754R/`. Travail en cours. [Pages 9–11, 16, 18, 54, 87, 105 et 107]

[Int96] Intel. *Pentium Pro Family : Developer's Manual*, 1996. Programmer's Reference Manual. [Page 96]

[Jac01] Christian Jacobi : Formal verification of a theory of IEEE rounding. Dans *14th International Conference on Theorem Proving in Higher Order Logics*,

pages 239–254, Edinburgh, Scotland, 2001. URL `http://www.inf.ed.ac.uk/publications/online/0046/b239.pdf`. supplemental proceedings. [Pages 6, 45 et 55]

[Jac02] Christian JACOBI : *Formal Verification of a Fully IEEE Compliant Floating Point Unit*. Thèse de doctorat, Computer Science Department, Saarland University, Saarbrucken, Germany, 2002. URL `http://engr.smu.edu/~seidel/research/diss-jacobi.ps.gz`. [Pages 6 et 55]

[KD98] William KAHAN et Joseph D. DARCY : How java's floating-point hurts everyone everywhere. Dans *ACM 1998 Workshop on Java for High-Performance Network Computing*, page 81, Palo Alto, California, 1998. URL `http://www.cs.berkeley.edu/~wkahan/JAVAhurt.pdf`. [Pages 17 et 88]

[KG99] Christoph KERN et Mark R. GREENSTREET : Formal verification in hardware design : a survey. *ACM Transactions on Design Automation of Electronic Systems*, 4(2):123–193, 1999. URL `http://doi.acm.org/10.1145/307988.307989`. [Page 5]

[KM97] Alan H. KARP et Peter MARKSTEIN : High precision division and square root. *ACM Transactions on Mathematical Software*, 23(4):561–589, 1997. URL `http://delivery.acm.org/10.1145/280000/279237/p561-karp.pdf`. [Page 25]

[KMM00a] Matt KAUFMANN, Panagiotis MANOLIOS et J Strother MOORE : *Computer-Aided Reasoning : ACL2 Case Studies*. Kluwer Academic Publishers, juin 2000. [Page 4]

[KMM00b] Matt KAUFMANN, Panagiotis MANOLIOS et J Strother MOORE : *Computer-Aided Reasoning : An Approach*. Kluwer Academic Publishers, juin 2000. [Page 4]

[Knu73] Donald E. KNUTH : *The Art of Computer Programming : Seminumerical Algorithms*. Addison Wesley, 1973. Seconde édition. [Page 75]

[Knu97] Donald E. KNUTH : *The Art of Computer Programming : Seminumerical Algorithms*. Addison-Wesley, 1997. Troisième édition. [Pages 19, 24 et 76]

[Kul00] Ulrich KULISCH : Rounding near zero. Dans *4th Real Numbers and Computers Conference*, pages 23–29, Dagstuhl, Germany, 2000. [Page 61]

[L+96] Jacques-Louis LIONS *et al.* : Ariane 5 flight 501 failure report by the inquiry board. Rapport technique, European Space Agency, Paris, France, 1996. [Page 1]

[Lam95] Leslie LAMPORT : How to write a proof. *American Mathematical Monthly*, 102(7):600–608, 1995. [Pages 2 et 23]

[Lan01] Philippe LANGLOIS : Automatic linear correction of rounding errors. *BIT - Numerical Mathematics*, 41(3):515–539, 2001. URL `http://ipsapp008.kluweronline.com/IPS/content/search/ext/x/J/6161/I/3/A/7/type/PDF/article.htm`. [Page 23]

[LDB+02] Xiaoye S. LI, James W. DEMMEL, David H. BAILEY, Greg HENRY, Yozo HIDA, Jimmy ISKANDAR, William KAHAN, Anil KAPUR, Michael C. MARTIN, Teresa TUNG et Daniel J. YOO : Design, implementation and testing of extended and mixed precision BLAS. *ACM Transactions on Mathematical Software*, 28(2):152–205, juin 2002. URL `http://www.nersc.gov/~dhb/dhbpapers/xblas-report.pdf`. [Page 75]

[Lef00] Vincent LEFÈVRE : *Moyens arithmétiques pour un calcul fiable*. Thèse de doctorat, École Normale Supérieure de Lyon, Lyon, France, 2000. URL `ftp://ftp.ens-lyon.fr/pub/LIP/Rapports/PhD/PhD2000/PhD2000-02.ps.Z`. [Pages 17 et 67]

[Lin76] Seppo LINNAINMAA : Taylor expansion of the accumulated rounding error. *BIT*, 16:146–160, 1976. [Page 91]

[Lin81] Seppo LINNAINMAA : Software for doubled precision floating point computations. *ACM Transactions on Mathematical Software*, 7(3):272–283, 1981. URL `http://delivery.acm.org/10.1145/360000/355960/p272-linnainmaa.pdf`. [Page 25]

[LM01] Vincent LEFÈVRE et Jean-Michel MULLER : Worst cases for correct rounding of the elementary functions in double precision. Dans Neil BURGESS et Luigi CIMINIERA, éditeurs : *Proceedings of the 15th Symposium on Computer Arithmetic*, pages 111–118, Vail, Colorado, 2001. URL `http://csdl.computer.org/comp/proceedings/arith/2001/1150/00/1150toc.htm`. [Pages 17 et 67]

[LMS85] Leslie LAMPORT et P. Michael MELLIAR-SMITH : Synchronizing clocks in the presence of faults. *Journal of the ACM*, 32(1):52–78, 1985. URL `http://www.acm.org/pubs/articles/journals/jacm/1985-32-1/p52-lamport/p52-lamport.pdf`. [Page 2]

[LMW01] Odile LAURENT, Pierre MICHEL et Virginie WIELS : Using formal verification techniques to reduce simulation and test effort. Dans *International Symposium of Formal Methods Europe*, pages 465–477, Berlin, Germany, 2001. URL `http://www.cert.fr/francais/deri/wiels/Publi/fme01.ps`. [Page 57]

[Loi97] Patrick LOISELEUR : Formalisation en coq de la norme IEEE 754 sur l'arithmétique à virgule flottante. Mémoire de DEA, École Normale Supérieure, Paris, France, 1997. URL `http://www.lri.fr/~loisel/rapport-stage-dea.ps.gz`. [Page 45]

[LP92] Zhaohui LUO et Randy POLLACK : Lego proof development system : User's manual. Rapport technique ECS-LFCS-92-211, LFCS, 1992. URL `http://www.lfcs.inf.ed.ac.uk/reports/92/ECS-LFCS-92-211/ECS-LFCS-92-211.ps`. [Page 4]

[LV74] Michel LA PORTE et Jean VIGNES : Étude statistique des erreurs dans l'arithmétique des ordinateurs ; application au contrôle des résultats d'algorithmes numériques. *Numerische Mathematik*, 23(1):63–72, 1974. [Page 67]

[Mar90] Peter W. MARKSTEIN : Computation of elementary functions on the IBM RISC System/6000 processor. *IBM Journal of Research and Development*, 34(1):111–119, 1990. URL `http://www.research.ibm.com/journal/rd/341/ibmrd3401N.pdf`. [Page 91]

[Mar00] Peter MARKSTEIN : *IA-64 and elementary functions : speed and precision*. Prentice Hall, 2000. URL `http://www.markstein.org/`. [Pages 42, 81, 87, 91 et 95]

[May01] Micaela MAYERO : *Formalisation et automatisation de preuves en analyses rélle et numérique*. Thèse de doctorat, Université Paris VI, décembre 2001. URL `ftp://ftp.inria.fr/INRIA/LogiCal/Micaela.Mayero/papers/these-mayero.ps.gz`. [Pages 19 et 55]

[MF01] Claire Moreau-Finot : *Preuves et algorithmes utilisant l'arithmétique normalisée IEEE*. Thèse de doctorat, École Normale Supérieure de Lyon, Lyon, France, 2001. URL `ftp://ftp.ens-lyon.fr/pub/LIP/Rapports/PhD/PhD2001/PhD2001-03.ps.Z`. [Pages 3, 10, 23 et 75]

[Min95] Paul S. Miner : Defining the IEEE-854 floating-point standard in PVS. Rapport technique 110167, NASA Langley Research Center, 1995. URL `http://shemesh.larc.nasa.gov/fm/ftp/larc/Floating_Point/TM-110167.ps`. [Pages 5, 32 et 45]

[ML96] Paul S. Miner et James F. Leathrum : Verification of IEEE compliant subtractive division algorithms. Dans *Proceedings of the First International Conference on Formal Methods in Computer-Aided Design*, pages 64–78, 1996. URL `http://www.ece.odu.edu/~leathrum/Formal_Methods/computer_arithmetic/fmcad.ps`. [Page 6]

[MLK98] J Strother Moore, Thomas Lynch et Matt Kaufmann : A mechanically checked proof of the AMD5K86 floating point division program. *IEEE Transactions on Computers*, 47(9):913–926, 1998. URL `http://csdl.computer.org/dl/trans/tc/1998/09/t0913.pdf`. [Page 6]

[Møl65a] Ole Møller : Note on quasi double-precision. *BIT*, 5(4):251–255, 1965. [Pages 23 et 24]

[Møl65b] Ole Møller : Quasi double-precision in floating point addition. *BIT*, 5(1):37–50, 1965. [Pages 23 et 24]

[Mul89] Jean-Michel Muller : *Arithmétique des Ordinateurs*. Masson, 1989. [Page 76]

[Mul95] Jean-Michel Muller : Algorithmes de division pour microprocesseurs : illustration à l'aide du "bug" du Pentium. *Technique et Science Informatiques*, 14(8), 1995. URL `ftp://ftp.ens-lyon.fr/pub/LIP/Rapports/RR/RR1995/RR1995-06.ps.Z`. [Page 1]

[Mul97] Jean-Michel Muller : *Elementary functions, algorithms and implementation*. Birkhauser, 1997. URL `http://www.birkhauser.com/detail.tpl?ISBN=081763990X`. [Pages 81 et 91]

[Ng92] Kwok C. Ng : Argument reduction for huge arguments : good to the last bit. URL `http://www.cs.nyu.edu/exact/pap/transcendental/NgTrigArgReduction.pdf`. Work in progress, 1992. [Page 91]

[NPW02] Tobias Nipkow, Lawrence C. Paulson et Markus Wenzel : *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 de *LNCS*. Springer, 2002. [Page 4]

[ORS92] Sam Owre, John. M. Rushby et Natarajan Shankar : PVS : A prototype verification system. Dans Deepak Kapur, éditeur : *11th International Conference on Automated Deduction (CADE)*, volume 607 de *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, juin 1992. Springer-Verlag. [Page 4]

[Ove95] Michael Overton : Notes on numerical computing. manuscript, University of New York, 1995. [Page 3]

[OZGS99] John O'Leary, Xudong Zhao, Rob Gerth et Carl-Johan H. Seger : Formally verifying IEEE compliance of floating point hardware. *Intel Technology Journal*,

3(1), 1999. URL `http://developer.intel.com/technology/itj/q11999/pdf/floating_point.pdf`. [Page 1]

[Per91] Georges Perec : *Cantatrix sopranica L. et autres écrits scientifiques*. La Librairie du 20e siècle. Éditions du Seuil, 1991. URL `http://pauillac.inria.fr/~xleroy/stuff/tomato/tomato.html`. [Page 133]

[PFTV89] William H. Press, Brian P. Flannery, Saul A. Teukolsky et William T. Vetterling : *Numerical Recipes*. Cambridge University Press, 1989. [Page 81]

[Pri91] Douglas M. Priest : Algorithms for arbitrary precision floating point arithmetic. Dans Peter Kornerup et David Matula, éditeurs : *Proceedings of the 10th Symposium on Computer Arithmetic*, pages 132–144, Grenoble, France, 1991. URL `http://www.cs.cmu.edu/afs/cs/project/quake/public/papers/related/Priest.ps`. [Page 75]

[Pri92] Douglas M. Priest : *On properties of floating point arithmetics : numerical stability and the cost of accurate computations*. Thèse de doctorat, University of California at Berkeley, Berkeley, California, 1992. URL `ftp://ftp.icsi.berkeley.edu/pub/theory/priest-thesis.ps.Z`. [Page 67]

[PRN80] John F. Palmer, Bruce W. Ravenel et Rafi Nave : Numeric data processor. US Patent 4 338 675, US Patent Office, 1980. URL `http://www.patents.ibm.com/details?pn=US04338675__`. [Page 91]

[PV93] Michèle Pichat et Jean Vignes : *Ingénierie du contrôle de la précision des calculs sur ordinateur*. Editions Technip, 1993. [Pages 24 et 67]

[RK75] John F. Reiser et Donald E. Knuth : Evading the drift in floating point addition. *Information Processing Letters*, 3(3):84–87, 1975. [Page 16]

[Rus98] David M. Russinoff : A mechanically checked proof of IEEE compliance of the floating point multiplication, division and square root algorithms of the AMD-K7 processor. *LMS Journal of Computation and Mathematics*, 1:148–200, 1998. URL `http://www.onr.com/user/russ/david/k7-div-sqrt.ps`. [Pages 6 et 45]

[Rus99] David M. Russinoff : A mechanically checked proof of correctness of the amd k5 floating point square root microcode. *Formal Methods in System Design*, 14(1):75, 1999. [Page 6]

[Rus00] David M. Russinoff : A case study in formal verification of register-transfer logic with ACL2 : The floating point adder of the AMD Athlon processor. *Lecture Notes in Computer Science*, 1954:3–36, 2000. URL `http://www.onr.com/user/russ/david/fadd.ps`. [Page 6]

[RvH91] John Rushby et Friedrich von Henke : Formal verification of algorithms for critical systems. Dans *Proceedings of the Conference on Software for Critical Systems*, pages 1–15, New Orleans, Louisiana, 1991. URL `http://www.acm.org/pubs/articles/proceedings/soft/125083/p1-rushby/p1-rushby.pdf`. [Page 2]

[S+81] David Stevenson *et al.* : A proposed standard for binary floating point arithmetic. *IEEE Computer*, 14(3):51–62, 1981. [Pages 2 et 9]

[S+87] David Stevenson *et al.* : An American national standard : IEEE standard for binary floating point arithmetic. *ACM SIGPLAN Notices*, 22(2):9–25, 1987. [Pages 2 et 9]

[She96] Jonathan R. Shewchuk : Robust adaptative floating point geometric predicates. Dans *Proceedings of the 12th Annual ACM Symposium on Computational Geometry*, pages 141–150, Philadelphia, Pensylvania, 1996. URL `http://www.acm.org/pubs/articles/proceedings/compgeom/237218/p141-shewchuk/p141-shewchuk.pdf`. [Page 75]

[She97] Jonathan R. Shewchuk : Adaptive precision floating-point arithmetic and fast robust geometric predicates. Dans *Discrete and Computational Geometry*, volume 18, pages 305–363, 1997. URL `http://link.springer.de/link/service/journals/00454/papers97/18n3p305.pdf`. [Pages 23, 75 et 76]

[SSK99] Eric M. Schwarz, Ronald M. Smith et Christopher A. Krygowski : The S/390 G5 floating point unit supporting hex and binary architectures. Dans Israel Koren et Peter Kornerup, éditeurs : *Proceedings of the 14th Symposium on Computer Arithmetic*, pages 258–265, Adelaide, Australia, 1999. URL `http://csdl.computer.org/comp/proceedings/arith/1999/0116/00/0116toc.htm`. [Page 11]

[Ste74] Pat H. Sterbenz : *Floating point computation*. Prentice Hall, 1974. [Pages 33, 35 et 63]

[Stu80] Friedrich Stummel : Rounding error analysis of elementary numerical algorithms. *Computing*, 24(Supplementum 2):169–195, 1980. [Page 81]

[Sun96] Sun Microsystems. *Numerical Computation Guide*, 1996. URL `ftp://192.18.99.138/802-5692/802-5692.pdf`. [Page 3]

[Tan91] Ping Tak Peter Tang : Table-lookup algorithms for elementary functions and their error analysis. Dans Peter Kornerup et David Matula, éditeurs : *Proceedings of the 10th Symposium on Computer Arithmetic*, pages 232–236, Grenoble, France, 1991. [Page 91]

[TE77] Kishor S. Trivedi et Miloš D. Ercegovac : On-line algorithms for division and multiplication. *IEEE Transactions on Computers*, 26(7):681–687, 1977. [Page 76]

[Tex97] Texas Instruments. *TMS320C3x — User's guide*, 1997. URL `http://www-s.ti.com/sc/psheets/spru031e/spru031e.pdf`. [Pages 57, 66 et 71]

[Thé03] Laurent Théry : Formal proof authoring : an experiment. Dans David Aspinall et Christoph Lüth, éditeurs : *Proc. International Workshop on User Interfaces for Theorem Provers UITP'03*, pages 143–159. Tech. Report 189, Institut für Informatik, Albert-Ludwigs-Universität Freiburg, septembre 2003. URL `http://www.informatik.uni-bremen.de/uitp03/proceedings.pdf`. [Page 107]

[Uni92] Patriot missile defense : Software problem led to system failure at Dhahran, Saudi Arabia. Report GAO/IMTEC-92-26, Information Management and Technology Division, United States General Accounting Office, Washington, D.C., février 1992. URL `http://www.fas.org/spp/starwars/gao/im92026.htm`. [Page 1]

[Val88] Jim Valerio : Computer system support for scientific and engineering computation. Lecture 13, 1988. [Page 91]

[Vol59] Jack Volder : The CORDIC trigonometric computing technique. *IRE Transactions on Electronic Computers*, 8:330–334, 1959. [Page 91]

[Wad60] W. G. Wadey : Floating point arithmetics. *Journal of the ACM*, 7(2):129–139, 1960. URL `http://delivery.acm.org/10.1145/330000/321024/p129-wadey.pdf`. [Page 18]

[WG95] Weng Fai WONG et Eiichi GOTO : Fast evaluation of the elementary functions in single precision. *IEEE Transactions on Computers*, 44(3):453–457, 1995. URL `http://csdl.computer.org/dl/trans/tc/1995/03/t0453.pdf`. [Page 91]

[Wie03a] Freek WIEDIJK : Comparing mathematical provers. Dans Andrea ASPERTI, Bruno BUCHBERGER et James DAVENPORT, éditeurs : *Proceedings of Mathematical Knowledge Management (MKM)*, volume 2594 de *LNCS*, pages 188–202, 2003. URL `http://www.cs.kun.nl/~freek/comparison/diffs.ps.gz`. [Page 4]

[Wie03b] Freek WIEDIJK : Formal proof sketches. Dans Wan FOKKINK et Jaco van de POL, éditeurs : *7th Dutch Proof Tools Day*, Amsterdam, 2003. URL `http://www.cs.kun.nl/~freek/notes/sketches.pdf`. [Page 2]

Liste des théorèmes

Table des figures

Table des symboles

On note $\mathbb{F}$ l'ensemble des nombres flottants.

Symbole	Type	Exemple	Explication
$e_?$	$\mathbb{F} \to \mathbb{Z}$	e_f	e_f est l'exposant du flottant f.
$n_?$	$\mathbb{F} \to \mathbb{Z}$	n_f	n_f est la mantisse du flottant f.
ulp	$\mathbb{F} \to \mathbb{R}$	$\text{ulp}(f)$	« Unit in the Last Place ».
$\tilde{?}$	$\mathbb{F} \to \mathbb{F}$	$\tilde{f}$	$\tilde{f}$ est le représentant canonique de f.
$?^+$	$\mathbb{F} \to \mathbb{F}$	f^+	f^+ est le successeur de f.
$?^-$	$\mathbb{F} \to \mathbb{F}$	f^-	f^- est le prédécesseur de f.
$=_{\mathbb{R}}$	$\mathbb{F} \to \mathbb{F} \to \text{Prop}$	$f =_{\mathbb{R}} g$	Les valeurs réelles de f et g sont égales.
$\mathcal{B}$	$(\mathbb{N}, \mathbb{N})$	Voir 2.3.	Format flottant usuel.
$\mathbb{B}$	$(\mathbb{N}, \mathbb{N}, \mathbb{N})$	Voir 6.2.1.	Format flottant générique.
$\circ$	$\mathbb{R} \to \mathbb{F}$	$f = \circ(z)$	f est l'un des arrondis au plus proche de z.
$\triangledown$	$\mathbb{R} \to \mathbb{F}$	$f = \triangledown(z)$	f est l'arrondi vers $-\infty$ de z.
$\triangle$	$\mathbb{R} \to \mathbb{F}$	$f = \triangle(z)$	f est l'arrondi vers $+\infty$ de z.
$\square$	$\mathbb{R} \to \mathbb{F}$	$f = \square(z)$	f est l'un des arrondis fidèles de z.
$\boxplus$	$\mathbb{R} \to \mathbb{R} \to \mathbb{F}$	$f = x \boxplus y$	f est un arrondi de $x + y$.
$\boxminus$	$\mathbb{R} \to \mathbb{R} \to \mathbb{F}$	$f = x \boxminus y$	f est un arrondi de $x - y$.
$\boxtimes$	$\mathbb{R} \to \mathbb{R} \to \mathbb{F}$	$f = x \boxtimes y$	f est un arrondi de $x \times y$.
$\boxslash$	$\mathbb{R} \to \mathbb{R} \to \mathbb{F}$	$f = x \boxslash y$	f est un arrondi de x/y.
$\boxdot$	$\mathbb{R} \to \mathbb{F}$	$f = \boxdot x$	f est un arrondi de $\sqrt{x}$.
$\oplus$	$\mathbb{R} \to \mathbb{R} \to \mathbb{F}$	$f = x \oplus y$	f est un arrondi au plus proche de $x + y$.
$\ominus$	$\mathbb{R} \to \mathbb{R} \to \mathbb{F}$	$f = x \ominus y$	f est un arrondi au plus proche de $x - y$.
$\otimes$	$\mathbb{R} \to \mathbb{R} \to \mathbb{F}$	$f = x \otimes y$	f est un arrondi au plus proche de $x \times y$.
$\oslash$	$\mathbb{R} \to \mathbb{R} \to \mathbb{F}$	$f = x \oslash y$	f est un arrondi au plus proche de x/y.
$\lceil ? \rfloor$	$\mathbb{R} \to \mathbb{Z}$	$m = \lceil z \rfloor$	m est l'un des entiers les plus proches de z.

Annexes

Annexe A

Dépendances des fichiers Coq

Computer... if you don't open that exit hatch this moment I shall zap straight off to your major data banks and reprogram you with a very large axe, got that?
Douglas Adams, The Hitch Hiker's Guide To The Galaxy (1979)

Cette annexe détaille les dépendances entre les différents fichiers Coq.

La sémantique des graphes est la suivante : dans l'exemple de la figure A.1, le fichier A.v dépend du fichier B.v, c'est-à-dire que pour prouver les résultats du fichier A.v, on a eu besoin des résultats déjà prouvés dans le fichier B.v.

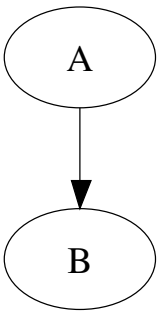


Fig. A.1 – Exemple de dépendance.

La figure A.2 de la page 130 représente le graphe complet de dépendances des fichiers Coq de la bibliothèque dans son ensemble.

La figure A.3 de la page 131 représente le graphe de dépendances des fichiers dont les résultats sont détaillés dans ce manuscrit.

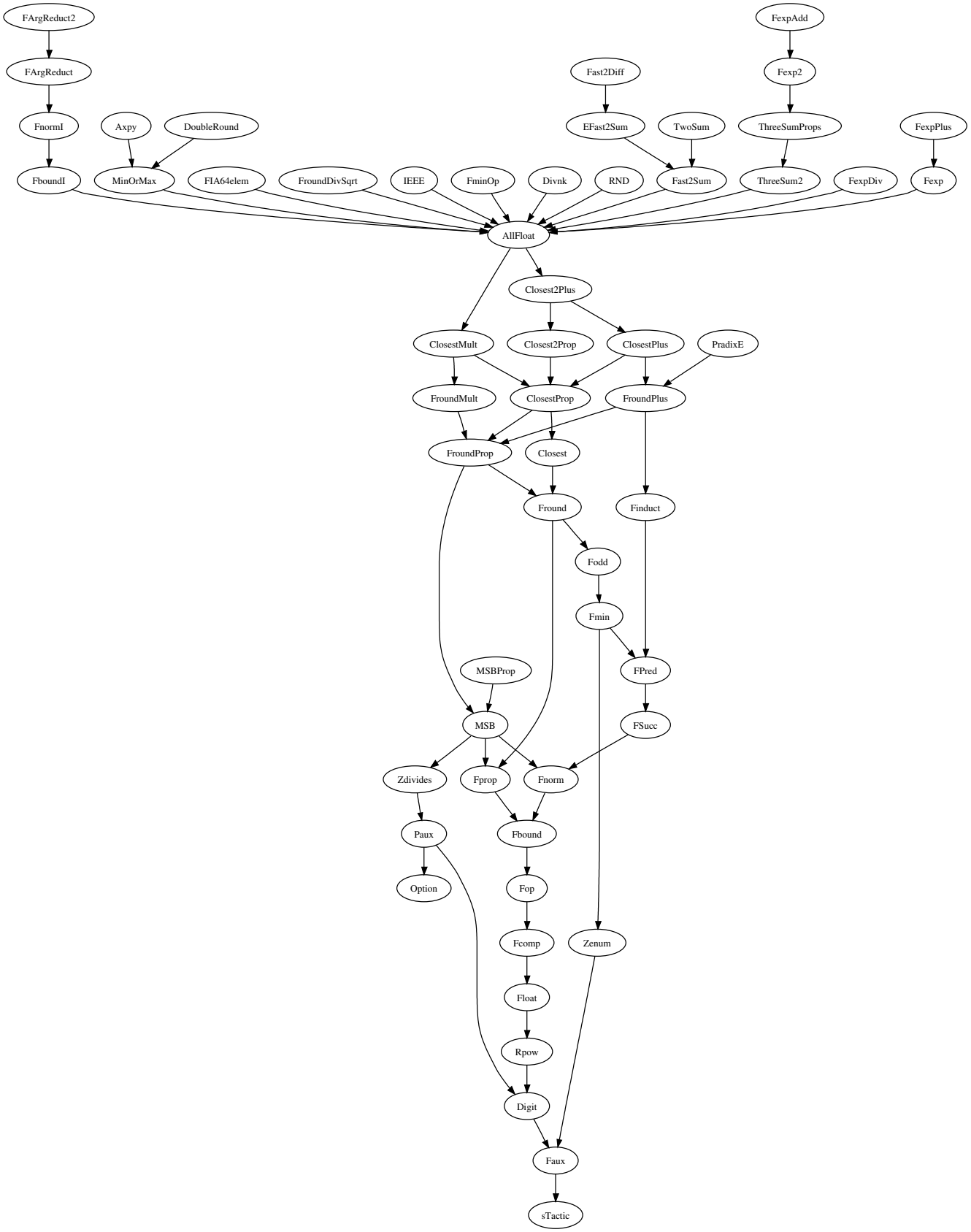


FIG. A.2 – Graphe de dépendances de la bibliothèque.

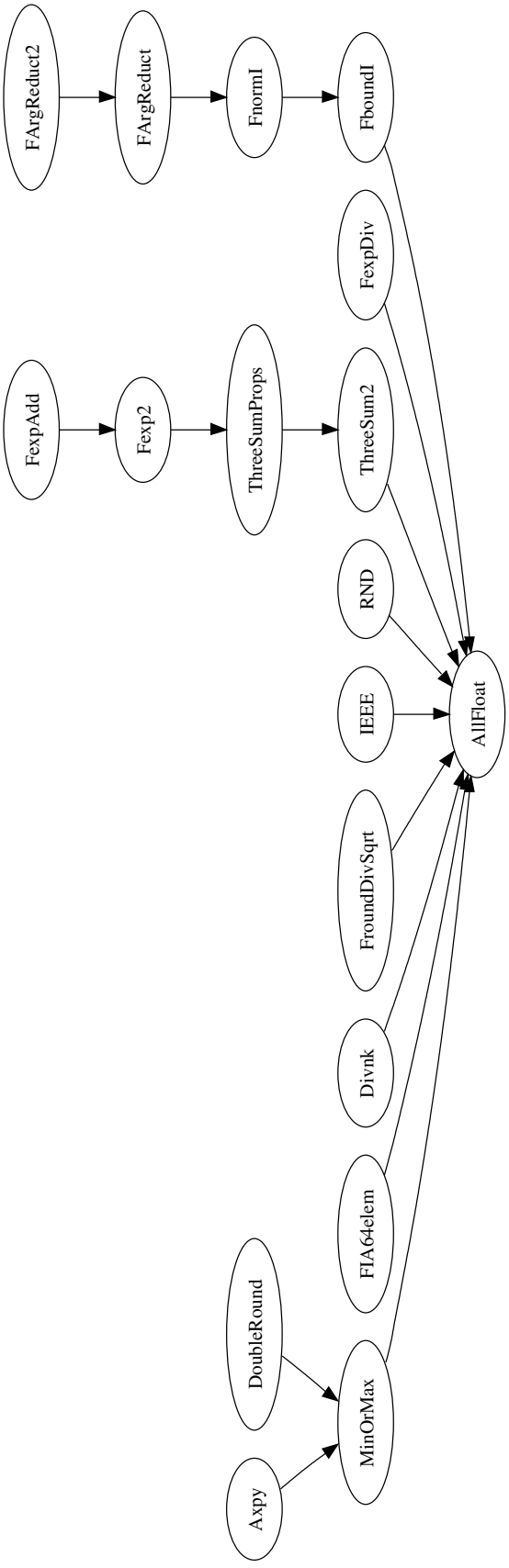


FIG. A.3 – Graphe de dépendances réduit.

Annexe B

Division d'expansions

At the end of the experiments, Sopranoes were perfused with olive oil, and 10% GlennFiddish, and incubated at 421°C in 15% orange juice during 47 hours.
Georges Perec [Per91]

Ce chapitre est une démonstration de la validité de l'opérateur de division d'expansions décrit en figure 8.6. On ne laisse passer c dans l'opérateur Σ'_3 que si $c \geq 2^{2-p}|a \oplus b| \ \wedge \ c \geq \frac{|a \oplus b| \, \varepsilon_D}{(1-2^{-p})^2}$ où $\varepsilon_D = \max_j(|d_{j+1}|/|d_j|)$.

Si c est trop petit, on fournit alors une nouvelle composante de Q qui est $q_i = (a' \oplus b') \oslash d_0$. Les nouvelles valeurs de a et b sont alors l'arrondi de $a' + b' - d_0 q_i$ et l'erreur commise.

Cet algorithme et cette condition garantissent que $a + b + c$ est toujours représentable comme la somme de deux nombres flottants (section B.1), que les futurs produits partiels $q_i \otimes d_j$ sont plus petits que les composantes déjà entrées dans l'opérateur Σ'_3 (section B.2) et que Q est une pseudo-expansion. Je peux borner le nombre de bits de chevauchement par une formule complexe (section B.3).

On montre facilement que $c \geq 2^{2-p}|a \oplus b|$ implique $|c| \geq 2\text{ulp}(a \oplus b)$ et c'est cette seconde condition que j'utilise. On note $\mathcal{N}$ la fonction renvoyant le canonique associé à un nombre flottant. On note $\varepsilon_R = \max_j(|r_{j+1}|/|r_j|)$. On suppose un format à p bits et on suppose qu'aucun nombre flottant n'est dénormalisé, donc si $f = \circ(z)$, alors $|f| \leq \frac{|z|}{1-2^{-p}}$ (adaptation du lemme 56).

B.1 L'opérateur Σ'_3 fonctionne : $c' = 0$

Je veux montrer que $|c| \geq 2\text{ulp}(a \oplus b) \Rightarrow c' = 0$.

Si $a = 0$ ou $w = 0$, j'ai $c' = 0$. Je suppose donc que a et w sont non nuls.

Je commence par montrer que $|b| \leq \frac{|a|}{2}$. Si le nouveau a est l'arrondi de $a' + b' - d_0 q_i$ et b est l'erreur, c'est vrai. Sinon, ce sont des a' et b' de l'itération précédente.

Si $e_{\mathcal{N}(a)} \geq e_{\mathcal{N}(u)}$, alors on peut représenter w avec l'exposant $\min(e_{\mathcal{N}(a)}, e_{\mathcal{N}(u)}) = e_{\mathcal{N}(u)}$ et $\text{ulp}(u) = 2^{e_{\mathcal{N}(u)}} = 2^{e_w} \leq |w|$ donc

$$|b'| \leq \frac{1}{1-2^{-p}}|w+v| \leq \frac{1}{1-2^{-p}}\left(|w| + \frac{1}{2}\text{ulp}(u)\right) \leq \frac{1}{1-2^{-p}}\,\frac{3}{2}|w| \leq \frac{1}{1-2^{-p}}\,\frac{3}{4}\text{ulp}(a') \leq \frac{|a'|}{2}.$$

Sinon, on a $e_{\mathcal{N}(a)} > e_{\mathcal{N}(u)}$ et on sait alors que $|a| > |u|$. . On a ici encore deux cas : si $w = 0$, alors $e_{a'} \geq e_{\mathcal{N}(u)}$ et donc

$$|b'| = |v| \leq \frac{1}{2}\text{ulp}(u) \leq \frac{|a'|}{2}.$$

Sinon, je vais montrer que $|a'| \geq |u|$. En effet, si a et u sont de même signe, c'est évident. Sinon, on a deux cas : si $2|u| \geq |a|$, alors le théorème de Sterbenz s'applique et $w = 0$, ce qui est impossible. On a donc $2|u| < |a|$ donc $|a| - |u| \geq |u|$ et par monotonie de l'arrondi $|a'| \geq |u|$. On a alors :

$$|b'| \leq \frac{1}{1-2^{-p}}|w+v| \leq \frac{1}{1-2^{-p}}\left(\frac{1}{2}\text{ulp}(a') + \frac{1}{2}\text{ulp}(u)\right) \leq \frac{1}{1-2^{-p}}\text{ulp}(a') \leq \frac{|a'|}{2}.$$

Donc dans tous les cas, $\text{ulp}(a \oplus b) = 2^{e_{\mathcal{N}(a\oplus b)}} \geq 2^{e_{\mathcal{N}(a)}-1} = \frac{1}{2}\text{ulp}(a)$.
Et $2^{e_c+p} > (2^p - 1)2^{e_c} \geq |c| \geq \text{ulp}(a) = 2^{e_{\mathcal{N}(a)}}$ donc

$$e_{\mathcal{N}(a)} - e_c \leq p - 1.$$

Pour montrer que $c' = 0$, il suffit alors de montrer que $v + w$ est borné. Or $v + w = n_w 2^{e_w} + n_v 2^{e_v} = (n_w 2^{e_u - e_c} + n_v)2^{e_c}$ Si je prends e_c comme exposant, il ne reste qu'à borner la mantisse.

Or $e_{a'} \leq e_{\mathcal{N}(a)} + 1$ et $e_u = e_w < e_{a'}$ donc

$$\begin{aligned} |(n_w 2^{e_u-e_c} + n_v)| &= |v+w|2^{-e_c} \leq (2^{e_{a'}-1} + 2^{e_u-1})2^{-e_c} \\ &\leq 2^{e_{a'}-e_c-1} + 2^{e_u-e_c-1} < 2 \times 2^{e_{\mathcal{N}(a)}-e_c} \leq 2^p \end{aligned}$$

B.2 Les apports à la MQ décroissent

Si $c \geq \left(\frac{1}{1-2^{-p}}\right)^2 \varepsilon_D |a \oplus b|$, les futurs termes sont bien plus petits que c car :

$$\begin{aligned} |q_i \otimes d_j| &\leq \frac{1}{1-2^{-p}}|q_i||d_j| = \frac{1}{1-2^{-p}}|(a \oplus b) \oslash d_0||d_j| \\ &\leq \left(\frac{1}{1-2^{-p}}\right)^2 |(a \oplus b)||d_j|/|d_0| \\ &\leq \left(\frac{1}{1-2^{-p}}\right)^2 \varepsilon_D |a \oplus b|. \end{aligned}$$

B.3 Chevauchement de Q

Soit $M = \max\left(2\text{ulp}(a \oplus b), \frac{|a\oplus b|\,\varepsilon_D}{(1-2^{-p})^2}\right)$.

On note sous l'indice i, les variables (a, b et M) correspondant au moment où on sort la composante q_i de Q et de même pour $i + 1$ et q_{i+1}.

Le terme suivant q_{i+1} de Q est obtenu après avoir accumulé des termes dans l'opérateur Σ'_3. Ces termes sont :

1. les nouveaux a' et b' après le calcul de q_i ;
2. les termes de R ayant été ajoutés ;
3. les termes du type $q_j \otimes d_k$ ayant été ajoutés ;
4. les termes du type $q_j d_k - (q_j \otimes d_k)$ ayant été ajoutés.

Étudions maintenant chacun de ces termes, sachant que tous les termes ajoutés sont inférieurs à M_i.

Tout d'abord, les nouveaux a' et b' après le calcul de q_i.

$$\begin{aligned} |a'+b'| &= |a_i+b_i-d_0q_i| \\ &= |a_i+b_i-(a_i\oplus b_i)+d_0((a_i\oplus b_i)/d_0-(a_i\oplus b_i)\oslash d_0) \\ &\leq \frac{1}{2}\text{ulp}(a_i\oplus b_i)+|d_0|\frac{1}{2}\text{ulp}(q_i). \end{aligned}$$

Ensuite les termes de R ayant été ajoutés.

$$|\Sigma r_j| = \Sigma_{j=K}^{K'}|r_j| \leq |r_K|(1+\varepsilon_R+\cdots+\varepsilon_R^{K'-K}) \leq \frac{|r_K|}{1-\varepsilon_R} \leq \frac{M_i}{1-\varepsilon_R}.$$

Ensuite les termes du type $q_j \otimes d_k$ ayant été ajoutés.

$$\begin{aligned} |\Sigma q_j\otimes d_k| &\leq \frac{1}{1-2^{-p}}\Sigma_{j\in J}\Sigma_{k\in K(j)}|q_j||d_k| \\ &\leq \frac{1}{1-2^{-p}}\Sigma_{j\in J}|q_j|\frac{1}{1-\varepsilon_D}\max_{k\in K(j)}(|d_k|) \\ &\leq \left(\frac{1}{1-2^{-p}}\right)^2\Sigma_{j\in J}\frac{M_i}{1-\varepsilon_D} \\ &\leq \left(\frac{1}{1-2^{-p}}\right)^2 \#Q\ \frac{M_i}{1-\varepsilon_D}. \end{aligned}$$

Enfin les termes du type $q_jd_k-(q_j\otimes d_k)$ ayant été ajoutés.

$$\begin{aligned} |\Sigma q_jd_k-(q_j\otimes d_k)| &\leq \Sigma_{j\in J,k\in K(j)}\text{ulp}(q_i\otimes d_k) \\ &\leq \Sigma_{j\in J,k\in K(j)}2^{1-p}|q_i\otimes d_k| \\ &\leq 2\times 2^{-p}\max_{i,k}|q_i\otimes d_k|\left(\frac{1}{1-2^{-p}}\right)^2 \#Q\ \frac{1}{1-\varepsilon_D} \\ &\leq 2\left(\frac{1}{1-2^{-p}}\right)^2 \#Q\ \frac{M_i}{1-\varepsilon_D}. \end{aligned}$$

Donc pour les a_{i+1} et b_{i+1} servant à calculer q_{i+1}, j'ai

$$\begin{aligned} a_{i+1}+b_{i+1} &= a'+b'+\Sigma r_j+\Sigma q_j\otimes d_k+\Sigma q_jd_k-(q_j\otimes d_k) \\ |a_{i+1}+b_{i+1}| &\leq \frac{1}{2}\text{ulp}(a_i\oplus b_i)+|d_0|\frac{1}{2}\text{ulp}(q_i)+\frac{M_i}{1-\varepsilon_R}+3\left(\frac{1}{1-2^{-p}}\right)^2 \#Q\ \frac{M_i}{1-\varepsilon_D}. \end{aligned}$$

Lemme intermédiaire :

$$\frac{\text{ulp}(x)}{|y|} \leq 2\text{ulp}(x\oslash y)$$

En effet, comme $2^{p-1}\text{ulp}(x) \leq |x|$ d'une part, $|x \oslash y| \leq (2^p-1)\text{ulp}(x \oslash y)$ d'autre part et enfin $\left|\frac{x}{y} - x \oslash y\right| \leq \frac{1}{2}\text{ulp}(x \oslash y)$. À partir de cela, je déduis :

$$\begin{aligned}\frac{\text{ulp}(x)}{|y|} &= 2^{1-p}\frac{2^{p-1}\text{ulp}(x)}{|y|} \leq 2^{1-p}\left|\frac{x}{y}\right| \\ &\leq 2^{1-p}\left(|x \oslash y| + \frac{1}{2}\text{ulp}(x \oslash y)\right) \\ &\leq 2^{1-p}\left(2^p\text{ulp}(x \oslash y)\right) \leq 2\text{ulp}(x \oslash y).\end{aligned}$$

Calcul final : j'ai alors $q_{i+1} = (a_{i+1} \oplus b_{i+1}) \oslash d_0$ avec $a+b$ ayant été majoré plus haut. En utilisant le lemme intermédiaire,

$$\begin{aligned}|q_{i+1}| &= \left|(a_{i+1} \oplus b_{i+1}) \oslash d_0 - \frac{a_{i+1} \oplus b_{i+1}}{d_0} + \frac{a_{i+1} \oplus b_{i+1} - (a_{i+1} + b_{i+1})}{d_0} + \frac{a_{i+1} + b_{i+1}}{d_0}\right| \\ &\leq \frac{\text{ulp}(q_{i+1})}{2} + \frac{1}{2|d_0|}\text{ulp}(a_{i+1} \oplus b_{i+1}) + \frac{|a_{i+1} + b_{i+1}|}{|d_0|} \\ &\leq \frac{\text{ulp}(q_{i+1})}{2} + \frac{3}{2}\text{ulp}(q_i) + \frac{1}{|d_0|}\left(\frac{\text{ulp}(a_i \oplus b_i)}{2} + \frac{M_i}{1-\varepsilon_R} + \frac{3\,\#Q\,M_i}{(1-\varepsilon_D)(1-2^{-p})^2}\right) \\ &\leq \frac{\text{ulp}(q_{i+1})}{2} + \frac{5}{2}\text{ulp}(q_i) + \frac{M_i}{|d_0|}\left(\frac{1}{1-\varepsilon_R} + \left(\frac{1}{1-2^{-p}}\right)^2\frac{3\,\#Q}{1-\varepsilon_D}\right).\end{aligned}$$

Or

$$\begin{aligned}\frac{M_i}{|d_0|} &= \frac{1}{|d_0|}\ \max\left(2\text{ulp}(a_i \oplus b_i), \frac{|a_i \oplus b_i|\,\varepsilon_D}{(1-2^{-p})^2}\right) \\ &\leq \max\left(4\text{ulp}(q_i), \frac{|q_i|(1+2^{-p})\,\varepsilon_D}{(1-2^{-p})^2}\right) \\ &\leq \text{ulp}(q_i)\ \max\left(4, \frac{(2^p-2^{-p})\,\varepsilon_D}{(1-2^{-p})^2}\right).\end{aligned}$$

Je suppose que D est quasi-compressée : $\varepsilon_D \leq 2^{1-p}$. Le terme de gauche est alors prépondérant et

$$\frac{M_i}{|d_0|} \leq 4\text{ulp}(q_i) \leq 8|q_i|2^{-p}.$$

Il vient donc

$$|q_{i+1}| \leq |q_{i+1}|2^{-p} + 5|q_i|2^{-p} + 8|q_i|2^{-p}\left(\frac{1}{1-\varepsilon_R} + \left(\frac{1}{1-2^{-p}}\right)^2\frac{3\,\#Q}{1-\varepsilon_D}\right).$$

Et j'en déduis

$$|q_{i+1}| \leq \frac{2^{-p}|q_i|}{1-2^{-p}}\left(5 + 8\left(\frac{1}{1-\varepsilon_R} + \left(\frac{1}{1-2^{-p}}\right)^2\frac{3\,\#Q}{1-\varepsilon_D}\right)\right).$$

Si je suppose de plus un chevauchement de 10 bits pour R, j'obtiens pour Q au maximum 9 bits de chevauchement si j'utilise la simple précision et 10 pour la double précision.

Résumé :

Cette thèse est représentative de mon expérience de rapprochement de l'arithmétique à virgule flottante, régie par la norme IEEE-754, et de la preuve formelle, ici l'assistant de preuves Coq. La formalisation des nombres flottants utilisée a été premièrement développée par L. Théry.

J'ai tout d'abord testé et enrichi la bibliothèque avec des propriétés simples à exprimer dans le formalisme choisi : le fait qu'une valeur réelle soit exactement représentable par un nombre flottant. J'ai ensuite fait différentes extensions du modèle : rapprochement avec la réalité matérielle des processeurs, généralisation à la représentation en complément à 2 et étude d'un arrondi plus faible. En utilisant les résultats précédents, j'ai étudié deux applications réelles. La première est une bibliothèque de calcul multi-précision basée sur les expansions. La seconde est l'évaluation de fonctions élémentaires (exponentielle, cosinus...) : j'ai résolu la plupart des problèmes de la réduction d'argument en garantissant formellement les conditions et algorithmes associés et j'ai étudié l'évaluation polynomiale par l'algorithme de Horner.

J'ai montré la faisabilité de preuves formelles dans le domaine complexe de l'arithmétique des ordinateurs. J'ai déterminé les points forts et les limites de cette démarche en obtenant un recul suffisant face à cette formalisation par différents moyens : enrichissement de la bibliothèque, extensions du modèle et validation de vraies applications.

Mots-clés :

Arithmétique des ordinateurs, Nombres à virgule flottante, Preuve formelle, Coq.

Abstract:

This work is representative of my experiments on joining computer arithmetic, directed by the IEEE-754 standard, and formal proofs, here the Coq proof assistant. The floating-point number formalization used was first developed by L. Théry.

I first tested and expanded the library with properties that are easy to express in our formalism: the fact that a real value may be represented exactly by a floating-point number. I then tried some extensions of the model: bring it closer to the hardware realities, generalize it to use the two's complement representation and go into a weaker rounding. With these results, I studied two applications. The first one is a multi-precision library using expansions. The second one is the evaluation of elementary functions (logarithm, cosine...): I solved the main problems of argument reduction by formally proving the conditions and algorithms involved and I studied the polynomial evaluation using Horner's rule.

I showed the possibility to make formal proofs about the difficult topic of computer arithmetic. I found the advantages and drawbacks of this method by getting enough distance from the formalization by three different means: supplements to the library, extensions of the formalization and validation of real-life applications.

Keywords:

Computer arithmetic, Floating-point numbers, Formal proof, Coq.