# Arrondis dirigés efficaces dans Coq

**Proposition de stage de L3**

Équipe ACADIE, Lab. IRIT, Université Toulouse 3

Janvier 2019

**Informations pratiques** Le stage se déroulera à l'*Institut de Recherche en Informatique de Toulouse* (IRIT) sur le campus de l'*Université Paul Sabatier* (UPS). Il sera encadré par Érik MARTIN-DOREL (erik.martin-dorel@irit.fr) et Pierre ROUX (pierre.roux@onera.fr).

**Contexte** Les assistants de preuve sont des logiciels complexes qui permettent d'encoder des théories mathématiques et des algorithmes dans un langage formel, de formuler des propriétés et des théorèmes sur ces algorithmes, de développer des preuves pour ces propriétés et de vérifier mécaniquement ces preuves formalisées avec un très haut niveau de garantie.

Grâce à sa logique sous-jacente très expressive, l'assistant de preuve Coq offre un support intégré pour les calculs. Il propose plusieurs tactiques de réduction comme `compute`, `vm_compute` [3] et `native_compute` [1] qui peuvent tirer parti de la compilation vers du bytecode ou du code natif OCaml, de façon à améliorer la performance des calculs considérés, au prix d'une base de confiance légèrement plus grande.

De plus, des travaux ont été menés pour étendre Coq avec des librairies arithmétiques (`BigN`, `BigZ`, `BigQ`) [4] qui tirent parti de l'efficacité des entiers machine.

Couplé avec les fonctionnalités de preuve par réflection de Coq, ces tactiques de réduction et ces types de données efficaces ont joué un rôle clef dans de nombreux travaux de certification formelle de calculs numériques ou symboliques.

Un exemple de ces travaux est donné par la librairie CoqInterval [7, 6] : une tactique réflexive pour prouver formellement et automatiquement des inégalités sur des expressions à variable réelle. Cette tactique se base essentiellement sur des techniques d'arithmétique d'intervalles, les bornes des intervalles calculés étant des nombres à virgule flottante émulés par des entiers `BigZ`.

Cette arithmétique flottante émulée étant environ trois ordres de grandeur plus lente que les opérations flottantes natives du processeur, nous avons récemment développé [1] une version de Coq [2] permettant l'utilisation directe des opérations flottantes du processeur. Toutefois ce développement offre uniquement accès aux opération arrondies au plus proche alors que l'arithmétique d'intervalles fait un grand usage des arrondis dirigés (vers $-\infty$ et $+\infty$). La norme IEEE754 [5] définissant l'arithmétique flottante implémentée par les processeurs spécifie bien ces arrondis mais leur usage passe généralement par un mécanisme de changement du mode d'arrondi, valable ensuite pour toutes les

1. En grande partie durant un précédent stage de L3.
2. https://github.com/validsdp/coq/tree/primitive-floats

opérations suivantes. Ce changement de mode d'arrondi, relativement coûteux, est fondamentalement impératif (modification de l'état du processeur) et s'inscrit mal dans le paradigme de programmation fonctionnel de Coq.

La solution actuellement adoptée est l'usage des fonctions successeur/prédécesseur (parmi les nombres flottants) permettant de surapproximer les arrondis dirigés à partir de l'arrondi au plus proche. Ces fonctions sont actuellement implémentées en OCaml ou en C par des manipulations de la représentation binaire des flottants mais il est connu qu'elles peuvent être implémentées uniquement à l'aide d'opération arithmétiques arrondies au plus proche [9, 10]. Cette possibilité ouvre donc la voie à une implémentation entièrement en Coq donc prouvable et limitant la base de confiance aux opérations arrondies au plus proche.

Nous souhaiterions donc comparer les performances de ces différentes implémentations (d'approximations) des arrondis dirigés, en terme de précision mais surtout de temps de calcul. Si une implémentation en Coq s'avère pertinente, il sera alors possible d'en prouver la correction, voire la précision.

Au delà de l'impact sur la librairie CoqInterval, ces travaux pourraient améliorer l'état de l'art actuel des calculs numériques certifiés utilisant Coq et pourrait constituer une motivation pour rendre des approches de calcul numériques rigoureux [8] disponibles en preuve formelle.

**Contribution attendue** Ce stage vise à évaluer la pertinence de l'implémentation en Coq d'approximations des arrondis dirigés en arithmétique à virgule flottante en se basant sur l'arrondi au plus proche. Les aspects à étudier comprennent bien entendu le temps d'exécution mais aussi la preuve formelle de ces implémentations.

Le développement formel pourra se baser sur la librairie Flocq [2] formalisant l'arithmétique à virgule flottante en Coq.

**Pré-requis** Le candidat doit avoir un intérêt pour la programmation fonctionnelle et la logique mathématique. Une expérience préalable de Coq ou d'un autre assistant de preuve serait appréciée mais n'est pas nécessaire. Aucune connaissance des librairies Coq citées dans ce sujet (Flocq, CoqInterval) ni de l'arithmétique flottante ou de l'arithmétique d'intervalle n'est attendue.

## Références

[1] Mathieu Boespflug, Maxime Dénès, and Benjamin Grégoire. Full reduction at full throttle. In Jean-Pierre Jouannaud and Zhong Shao, editors, *Certified Programs and Proofs - First International Conference, CPP 2011, Kenting, Taiwan, December 7-9, 2011. Proceedings*, volume 7086 of *Lecture Notes in Computer Science*, pages 362–377. Springer, 2011. doi:10.1007/978-3-642-25379-9_26.

[2] Sylvie Boldo and Guillaume Melquiond. Flocq : A Unified Library for Proving Floating-Point Algorithms in Coq. In Elisardo Antelo, David Hough, and Paolo Ienne, editors, *IEEE Symposium on Computer Arithmetic*, pages 243–252. IEEE Computer Society, 2011. doi:10.1109/ARITH.2011.40.

[3] Benjamin Grégoire and Xavier Leroy. A compiled implementation of strong reduction. In Mitchell Wand and Simon L. Peyton Jones, editors, *Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming (ICFP '02), Pittsburgh, Pennsylvania, USA, October 4-6, 2002.*, pages 235–246. ACM, 2002. doi:10.1145/581478.581501.

[4] Benjamin Grégoire and Laurent Théry. A Purely Functional Library for Modular Arithmetic and Its Application to Certifying Large Prime Numbers. In Ulrich Furbach and Natarajan Shankar, editors, *IJCAR*, volume 4130 of *Lecture Notes in Computer Science*, pages 423–437. Springer, 2006. doi:10.1007/11814771_36.

[5] IEEE Computer Society. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754–2008*, August 2008. doi:10.1109/IEEESTD.2008.4610935.

[6] Érik Martin-Dorel and Guillaume Melquiond. Proving Tight Bounds on Univariate Expressions with Elementary Functions in Coq. *Journal of Automated Reasoning*, pages 1–31, 2015. URL : https://hal.inria.fr/hal-01086460v2, doi:10.1007/s10817-015-9350-4.

[7] Guillaume Melquiond. Floating-point arithmetic in the Coq system. *Inf. Comput.*, 216 :14–23, 2012. doi:10.1016/j.ic.2011.09.005.

[8] Siegfried Rump. Verification methods : Rigorous results using floating-point arithmetic. *Acta Numerica*, 19 :287–449, May 2010. URL : http://journals.cambridge.org/article_S096249291000005X, doi:10.1017/S096249291000005X.

[9] Siegfried M Rump, Takeshi Ogita, Yusuke Morikura, and Shin'ichi Oishi. Interval arithmetic with fixed rounding mode. *Nonlinear Theory and Its Applications, IEICE*, 7(3) :362–373, 2016.

[10] Siegfried M. Rump, Paul Zimmermann, Sylvie Boldo, and Guillaume Melquiond. Computing predecessor and successor in rounding to nearest. *BIT Numerical Mathematics*, 49(2) :419–431, Jun 2009. URL : https://doi.org/10.1007/s10543-009-0218-z, doi:10.1007/s10543-009-0218-z.