Un nouvel algorithme de consistance locale sur les nombres flottants

Mohammed Said Belaid, Claude Michel, Michel Rueher

Un nouvel algorithme de consistance locale sur les nombres flottants

Mohammed Said BELAID, Claude MICHEL, Michel RUEHER

I3S UNS/CNRS, 2000 route des Lucioles, BP 121, 06903 Sophia Antipolis Cedex, France
{MSBelaid, Claude.Michel}@i3s.unice.fr, Michel.Rueher@gmail.com

Résumé

La résolution de contraintes sur les nombres à virgule flottante soulève des problèmes critiques dans de nombreuses applications, notamment en vérification de programmes. Jusqu'à maintenant, les algorithmes de filtrage sur les nombres à virgule flottante ont été limités à la 2B–consistance et ses dérivées. Bien que ces filtrages soient conservatifs des solutions, ils souffrent des problèmes bien connus des consistances locales, e.g., leur incapacité à traiter efficacement les occurrences multiples de variables. Leurs limitations proviennent aussi de la pauvreté des propriétés de l'arithmétique des nombres à virgule flottante. Afin de pallier à ces limitations, nous proposons dans cet article un nouvel algorithme de filtrage de contraintes sur les flottants qui repose sur des relaxations successives sur les réels du problème initial sur les flottants. Des bornes conservatives des domaines sont obtenues à l'aide d'un solveur de programme linéaire mixte (MILP) appliquées à des linéarisations conservatives de ces relaxations. Les résultats préliminaires sont prometteurs et montrent que cette approche peut effectivement accélérer les filtrages par consistances locales.

Abstract

Solving constraints over floating-point numbers is a critical issue in numerous applications notably in program verification. Capabilities of filtering algorithms over the floating-point numbers have been so far limited to 2b-consistency and its derivatives. Though safe, such filtering techniques suffer from the well known pathological problems of local consistencies, e.g., inability to efficiently handle multiple occurrences of the variables. These limitations also take roots in the strongly restricted floating-point arithmetic. To circumvent the poor properties of floating-point arithmetic, we propose in this paper a new filtering algorithm which relies on various relaxations over the reals of the problem over the floats. Safe bounds of the domains are computed with a mixed integer linear programming solver (MILP) on safe linearizations of these relaxations. Preliminary experiments on a relevant set of benchmarks are very promising and show that this approach can be very effective for boosting local consistency algorithms over the floats.

1 Introduction

Les logiciels critiques s'appuient de plus en plus sur le calcul en virgule flottante. Par exemple, les systèmes embarqués sont contrôlés par des logiciels qui utilisent des mesures et des données de leur environnement évaluées par des nombres flottants. Les calculs effectués sur ces nombres font l'objet d'erreurs d'arrondi. Ces erreurs d'arrondi peuvent avoir des conséquences importantes et même, modifier le flot de contrôle du programme. Ainsi, la vérification de programmes qui effectuent des calculs sur les nombres flottants est un problème clef dans le processus de développement de systèmes critiques.

Les méthodes de vérification de programmes numériques sont principalement issues des méthodes classiques de vérification de programmes. La vérification bornée de modèles (BMC) a largement été utilisée pour détecter des bugs lors de la conception de processeurs [3] ou de logiciels [10]. Les solveurs SMT sont aujourd'hui présents dans la plupart des outils de BMC pour manipuler directement des formules de haut niveau [2, 8, 10]. L'outil CBMC de vérification bornée de modèles code chacune des opérations sur les flottants comme une fonction logique sur des vecteurs de bits. Cet encodage nécessite l'introduction de milliers de variables additionnelles et rend souvent le problème insoluble [6]. D'autres outils basés sur l'interprétation abstraite [9, 20] peuvent montrer l'absence d'erreurs d'exécution (e.g., la division par zéro) dans des programmes numériques. Grâce à une sur-

approximation du calcul sur les flottants, ces outils sont conservatifs de l'ensemble des solutions. Cependant, cette sur-approximation peut être très large et, par conséquent, ces outils rejettent un nombre significatif de programmes valides. La programmation par contraintes (CP) a aussi été utilisée pour la génération de cas de tests [12, 13] et la vérification de programmes [7]. La CP offre de multiples bénéfices tels que la capacité de déduire des informations de problèmes partiellement instanciés ou de fournir des contre-exemples. La programmation par contraintes est dotée d'un cadre très flexible qui facilite l'intégration de nouveau solveurs sur des domaines spécifiques comme les solveurs sur les flottants. Notons que les solveurs sur les réels ne sont pas capable de traiter correctement l'arithmétique sur les flottants. Des solveurs dédiés et corrects sont donc requis par les outils de programmation par contraintes et de vérification bornée de modèles pour tester ou vérifier des programmes numériques [1].

Les techniques existantes de résolution de contraintes sur les flottants sont basées sur une adaptation des consistances locales sur les réels (e.g. box-consistance, 2B-consistance) [19, 18, 5]. Toutefois, ces solveurs peinent à passer à l'échelle. C'est pourquoi nous introduisons ici une nouvelle méthode qui s'appuie sur des solveurs sur les réels pour filtrer des contraintes sur les nombres à virgule flottante. L'idée principale est de construire des relaxations fines et conservatives sur les réels pour des contraintes sur les flottants. Afin d'obtenir des relaxations fines, chaque opération sur les flottants est approximée selon le mode d'arrondi. Par exemple, supposons que x et y sont des nombres flottants positifs normalisés [2], alors le produit $x \otimes y$, pour un mode d'arrondi vers $-\infty$, sera borné par :

$$\alpha \times (x \times y) < x \otimes y \leq x \times y$$

où $\alpha = 1/(1 + 2^{-p+1})$ et p est la taille de la mantisse. Nous avons aussi défini des approximations fines pour les autres cas tels que l'addition avec un mode d'arrondi au plus près ou la multiplication par une constante.

À l'aide de ces relaxations, le problème initial sur les nombres flottants est tout d'abord transformé en un ensemble de contraintes non-linéaires sur les réels. Une linéarisation des contraintes non-linéaires est ensuite effectuée afin d'obtenir un problème linéaire mixte (MILP) sur les réels. Lors de ce processus, des variables binaires permettent de traiter les parties concaves tout en évitant l'utilisation de sur-approximations trop lâches. Ce dernier ensemble de contraintes peut directement être résolu par les solveurs MILP disponibles sur les réels qui, eux, ne sont par limités par l'arithmétique des nombres flottants. Les solveurs MILP efficaces utilisent des calculs en virgule flottante et peuvent donc perdre des solutions. Afin d'obtenir un comportement correct de tel solveurs, des procédures d'arrondi correct sont appliquées aux coefficients des relaxations [17, 4] et la méthode décrite dans [21] est utilisée pour calculer un minimum correct à partir du résultat incorrect fourni par un solveur MILP. Les résultats préliminaires sont très prometteurs et montrent que la technique introduite ici peut faciliter le passage à l'échelle des outils nécessitant un solveur de contraintes sur les flottants.

Notre méthode repose sur une représentation de haut niveau des opérations sur les flottants et ne souffre donc pas des mêmes limitations que celle reposant sur un encodage en vecteur de bits. Cet encodage utilisé par CBMC génère des milliers de variables binaires additionnelles pour chacune des opérations sur les flottants du programme. Par exemple, une addition de deux variables flottantes codées sur 32 bits demande 2554 variables binaires [6]. L'approximation mixte proposée dans [6] réduit notablement le nombre de ces variables. Mais le système résultant reste encore coûteux en mémoire. Notons qu'une simple addition avec seulement 5 bits de précision nécessite encore 1035 variables additionnelles. Notre méthode génère aussi des variables additionnelles : des variables intermédiaires sont ajouter afin de décomposer les expressions complexes en en opérations élémentaires sur les flottants et quelques variables binaires permettent de gérer les différents cas de nos relaxations. Cependant, la quantité de variables additionnelles ainsi générées reste négligeable en comparaison de celle requise par le modèle à base de vecteurs de bits.

1.1 Un exemple illustratif

Avant de détailler notre méthode, illustrons notre approche sur un exemple très simple. Considérons la contrainte suivante :

$$z = x + y - x \tag{1}$$

où x, y et z sont des variables de type flottant sur 32 bits. Sur les nombres réels, cette expression peut évidemment être réduite à $z = y$. Ce n'est pas le cas pour les nombres flottants. Par exemple, sur les

1. voir, par exemple, FPSE (http ://www.irisa.fr/celtique/carlier/fpse.html), un solveur de contraintes sur les flottants issue de programmes C.

2. Un nombre flottant x est déterminé par un triplet (s, e, m) où s est son signe, e son exposant et m sa mantisse. Sa valeur est donnée par $(-1)^s \times 1.m \times 2^e$. r et p spécifient le nombre de bits de son exposant et de sa mantisse. Dans la norme IEEE 754, les simples sont définis par $(r, p) = (8, 23)$ et les doubles par $(r, p) = (11, 52)$.

flottants et avec un mode d'arrondi au plus proche $10.0 + 10^{-8} - 10.0$ n'est pas égal à 10^{-8} mais à 0. Ce phénomène d'absorption montre bien pourquoi les expressions sur les flottants ne peuvent pas être simplifiées de la même manière que les expressions sur les réels.

Supposons que $x \in [0.0, 10.0]$, $y \in [0.0, 10.0]$ et $z \in [0.0, 10.0^8]$. FP2B, un algorithme de 2B-consistance [15] adapté aux flottants [5], propage les domaines de x et y vers le domaine de z en utilisant l'arithmétique des intervalles. La propagation inverse n'étant d'aucune utilité ici, le processus de filtrage donne :

$$x \in [0.0, 10.0],\ y \in [0.0, 10.0],\ z \in [0.0, 20.0]$$

Ce résultat souligne les difficultés des algorithmes de filtrages classiques à traiter les occurrences multiples. Une consistance plus forte comme la 3B-consistance [15] peut réduire le domaine de z à $[0.0, 10.01835250854492188]$. Par contre, la 3B-consistance ne réussit pas à réduire le domaine de z lorsque x et y ont tous les deux plus de deux occurrences comme dans la contrainte $z = x + y - x - y + x + y - x$.

L'algorithme introduit dans cet article construit d'abord une relaxation conservative non linéaire sur les réels du problème initial sur les flottants. Appliqué à la contrainte 1, il produit les relaxations sur les réels suivantes :

$$\begin{cases} (1 - \frac{2^{-p}}{(1-2^{-p})})(x+y) \leq tmp1 \\ tmp1 \leq (1 + \frac{2^{-p}}{(1+2^{-p})})(x+y) \\ (1 - \frac{2^{-p}}{(1-2^{-p})})(tmp1 - x) \leq tmp2 \\ tmp2 \leq (1 + \frac{2^{-p}}{(1+2^{-p})})(tmp1 - x) \\ z = tmp2 \end{cases}$$

où p est la taille de la mantisse de la variable flottante. $tmp1$ approxime le résultat de l'opération $x \oplus y$ à l'aide de deux plans sur les réels qui capturent les résultats de l'addition sur les flottants. $tmp2$ fait la même chose pour la soustraction. Certaines relaxations, comme la multiplication, contiennent des termes non-linéaires. Dans ce cas là, un processus de linéarisation des termes non-linéaires est appliqué afin d'obtenir un programme linéaire. Une fois le système complètement linéaire, un solveur MILP réduit le domaine de chaque variable en en calculant successivement le minimum et le maximum.

FPLP (Floating-point linear programming) est un outil qui implémente l'algorithme précédemment esquissé. Un appel à FPLP sur la contrainte (1) donne le résultat suivant :

$$x \in [0, 10],\ y \in [0, 10],\ z \in [0, 10.0000023841859]$$

qui est beaucoup plus précis que celui de FP2B. Contrairement à la 3B-consistance, FPLP donne le même résultat avec $z = x + y - x - y + x + y - x$.

1.2 Organisation de l'article

La suite de cet article est organisée comme suit : la section suivante introduit les relaxations non-linéaires sur les réels des contraintes sur les flottants. Le processus de linéarisation des relaxations est ensuite décrit avant de détailler l'algorithme de filtrage. Les résultats expérimentaux sont présentés dans la section qui précède la conclusion.

2 Relaxation des contraintes sur les flottants

Cette section introduit les relaxations sur les réels des contraintes sur les flottants issues du problème initial. Ces relaxations sont la pierre angulaire du processus de filtrage décrit dans cet article. Elles doivent non seulement être *correctes*, i.e., préserver l'ensemble des solutions du problème initial, mais aussi *fines*, i.e., inclure le moins possible de flottants non solution.

La construction de ces relaxations repose sur deux techniques : l'*erreur relative* et les opérations *correctement arrondies*. La première de ces techniques est communément utilisée pour analyser la précision du calcul. La seconde propriété est garantie par toute implémentation conforme au standard IEEE 754 de l'arithmétique des flottants : une opération correctement arrondie est une opération dont le résultat sur les flottants est égal à l'arrondi du résultat de l'opération équivalente sur les réels. En d'autre terme : soit x et y deux nombres flottants, $\odot$ et $\cdot$, respectivement, une opération sur les flottants et son équivalent sur les réels, si $\odot$ est correctement arrondie alors, $x \odot y = round(x \cdot y)$.

La suite de cette section détaille d'abord la construction de ces relaxations pour un cas particulier avant de la généraliser aux autres cas. Nous montrons ensuite comment les différents cas peuvent être simplifiés.

2.1 Un cas particulier

Afin d'expliquer comment obtenir ces relaxations, considérons d'abord le cas où le mode d'arrondi est vers $-\infty$ et le résultat de l'opération est un nombre flottant positif et normalisé. Un telle opération, dénotée $\odot$, peut être n'importe laquelle des quatre opérations de base de l'arithmétique des flottants. Toutes les opérandes sont supposées être du même type, i.e., float, double ou long double. On a alors la propriété suivante :

Proposition 1. *Soient x et y, des nombres flottants dont la mantisse possède p bits. Supposons que le mode d'arrondi soit fixé $-\infty$ et que le résultat de $x \odot y$ soit un nombre positif normalisé strictement inférieur à max_{float}, le plus grand des flottants, alors on a*

$$\frac{1}{1+2^{-p+1}}(x \cdot y) < x \odot y \leq (x \cdot y)$$

où $\odot$ une opération basique sur les nombres flottants et $\cdot$ est l'opération équivalente sur les nombres réels.

Démonstration. Selon la norme IEEE 754, les opérations sont correctement arrondies et le mode d'arrondi est fixé à $-\infty$. On a donc :

$$x \odot y \leq x \cdot y < (x \odot y)^+ < max_{float} \tag{2}$$

$(x \odot y)^+$, le successeur de $(x \odot y)$ dans l'ensemble des nombres à virgule flottante, peut être calculé par

$$(x \odot y)^+ = (x \odot y) + ulp(x \odot y)$$

puisque, par définition, $ulp(x) = x^+ - x$. Il résulte donc de (2) que

$$x \odot y \leq x \cdot y < (x \odot y) + ulp(x \odot y)$$

La seconde inéquation peut être réécrite comme suit :

$$\frac{1}{x \odot y + ulp(x \odot y)} < \frac{1}{x \cdot y}$$

En multipliant les deux parties de l'inéquation par $x \odot y$ – qui est un nombre positif – on obtient :

$$\frac{x \odot y}{x \odot y + ulp(x \odot y)} < \frac{x \odot y}{x \cdot y}$$

En multipliant chaque partie de l'inéquation précédente par -1 et en ajoutant 1 à chaque membre, on obtient

$$1 - \frac{x \odot y}{x \cdot y} < 1 - \frac{x \odot y}{x \odot y + ulp(x \odot y)} = \frac{ulp(x \odot y)}{x \odot y + ulp(x \odot y)} \tag{3}$$

Considérons maintenant ϵ, l'erreur relative définie par

$$\epsilon = \left| \frac{real\_value - float\_value}{real\_value} \right|$$

ϵ est la valeur absolue de la différence entre le résultat sur les réels et le résultat sur les flottants divisé par le résultat sur les réels. Dans notre cas, le résultat de $x \odot y$ étant un flottant positif normalisé, et sachant que $x \cdot y \geq x \odot y$, l'erreur relative est donnée par :

$$0 \leq \epsilon = \frac{x \cdot y - x \odot y}{x \cdot y} = 1 - \frac{x \odot y}{x \cdot y}$$

Donc, grâce à (3), on a

$$0 \leq \epsilon < \frac{ulp(x \odot y)}{x \odot y + ulp(x \odot y)}$$

z, le résultat de l'opération $x \odot y$, est un nombre flottant binaire positif et normalisé qui peut s'écrire $z = 1.m_z 2^{e_z}$, où m_z, sa mantisse, a p bits. Par ailleurs, $ulp(z) = 2^{-p+1}2^{e_z}$. On a donc,

$$0 \leq \epsilon < \frac{2^{-p+1}2^{e_z}}{m_z 2^{e_z} + 2^{-p+1}2^{e_z}} = \frac{2^{-p+1}}{m_z + 2^{-p+1}}$$

La valeur de la mantisse pour un nombre flottant normalisé appartient à l'intervalle $[1.0, 2.0[$. La borne supérieure de l'erreur relative ϵ est donnée par le minimum de $m_z + 2^{-p}$ qui est atteint lorsque $m_z = 1$. Donc

$$0 \leq \epsilon < \frac{2^{-p+1}}{1+2^{-p+1}}$$

Puisqu'on a

$$\epsilon = \frac{x \cdot y - x \odot y}{x \cdot y}$$

On a

$$0 \leq \frac{x \cdot y - x \odot y}{x \cdot y} < \frac{2^{-p+1}}{1+2^{-p+1}}$$

et

$$0 \leq x \cdot y - x \odot y < (x \cdot y)\frac{2^{-p+1}}{1+2^{-p+1}}$$

En multipliant chaque partie par -1 et en ajoutant $x \cdot y$, on obtient finalement

$$\frac{1}{1+2^{-p+1}}(x \cdot y) < x \odot y \leq x \cdot y$$

□

2.2 Généralisation

Le tableau 1 donne les relaxations pour les différents modes d'arrondi et les différent cas, i.e., les nombres flottants positifs ou négatifs, normalisés ou non. À chaque cas est associée une approximation correcte et fine obtenue de façon similaire à celle détaillée dans la section précédente.

Notez que certains cas particuliers permettent d'améliorer la précision des relaxations. Par exemple, l'addition avec un mode d'arrondi vers $\pm\infty$ peut être légèrement améliorée. La structure du problème offre aussi des possibilités d'amélioration des approximations. Par exemple, $2 \otimes x$ étant calculé exactement [3], cette expression peut directement être évaluée sur les réels.

3. S'il n'y a pas de dépassement de capacité.

Mode d' arrondi	Négatif normalisé	Négatif dénormalisé	Positif dénormalisé	Positif normalisé
vers $-\infty$	$[(1+2^{-p+1})z_r, z_r]$	$[z_r - min_f, z_r]$	$[z_r - min_f, z_r]$	$[\frac{1}{(1+2^{-p+1})}z_r, z_r]$
vers $+\infty$	$[z_r, \frac{1}{(1+2^{-p+1})}z_r]$	$[z_r, z_r + min_f]$	$[z_r, z_r + min_f]$	$[z_r, (1+2^{-p+1})z_r]$
vers 0	$\left[z_r, \frac{1}{(1+2^{-p+1})}z_r\right]$	$[z_r - min_f, z_r]$	$[z_r, z_r + min_f]$	$[\frac{1}{(1+2^{-p+1})}z_r, z_r]$
au plus près	$[(1+\frac{2^{-p}}{(1+2^{-p})})z_r,$ $(1-\frac{2^{-p}}{(1-2^{-p})})z_r]$	$[z_r - \frac{min_f}{2},$ $z_r + \frac{min_f}{2}]$	$[z_r - \frac{min_f}{2}$ $z_r + \frac{min_f}{2}]$	$[(1-\frac{2^{-p}}{(1-2^{-p})})z_r,$ $(1+\frac{2^{-p}}{(1+2^{-p})})z_r]$

TABLE 1 – Relaxations de $x \odot y$ pour chaque mode d'arrondi, avec $z_r = x \cdot y$.

2.3 Simplification des relaxations

Le principal problème des relaxations précédentes est que le processus de résolution doit traiter les différents cas. Ainsi, pour n opérations élémentaires, le solveur devra potentiellement traiter 4^n combinaisons de relaxations. Afin de diminuer substantiellement cette complexité, nous décrivons ici une combinaison des quatre cas liés à chaque arrondi en une unique relaxation.

Considérons d'abord le cas où le mode d'arrondi est fixé à $-\infty$:

Proposition 2. *Soient x et y deux nombres flottants dont la taille de la mantisse est p. Supposons que le mode d'arrondi est fixé $-\infty$ et que le résultat de $x \odot y$ est tel que $-max_{float} < x \odot y < max_{float}$. Alors, on a :*

$$z_r - 2^{-p+1}|z_r| - min_f \leq x \odot y \leq z_r$$

où min_f est le plus petit nombre flottant positif, $\odot$ et $\cdot$ sont, respectivement, une opération arithmétique de base sur les flottants et son équivalent sur les réels, et $z_r = x \cdot y$.

Démonstration. La première étape consiste à combiner les approximations des nombres normalisés et dénormalisés. Si $z_r > 0$ alors $\frac{1}{1+2^{-p+1}}z_r < z_r$. Donc,

$$\frac{1}{1+2^{-p+1}}z_r - min_f < z_r - min_f$$

et

$$\frac{1}{1+2^{-p+1}}z_r - min_f < \frac{1}{1+2^{-p+1}}z_r$$

Il s'en suit que :

$$\frac{1}{1+2^{-p+1}}z_r - min_f < x \odot y \leq z_r,\ z_r \geq 0$$

De la même manière, lorsque $z_r \leq 0$, on a :

$$(1+2^{-p+1})z_r - min_f < x \odot y \leq z_r,\ z_r \leq 0$$

Mode d'arrondi	Approximation de $x \odot y$
vers $-\infty$	$[z_r - 2^{-p+1}\lvert z_r\rvert - min_f, z_r]$
vers $+\infty$	$[z_r, z_r + 2^{-p+1}\lvert z_r\rvert + min_f]$
vers 0	$[z_r - 2^{-p+1}\lvert z_r\rvert - min_f,$ $z_r + 2^{-p+1}\lvert z_r\rvert + min_f]$
au plus près	$[z_r - \frac{2^{-p}}{(1-2^{-p})}\lvert z_r\rvert - \frac{min_f}{2},$ $z_r + \frac{2^{-p}}{(1-2^{-p})}\lvert z_r\rvert + \frac{min_f}{2}]$

TABLE 2 – Les relaxations simplifiées de $x \odot y$ pour chaque mode d'arrondi.(avec $z_r = x \cdot y$).

Ces deux approximations peuvent être réécrites comme suit :

$$\begin{cases} z_r - \frac{2^{-p+1}}{1+2^{-p+1}}z_r - min_f < x \odot y \leq z_r, & z_r \geq 0 \\ z_r + 2^{-p+1}z_r - min_f < x \odot y \leq z_r, & z_r \leq 0 \end{cases}$$

Pour combiner les approximations des cas positifs et négatifs, on utilise la valeur absolue :

$$\begin{cases} z_r - \frac{2^{-p+1}}{1+2^{-p+1}}|z_r| - min_f < x \odot y \leq z_r, & z_r \geq 0 \\ z_r - 2^{-p+1}|z_r| - min_f < x \odot y \leq z_r, & z_r \leq 0 \end{cases}$$

Puisque $max\{\frac{2^{-p+1}}{1+2^{-p+1}}, 2^{-p+1}\} = 2^{-p+1}$, on a

$$z_r - 2^{-p+1}|z_r| - min_f \leq x \odot y \leq z_r$$

□

Le même raisonnement s'applique aux autres cas. Le tableau 2 donne les relaxations simplifiées pour chaque mode d'arrondi. Notez que ces approximations définissent un espace concave.

3 Linéarisation des relaxations

Les relaxations introduites dans les sections précédentes contiennent des termes non-linéaires qui ne peuvent pas être directement manipulés par un solveur MILP. Cette section décrit comment ces termes sont approximés par un ensemble de contraintes linéaires.

3.1 Linéarisation de la valeur absolue

Les relaxations simplifiées précédentes contiennent des valeurs absolues. Elles peuvent être soit grossièrement approximées par trois inégalités ou implémentées grâçe à une décomposition classique plus fine basée sur une réécriture utilisant des "big M" :

$$\begin{cases} z = z_p - z_n \\ |z| = z_p + z_n \\ 0 \leq z_p \leq M \times b \\ 0 \leq z_n \leq M \times (1-b) \end{cases}$$

où b est une variable booléenne, z_p et z_n sont des variables réelles positives, et M est un grand nombre flottant tel que $M \geq max\{|\underline{z}|, |\overline{z}|\}$. La méthode sépare z_p, les valeurs positives de z, de z_n, ses valeurs négatives. Lorsque $b = 1$, z prend ses valeurs positives avec $z = z_p = |z|$. Si $b = 0$, z prend ses valeurs négatives avec $z = -z_n$ et $|z| = z_n$.

Lorsque le solveur MILP permet l'utilisation de contraintes d'indicateur, les deux dernières contraintes peuvent alors être remplacé par :

$$\begin{cases} b = 0 \rightarrow z_p = 0 \\ b = 1 \rightarrow z_n = 0 \end{cases}$$

3.2 Linéarisation des opérations non-linéaires

La linéarisation du produit, du carrée, et de la division sont basées sur les techniques standard proposées par Sahinidis et al [22]. Elles ont aussi été utilisées dans la Quad [14] pour résoudre des contraintes sur les réels. $x \times y$ est linéarisé selon Mc Cormick [16] comme suit :

Supposons que $x \in [\underline{x}, \overline{x}]$ et $y \in [\underline{y}, \overline{y}]$, alors

$$\begin{cases} L1 : z - \underline{x}y - \underline{y}x + \underline{x}\underline{y} \geq 0 \\ L2 : -z + \underline{x}y + \overline{y}x - \underline{x}\overline{y} \geq 0 \\ L3 : -z + \overline{x}y + \underline{y}x - \overline{x}\underline{y} \geq 0 \\ L4 : z - \overline{x}y - \overline{y}x + \overline{x}\overline{y} \geq 0 \end{cases}$$

Ces linéarisations ont été prouvées optimales par Al-Khayyal et Falk [1].

Lorsque $x = y$, i.e., dans le cas où $z = x \otimes x$, la linéarisation peut être améliorée : l'espace convexe de x^2 est sous-estimé par l'ensemble des tangentes à la courbe de x^2 entre $\underline{x}$ et $\overline{x}$, et sur-estimé par la ligne qui joint $(\underline{x}, \underline{x}^2)$ à $(\overline{x}, \overline{x}^2)$. Un bon compromis est obtenu en ne prenant que les deux tangentes correspondant aux bornes de x. La linéarisation de x^2 donne donc :

$$\begin{cases} L1 : z + \underline{x}^2 - 2\underline{x}x \geq 0 \\ L2 : z + \overline{x}^2 - 2\overline{x}x \geq 0 \\ L3 : (\underline{x} + \overline{x})x - z - \underline{x}\overline{x} \geq 0 \\ L4 : z \geq 0 \end{cases}$$

La division peut bénéficier des propriétés de l'arithmétique sur les réels : observons simplement que $z = x/y$ est équivalent à $x = z \times y$. Les linéarisations de Mc Cormick [16] s'appliquent donc aussi à ce cas. Elles nécessitent cependant les bornes de z qui sont calculables à l'aide de l'arithmétique par intervalles :

$$[\underline{z}, \overline{z}] = [\nabla(min(\underline{x}/\underline{y}, \underline{x}/\overline{y}, \overline{x}/\underline{y}, \overline{x}/\overline{y})), \Delta(max(\underline{x}/\underline{y}, \underline{x}/\overline{y}, \overline{x}/\underline{y}, \overline{x}/\overline{y}))]$$

où ∇ et Δ sont respectivement les modes d'arrondis vers $-\infty$ et $+\infty$.

4 Algorithme de filtrage

L'algorithme de filtrage proposé s'appuie sur les linéarisations des relaxations sur les réels du problème initial sur les flottants pour tenter de réduire les domaines des variables au moyen d'un solveur MILP. L'algorithme 1 détaille les étapes de ce processus de filtrage.

Initialement, les contraintes sur les flottants sont relaxées vers des contraintes non-linéaires sur les réels. Puis, les termes non-linéaires de ces relaxations sont linéarisés afin d'obtenir un programme linéaire mixte.

Algorithm 1 FPLP

1: **Function** $FPLP(\mathcal{V}, \mathcal{D}, \mathcal{C}, \epsilon)$
2: % $\mathcal{V}$: Variables flottantes
3: % $\mathcal{D}$: Domaines des variables
4: % $\mathcal{C}$: Contraintes sur les flottants
5: % ϵ : Réduction minimale entre deux itérations
6: $\mathcal{C}' \leftarrow$ **Approximate** $(\mathcal{C})$;
7: $\mathcal{C}'' \leftarrow$ **Linearise** $(\mathcal{C}', \mathcal{D})$;
8: $reduction \leftarrow 0$;
9: **repeat**
10: $\quad \mathcal{D}' \leftarrow$ **FP2B**$(\mathcal{V}, \mathcal{D}, \mathcal{C}, \epsilon)$;
11: $\quad \mathcal{C}'' \leftarrow$ **UpdateLinearisations**$(\mathcal{C}'', \mathcal{D}')$;
12: $\quad oldReduction \leftarrow reduction$;
13: $\quad$ **for all** $x \in \mathcal{V}$ **do**
14: $\quad\quad [\underline{x}_{\mathcal{D}'}, \overline{x}_{\mathcal{D}'}] \leftarrow$ [**safeMin**$(x, \mathcal{C}'')$, $-$**safeMin**$(-x, \mathcal{C}'')$] ;
15: $\quad$ **end for**
16: $\quad reduction \leftarrow \sum_{x \in \mathcal{V}} ((\overline{x}_{\mathcal{D}} - \underline{x}_{\mathcal{D}}) - (\overline{x}_{\mathcal{D}'} - \underline{x}_{\mathcal{D}'}))$;
17: $\quad \mathcal{D} \leftarrow \mathcal{D}'$
18: **until** $reduction \leq \epsilon \times oldReduction$;
19: **return** $\mathcal{D}$;

Le processus de filtrage commence par un appel à FP2B, un filtrage qui s'appuie sur une adaptation de la 2B-consistance aux contraintes sur les flottants, qui fait une première tentative de réduction des bornes des variables. FP2B permet aussi la propagation des

bornes des variables aux variables intermédiaires. Le coût de ce filtrage est relative léger (w est fixé à 10%). La réduction des bornes opérées par FP2B facilite le travail du solveur MILP.

Après cela, le solveur MILP est utilisé afin de calculer la borne supérieure et la borne inférieure du domaine de chacune des variables.

Ce processus est répété jusqu'à ce que le pourcentage de réduction des domaines des variables soit inférieur à un ϵ donné.

4.1 Correction du solveur MILP

L'utilisation d'un solveur efficace MILP tel que CPLEX pour filtrer les domaines des variables soulève deux problèmes importants liés à l'usage de calculs sur les flottants.

Premièrement, les coefficients des linéarisations sont calculés en utilisant l'arithmétique des nombres à virgule flottante et sont donc sujet à des erreurs d'arrondi. Par conséquent, afin d'éviter la perte de solutions, ces calculs doivent reposer sur des directions d'arrondi correctement choisies. Par exemple, considérons la linéarisation de x^2 avec $\overline{x} \geq \underline{x} \geq 0$, on a alors

$$\begin{cases} L1 : y + \Delta(\underline{x}^2) - \Delta(2\underline{x})x \geq 0 \\ L2 : y + \Delta(\overline{x}^2) - \Delta(2\overline{x})x \geq 0 \\ L3 : \Delta(\underline{x} + \overline{x})x - y - \nabla(\underline{x}\overline{x}) \geq 0 \\ L4 : y \geq 0 \end{cases}$$

Ces directions correctes d'arrondi nous assure que l'ensemble des solutions est préservé. Le calcul de coefficients corrects est plus largement détaillé dans [17, 4].

Deuxièmement, les solveurs MILP efficaces utilisent l'arithmétique sur les nombres flottants. Donc, le minimum qu'ils calculent peut être erroné. Le solveur MILP incorrect est rendu correct grâce à la procédure de correction introduite dans [21]. Elle consiste en un calcul d'une borne inférieure correcte du minimum global.

5 Expérimentations

Cette section compare les résultats de différentes techniques de filtrage de contraintes sur les nombres flottants avec la méthode introduite dans cet article. Les expérimentations ont été faites sur un PC portable équipé d'un processeur Intel Duo Core 2.8Ghz, de 4Go de mémoire et dans un environnement Linux.

Nos expérimentations sont basées sur l'ensemble de des programmes suivants :

- `Absorb 1` détecte si x absorbe y lors d'une simple addition ; `Absorb 2` vérifie si y absorbe x.
- `Fluctuat1` est un chemin dans un programme extrait d'un article qui présente l'outil Fluctuat [11]. `Fluctuat2` est un autre chemin du même programme `Fluctuat`.
- `MeanValue` est un programme qui retourne vrai si un intervalle contient une solution en utilisant le théorème des accroissement finis.
- `Cosine` est un programme qui calcule la fonction $cos()$ avec une formule de Taylor.
- `SqrtV1` calcule la racine carrée d'un nombre dans l'intervalle $[0.5, 2.5]$ en utilisant une méthode itérative à deux variables.
- `SqrtV2` calcule la racine carrée en utilisant les séries de Taylor.

La tableau 3 présente les résultats des expérimentations faites avec les méthodes de filtrage suivantes : FP2B, une adaptation de l'algorithme de la 2B-consistance aux contraintes sur les flottants, FP3B, une adaptation de l'algorithme de la 3B-consistance aux contraintes sur les flottants, FPLP(without 2B), une implémentation de l'algorithme 1 sans appel intermédiaire à FP2B, et, FPLP, une implémentation de l'algorithme 1. La première colonne du tableau 3 donne le nom du programme, la deuxième, le nombre de variables du problème initial, et la troisième, le nombre de variables intermédiaires utilisées pour décomposer les contraintes initiales complexes en opérations élémentaires. La quatrième colonne donne le nombre de variables binaires utilisées par FPLP. Pour chaque algorithme de filtrage, le tableau 3 donne le temps nécessaire au filtrage des contraintes (colonnes $t(ms)$). Pour tout les algorithmes de filtrage – à l'exception de FP2B – le tableau 3 donne le pourcentage de réduction obtenu par rapport à celle obtenue en utilisant la FP2B (colonne %(FP2B)). Notez qu'une limite de 2 minutes pour terminer le filtrage est imposée dans ces expérimentations.

Les résultat obtenus dans le tableau 3 montrent que FPLP effectue de meilleures réductions des domaines en moins de temps que les méthodes de filtrage locales. Ce phénomène est particulièrement accentué sur les exemples `Absorb1` et `SqrtV1`. Dans ces deux exemples, FP2B souffre du problème des occurrences multiples de variables. FPLP présente de manière consistante de meilleurs performances que FP3B : il fourni presque toujours des domaines plus petits et nécessite toujours moins de temps.

Une comparaison entre FPLP sans et avec appel à FP2B montre l'intérêt d'une coopération entre ces deux méthodes de filtrage qui peut réduire considérablement le temps de calcul sans modifier les capacités de filtrage.

				FP2B	FP3B		FPLP (without 2B)		FPLP	
Programme	n	n_T	n_B	$t(ms)$	$t(ms)$	%(FP2B)	$t(ms)$	%(FP2B)	$t(ms)$	%(FP2B)
`Absorb1`	2	1	1	TO	TO	-	3	98.91	5	98.91
`Absorb2`	2	1	1	1	24	0.00	3	100.00	4	100.00
`Fluctuat1`	3	12	2	4	156	99.00	264	99.00	172	99.00
`Fluctuat2`	3	10	2	1	4	0.00	29	0.00	21	0.00
`MeanValue`	4	28	6	3	82	97.45	530	97.46	78	97.46
`Cosine`	5	33	7	5	153	33.60	104	33.61	43	33.61
`SqrtV1`	11	140	29	9	27198	99.63	1924	100.00	1187	100.00
`SqrtV2`	21	80	17	7	TO	-	6081	100.00	1321	100.00

TABLE 3 – Expérimentations

6 Conclusion

Dans cet article, nous avons introduit un nouvel algorithme de filtrage pour les contraintes sur les nombres flottants. Grâce aux linéarisations des relaxations non-linéaires sur les réels des contraintes initiales sur les flottants, cet algorithme peut réduire les domaines des variables avec un solveur MILP. Les expérimentations montrent que FPLP améliore significativement le processus de filtrage, en particulier, lorsqu'il est associé à un algorithme de filtrage de type FP2B. La programme linéaire mixte bénéficie d'une vue plus globale du problème que les filtrage locaux et peut ainsi apporter une solution au problème des occurrences multiples de variables.

Plus d'expérimentations sont cependant nécessaire afin d'améliorer notre compréhension des interactions entre les deux algorithmes ainsi que leur performances. Par exemple, il n'est pas encore certain qu'un appel systématique à FPLP à chaque nœud de l'arbre de recherche soit requis.

Références

[1] F.A. Al-Khayyal and J.E. Falk. Jointly constrained biconvex programming. *Mathematics of Operations Research*, pages 8 :2 :273–286, 1983.

[2] Alessandro Armando, Jacopo Mantovani, and Lorenzo Platania. Bounded model checking of software using smt solvers instead of sat solvers. *Int. J. Softw. Tools Technol. Transf.*, 11 :69–83, January 2009.

[3] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, TACAS '99, pages 193–207, London, UK, 1999. Springer-Verlag.

[4] Glencora Borradaile and Pascal Van Hentenryck. Safe and tight linear estimators for global optimization. *Mathematical Programming*, 2005.

[5] Bernard Botella, Arnaud Gotlieb, and Claude Michel. Symbolic execution of floating-point computations. *Softw. Test., Verif. Reliab.*, 16(2) :97–121, 2006.

[6] Angelo Brillout, Daniel Kroening, and Thomas Wahl. Mixed abstractions for floating-point arithmetic. In *Proceedings of FMCAD 2009*, pages 69–76. IEEE, 2009.

[7] Hélène Collavizza, Michel Rueher, and Pascal Hentenryck. CPBPV : a constraint-programming framework for bounded program verification. *Constraints*, 15(2) :238–264, 2010.

[8] Lucas Cordeiro, Bernd Fischer, and Joao Marques-Silva. Smt-based bounded model checking for embedded ansi-c software. *IEEE Transactions on Software Engineering*, May 2011.

[9] Patrick Cousot, Radhia Cousot, Jerome Feret, Antoine Mine, Laurent Mauborgne, David Monniaux, and Xavier Rival. Varieties of static analyzers : A comparison with astree. In *TASE '07*, pages 3–20. IEEE, 2007.

[10] Malay K Ganai and Aarti Gupta. Accelerating high-level bounded model checking. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, ICCAD '06, pages 794–801, New York, NY, USA, 2006. ACM.

[11] K. Ghorbal, E. Goubault, and S. Putot. A logical product approach to zonotope intersection. In *Computer Aided Verification*, pages 212–226. Springer, 2010.

[12] Arnaud Gotlieb, Bernard Botella, and Michel Rueher. Automatic test data generation using constraint solving techniques. In *ISSTA*, pages 53–62, 1998.

[13] Arnaud Gotlieb, Bernard Botella, and Michel Rueher. A clp framework for computing structural test data. In John W. Lloyd, Verónica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Luís Moniz Pereira, Yehoshua Sagiv, and Peter J. Stuckey, editors, *Computational Logic*, volume 1861 of *Lecture Notes in Computer Science*, pages 399–413. Springer, 2000.

[14] Yahia Lebbah, Claude Michel, Michel Rueher, David Daney, and Jean-Pierre Merlet. Efficient and safe global constraints for handling numerical constraint systems. *SIAM J. Numer. Anal*, 42 :2076–2097, 2005.

[15] Olivier Lhomme. Consistency techniques for numeric csps. In *IJCAI*, pages 232–238, 1993.

[16] G.P. McCormick. Computability of global solutions to factorable nonconvex programs – part i – convex underestimating problems. *Mathematical Programming*, 10 :147–175, 1976.

[17] C. Michel, Y. Lebbah, and M. Rueher. Safe embedding of the simplex algorithm in a CSP framework. In *Proc. of CPAIOR 2003, CRT, Université de Montréal*, pages 210–220, 2003.

[18] Claude Michel. Exact projection functions for floating point number constraints (pdf). In *AMAI*, 2002.

[19] Claude Michel, Michel Rueher, and Yahia Lebbah. Solving constraints over floating-point numbers. In Toby Walsh, editor, *CP*, volume 2239 of *Lecture Notes in Computer Science*, pages 524–538. Springer, 2001.

[20] Antoine Miné. *Weakly Relational Numerical Abstract Domains*. PhD thesis, École Polytechnique, Palaiseau, France, December 2004.

[21] A. Neumaier and O. Shcherbina. Safe bounds in linear and mixed-integer programming. *Math. Programming A.*, page 99 :283 296, 2004.

[22] Hong S. Ryoo and Nikolaos V. Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, pages 107–138, 1996.