Détection des cas de débordement flottant avec une recherche locale

Mohamed Sayah, Yahia Lebbah

Laboratoire I3S/CNRS, Université de Nice Sophia Antipolis, France
Laboratoire LITIO, Université d'Oran Es sénia, Algérie
{sayahmh, ylebbah}@gmail.com

Résumé

Dans ce papier, nous proposons une approche de génération automatique des cas de test de programmes de calcul numérique, qui donnent lieu à des débordements flottants. Afin de résoudre les contraintes sur les flottants modélisant cette problématique, nous proposons un algorithme de recherche locale mis en œuvre avec la bibliothèque multi-précision MPFR. Notre démarche de résolution procède en deux étapes : exploitation d'une recherche locale numérique pour avoir une solution approchée sur le domaine réel, puis un algorithme spécifique de recherche locale démarrant à partir de la solution de la première recherche locale, pour déterminer la solution exacte qui mène à un débordement flottant. Nous exploitons la bibliothèque multi-précision MPFR, tout au long de notre résolution en deux étapes, afin de sécuriser le solveur contre les débordements lors de la recherche numérique de la solution flottante.

1 Introduction

Les algorithmes numériques sont conçus et prouvés sur les nombres réels $\mathbb{R}$, alors que leur mise en œuvre est faite sur les nombres flottants $\mathbb{F}$ qui ne respectent pas les propriétés mathématiques des nombres réels. En pratique, l'ordinateur calcule avec l'arithmétique à virgule flottante une approximation de l'arithmétique réelle. Cette approximation se traduit par des erreurs d'arrondi ou parfois des erreurs soudaines de débordement au niveau des calculs.

Considérons l'algorithme `square` ci-dessous qui calcule la racine carrée r de deux nombres réels x et y. Dans le cas où x est égal à y, r prend la valeur de x.

1: **if** $(x = y)$ **then**
2: $\quad r \leftarrow x$;
3: **else**
4: $\quad r \leftarrow sqrt(x * y)$;
5: **end if**
6: **return** r ;

Le programme `square` avec le cas de test $x = 1.5e + 180$ et $y = 5.03e + 129$ provoque l'exception $Overflow$; le cas de test $x = 1e{-}200$ et $y = 5.03e{-}129$ provoque l'exception $Underflow$. C'est ce que nous traitons dans cet article, les cas de test relatifs au débordement flottant dans les programmes informatiques. Nous notons que parmi les méthodes de résolution, seules les méthodes globales ont été étudiées dans [5, 1]. Cependant, et vu que l'espace des flottants a une taille de nature fortement exponentielle (Par exemple, le nombre de flottants entre 0 et 1 dépend de la taille de ces flottants, il existe approximativement 2147483648 flottants.), ces méthodes se trouvent confrontées à la combinatoire qui rend impossible l'exploration de l'espace $\mathbb{F}$ des flottants. Ainsi, le recours aux techniques locales de résolution s'avère incontournable.

Notre approche adopte l'hypothèse suivante : "*les solutions d'un système de contraintes donné sur les flottants sont proches des solutions du même système sur les réels.*". D'abord, les systèmes de contraintes sur les flottants provenant d'un calcul numérique sont en pratique sur les réels. Le recours aux variables à virgule flottante est dû à l'indisponibilité des nombres réels sur machine. Puis, la solution sur les flottants est une approximation (qui peut être large dans certains cas) de la solution sur les réels.

Notre hypothèse nous motive à faire appel initialement à une première recherche locale sur les réels. Cette première recherche locale va nous fournir une première solution approchée. Ici, comme les algorithmes de recherche locale dans le domaine continu ne peuvent pas trouver des cas (solutions) de débor-

dement sans faire déborder le solveur lui même, nous recourons à une mise en œuvre avec la bibliothèque MPFR [3].

Justement, pour remédier à l'inexactitude sur les flottants de cette première solution, nous faisons appel à une deuxième recherche locale qui démarre à partir de la solution approchée fournie par la première recherche locale.

2 Description de l'approche

Dans ce contexte de génération automatique des cas de test qui donnent lieu à des débordements flottants, l'approche procède par les étapes suivantes :

1. Le programme de calcul numérique est transformé en un système de contraintes sur les flottants en exploitant la forme SSA [4].
2. Le critère exprimant le débordement est écrit sous forme de contraintes tout en exploitant les valeurs extrémales des flottants (i.e. $max(\mathbb{F})$ et $min(\mathbb{F})$). A la fin de cette étape, on obtient le problème global de satisfaction de contraintes $\mathcal{CSP}_{\mathbb{F}}$ dont les solutions sont nécessairement les cas de débordement du programme initial.
3. Une première solution approchée du $\mathcal{CSP}_{\mathbb{F}}$ est produite avec une recherche locale classique, à savoir la méthode du gradient en supposant que les domaines sont réels.
4. Une deuxième résolution à base d'une recherche locale dédiée pour les nombres flottants, permet de déterminer une solution exacte du système de contraintes et donc du débordement.

   Le principe de cette recherche locale est classique. Elle consiste à démarrer d'une instanciation initiale I_0, fournie par la première recherche locale sur les réels, et d'essayer ensuite de l'améliorer, en cherchant une meilleure instanciation, relativement à une fonction coût à minimiser, dans le voisinage de celle ci. Initialement l'instanciation courante I_c est égale à I_0. L'algorithme s'arrête une fois qu'il a atteint une instanciation qui est une solution exacte du système de contraintes sur les flottants. Comme toute recherche locale, elle s'arrête aussi si elle dépasse un nombre maximum d'itérations donné par l'utilisateur.

   L'espace de voisinage d'une instanciation I_c est l'ensemble de toutes les instanciations distantes d'un seul flottant de I_c. La fonction de coût, à minimiser, d'une instanciation est la somme des violations des différentes contraintes du problème. Les contraintes traitées sont les contraintes d'égalité de la forme $(e_1 = e_2)$, et les contraintes d'inégalité large $(e_1 \leq e_2)$; avec e_1 et e_2 des expressions données. La fonction $evalC$ d'évaluation du coût de violation d'une contrainte est donnée comme suit :

$$\begin{array}{lcl} evalC(e_1 = e_2, I_c) & = & |v_1 - v_2|, \\ evalC(e_1 \leq e_2, I_c) & = & \text{si } v_1 \leq v_2 \text{ alors } 0 \\ & & \text{sinon } (v_1 - v_2). \end{array}$$

3 Discussion et conclusion

Dans ce papier, nous avons introduit une approche de génération automatique des cas de test relatifs au débordement pour les programmes de calcul numérique. Cette approche procède en deux étapes : une première étape de résolution sur les nombres réels produisant une solution approchée sur les flottants ; puis une deuxième étape avec une recherche locale visant à trouver une solution exacte sur les flottants.

Les expérimentations de cette démarche ont montré la nécessité d'une haute précision dans la première recherche locale pour que la deuxième recherche locale puisse explorer un espace réduit sur les flottants. D'où la première perspective, celle d'améliorer algorithmiquement la deuxième recherche locale pour qu'elle nécessite moins de précision dans la première recherche locale. La deuxième perspective est de comparer les résultats de notre approche avec les démarches de vérification des programmes de calcul numérique, et tout particulièrement l'outil Astrée [2].

Références

[1] B. Botella, A. Gotlieb, and C. Michel. Symbolic execution of floating-point computations. *Softw. Test., Verif. Reliab.*, 16(2) :97–121, 2006.

[2] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Varieties of static analyzers : A comparison with Astrée, invited paper. In *Proc. First IEEE & IFIP International Symposium on Theoretical Aspects of Software Engineering, TASE '07*, pages 3–17, Shanghai, China, 6–8 June 2007.

[3] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. MPFR : A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw*, 33(2) :13, 2007.

[4] A. Gotlieb. *Automatic Test Data Generation using Constraint Logic Programming.* PhD thesis, Université de Nice — Sophia Antipolis, France, 2000.

[5] C. Michel, M. Rueher, and Y. Lebbah. Solving constraints over floating point numbers. *Lecture Notes in Computer Science (LNCS)*, pages 2239 :524–538, 2001.