Résolution de contraintes sur les nombres à virgule flottante par une approximation sur les nombres réels*

Mohammed Said BELAID, Claude MICHEL, Michel RUEHER

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

Résumé

La mise en œuvre effective de méthodes de vérification de programmes comportant des calculs sur les nombres à virgule flottante reste encore problématique. C'est pourquoi nous présentons dans cet article une nouvelle méthode de résolution de contraintes sur les nombres à virgule flottante qui consiste à les approximer sur les réels. Elle est basée sur la construction d'approximations sur les réels, précises et conservatives des solutions des contraintes sur les nombres à virgule flottante. Cette méthode permet de s'appuyer sur l'utilisation d'algorithmes de filtrage sur les réels pour résoudre des problèmes sur les nombres à virgule flottante. Il devient ainsi possible de repousser les limitations actuelles des solveurs de contraintes sur les nombres à virgules flottantes, telles que le problème du passage à l'échelle, pour générer des jeux de tests, ou vérifier des programmes plus conséquents que ceux traités jusqu'à maintenant.

Keywords : contraintes sur les nombres à virgule flottante, approximation des solutions, vérification de programme.

1 Introduction

La vérification et le test de programmes sont des étapes très importantes dans un processus de développement de logiciels, plus particulièrement, lorsqu'il s'agit de logiciels critiques [6]. Par exemple, dans l'aéronautique beaucoup de ces logiciels effectuent des calculs basés sur l'arithmétique des nombres à virgule flottante. Or l'utilisation sans précaution de cette arithmétique peut causer de graves dégâts. Citons, par exemple, le crash de la fusée Ariane V lié à un problème de conversion d'un nombre à virgule flottante vers un entier 16 bits [11]. Un autre exemple qui montre bien que, même une erreur mineure peut avoir de graves conséquences, est l'échec de la mission de l'anti-missile Patriot [17]. Une imprécision dans le calcul du temps (de l'ordre de 10^{-7}) n'a pas permis un calcul assez précis de la position, le cumul de l'erreur étant d'autant plus grand que ce programme était lancé bien avant le départ de l'anti-missile. Ces événements montrent bien que la vérification et le test sont encore plus cruciaux pour les programmes avec des nombres à virgule flottante.

Plusieurs approches de vérification de programmes avec des calculs sur les nombres flottants ont été développées. On peut notamment citer les approches basées sur l'interprétation abstraite [14, 9] qui, si elles permettent de garantir pour certains programmes que les opérations arithmétiques sont correctes, rejettent, malheureusement, de nombreux programmes corrects. D'autres approches de la vérification de programmes avec des nombres à virgule flottante se basent sur la programmation par contraintes. Les contraintes ont été utilisées pour le test de programmes [2], ou encore, la vérification de la conformité des programmes vis-à-vis leurs spécifications. Les outils conçus pour ces approches se limitent souvent au traitement de l'arithmétique des entiers comme, par exemple, Euclide [8] pour le test de programmes et CPBPV [4] pour la vérification de conformité d'un programme vis à vis de sa spécification. Il existe cependant peu d'exemples d'outils de vérification de programmes avec des nombres à virgule flottante basés sur les contraintes, en particulier, à cause de l'absence d'un solveur efficace capable de traiter des contraintes sur les nombres à virgule flottante. En effet, les méthodes disponibles de résolu-

*Ce travail a été partiellement financé par l'ANR, programme SESUR, projet CAVERN (ANR-07-SESUR-003).

tion de contraintes sur les nombres à virgule flottante [13, 12, 3], qui s'appuient sur une adaptation aux flottants d'algorithmes de filtrage sur les réels, peinent à passer à l'échelle. Cela est en partie dû aux difficultés inhérentes à l'arithmétique des nombres à virgule flottante dont la pauvreté des propriétés rend souvent impossible la transposition de résultats établis sur les réels.

Afin d'améliorer le potentiel des solveurs de contraintes sur les nombres à virgule flottante, nous proposons dans cet article de construire une approximation sur les réels, précise et conservative, des solutions des contraintes sur les nombres à virgule flottantes. Une telle approximation pourra bénéficier de l'ensemble des outils développés pour résoudre des problèmes sur les réels et, ainsi, offrir des perspectives jusqu'alors interdites par les limitations de l'arithmétique des nombres à virgule flottante.

L'article est organisé comme suit : la section suivante rappelle quelques notations et notions de base utiles à la compréhension de cet article. La section 3 introduit notre méthode à l'aide d'un exemple illustratif. Les approximations sur les réels sont détaillées dans la section 4 alors que la section 5 décrit leur mise en œuvre. Enfin, la section 6 compare notre méthode aux travaux connexes.

2 Notations et définitions

2.1 Les nombres à virgule flottante

Les nombres à virgule flottante -souvent appelés nombres flottants, ou encore flottants- sont une représentation d'un sous-ensemble fini des nombres réels. On note par $\mathbb{F}$ l'ensemble des nombres flottants et $\mathbb{R}$ l'ensemble des nombres réels.

Les nombres flottants sont utilisés dans les programmes informatiques pour approximer des valeurs de type réel. Ils se composent de trois parties : le signe s (0 ou 1), la mantisse m et l'exposant e [7]. Un flottant est représenté par

$$(-1)^s m \times \beta^e$$

où β est la base de calcul (2 dans le cas général).

La mantisse m est représentée par un nombre fixe de chiffres.

$$d_0.d_1 d_2 \cdots d_{p-1} \text{ avec } (0 \leq d_i < \beta)$$

où p est le nombre de chiffres qui composent la mantisse. Notons qu'il existe une relation directe entre p et la précision du calcul. Afin de garantir une représentation unique d'un flottant, la mantisse est normalisée, i.e. $d_0 \neq 0$.

L'exposant est un entier signé délimité par $e_{min} \leq e \leq e_{max}$.

2.2 Arrondi et erreurs d'arrondi

En utilisant les opérations de l'artihmétique sur les réels, le résultat d'une opération sur les flottants n'est, le plus souvent, pas un nombre flottant. Aussi, afin de n'avoir que des nombres flottants à manipuler, ce résultat doit être arrondi vers le flottant le plus proche. L'opération d'arrondi introduit donc une erreur qui peut être mesurée de différentes manières.

L'erreur absolue

L'erreur absolue err_{abs} est la différence entre la valeur réelle et son approximation sur les flottants.

$$err_{abs} = |valeur_{relle} - valeur_{flottante}|$$

L'erreur relative

L'erreur relative ($\epsilon = err_{rel}$) est le rapport entre l'erreur absolue et la valeur réelle.

$$\epsilon = err_{rel} = \frac{err_{abs}}{|valeur_{relle}|}$$

L'erreur en ulp

L'*Ulp* (*Unit in the Last Place*) est la distance qui sépare deux flottants consécutifs [16] et donc, l'erreur maximale qui peut être faite lors d'un arrondi. Plus formellement, la valeur de l'*ulp* pour un nombre flottant x est

$$ulp(x) = \beta^{e_x} \times \beta^{-p+1}$$

où e_x est l'exposant de x, β sa base de calcul (2 en général) et p désigne le nombre de chiffres de la mantisse.

2.3 La norme IEEE-754

En 1985, l'IEEE normalise l'arithmétique des nombres à virgule flottante [10]. La plupart des processeurs récents utilisent cette norme. En 2008, la norme a été révisée [1] afin de, entre autres, lever certaines ambiguïtés, notamment dans les choix d'implémentation.

La norme IEEE-754 définit les nombres à virgule flottante en base 2. Elle définit deux formats principaux pour représenter les nombres à virgule flottante : le format simple précision sur 32 bits (dont 24 pour la mantisse, et 8 pour l'exposant) et le format double précision sur 64 bits (dont 53 pour la mantisse, et 11 pour l'exposant). Il existe également d'autres formats moins spécifiés comme le double étendu ainsi que le quadruple précision introduit lors de la révision de la norme.

Les nombres normalisés et dénormalisés

La norme IEEE-754 distingue les nombres normalisés des nombres dénormalisés. Les nombres normalisés ont une mantisse où la partie entière est égale à 1, i.e. $x = m_x 2^{e_x}$ est normalisé si $m_x = 1.x_1 \cdots x_{p-1}$ et $e_{max} < e_x < e_{min}$. Les nombres dénormalisés servent à représenter les nombres très proches du zéro. Un nombre $x = m_x 2^{e_x}$ est dénormalisé si la partie entière de la mantisse est nulle, i.e. $m_x = 0.x_1 \cdots x_{p-1}$, et l'exposant est le plus petit exposant possible, i.e. $e_x = e_{min}$.

La norme introduit également des valeurs symbolique telles que les infinis ($-\infty$, $+\infty$) et les NaN (*Not a Number*). Les infinis représentent des dépassements de capacité de calcul alors que les NaN représentent le résultat d'une opération impossible comme une division par zéro, ou encore, la racine carrée d'un nombre négatif.

Les modes d'arrondi

La norme IEEE-754 fournit quatre modes d'arrondi : l'arrondi vers 0, vers $-\infty$, vers $+\infty$ et au plus proche. Avec le mode d'arrondi vers $-\infty$, le réel x est arrondi vers le plus grand flottant inférieur à x, alors qu'avec le mode d'arrondi à $+\infty$, il est arrondi vers le plus petit flottant supérieur à x. Le mode d'arrondi vers 0 se comporte comme un arrondi vers $-\infty$ si $x >= 0$, et comme un arrondi vers $+\infty$ dans les autres cas. Pour le mode d'arrondi au plus proche, le nombre réel est arrondi au flottant le plus proche de x. Lorsque x est à équidistance de deux flottants, celui avec une mantisse paire est choisi. La version révisée de la norme [1] définit une variante du mode d'arrondi au plus près appelée *Round to nearest away* qui, dans le cas où x est à équidistance de deux flottants, choisit celui dont la valeur absolue est la plus grande.

Les opérations

La norme définit cinq opérations de base : les 4 opérations arithmétiques ($+$, $-$, $\times$, $/$), et la racine carrée. La norme impose à ces opérations d'être correctement arrondies, i.e. le résultat de ces opérations sur les flottants doit être égal à l'arrondi du résultat obtenu en effectuant l'opération équivalente sur les réels. Par exemple, pour l'addition, si r est la fonction d'arrondi, $+$ l'addition sur les réels et $\oplus$, l'addition sur les flottants, alors

$$x \oplus y = r(x + y)$$

2.4 Notations

Dans la suite de cet article, $\oplus$, $\ominus$, $\otimes$ et $\oslash$ dénotent respectivement une addition, une soustraction, une multiplication ou une division sur les nombres flottants. De la même manière, $+$, $-$, $\times$ et / dénotent les opérations équivalentes sur les réels. Pour un nombre flottant x, x^+ et x^- dénotent respectivement, le successeur et le prédécesseur de x.

3 La méthode proposée

Le principe de base de notre méthode consiste à construire des approximations sur les réels des contraintes sur les nombres flottants. Cette transformation doit conserver le maximum d'information des contraintes originales, et ne doit éliminer aucune solution du système de contraintes initiales. L'intuition de ce principe est illustrée par l'exemple suivant

$$x \oplus 16.0 == 16.0$$

Interprétée sur $\mathbb{R}$, cette contrainte a une solution unique $x = 0$. Par contre, elle en a plusieurs dans l'ensemble des nombres à virgule flottante. Supposons que l'opération est faite avec un mode d'arrondi vers le plus proche. $x \oplus 16$ est égal à 16 si et seulement si le résultat réel de l'opération $x + 16$ est compris entre $16 - \frac{1}{2}ulp(16)$ et $16 + \frac{1}{2}ulp(16)$, i.e.

$$16 - \frac{1}{2}ulp(16) \leq x + 16 \leq 16 + \frac{1}{2}ulp(16)$$

qui est une contrainte sur les réels. Pour cette contrainte, l'ensemble des flottants $x \in [-\frac{1}{2}ulp(16), \frac{1}{2}ulp(16)]$, sont des solutions de la contrainte initiale sur $\mathbb{F}$. Cependant, dans le cas général, il n'est pas possible d'obtenir une approximation aussi précise des solutions.

Pour illustrer ceci, considérons un type particulier de nombres flottants binaires représentés sur 4 bits, dont deux bits pour la mantisse ($p = 2$) et deux pour l'exposant. Ces flottants ne permettent que de représenter des nombres positifs car, pour des raisons de simplicité, le bit de signe est absent. Les valeurs réelles des nombres représentables dans ce format sont

$$\{0, 0.25, 0.5, 0.75, 1.0, 1.5, 2.0, 3.0, 4.0, 6.0\}$$

Par exemple, 0.75 est représenté par $1.1_{(2)} \times 2^{-1}$ et 4.0 par $1.0_{(2)} \times 2^2$. Le mode d'arrondi considéré étant l'arrondi vers $-\infty$, on a, par exemple, $1 \oplus 0.75 = r(1.75) = 1.5$.

Considérons le système de contraintes sur les

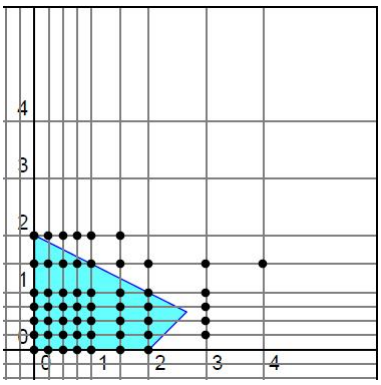

FIG. 1 – Solutions réelles vs solutions flottantes. *L'espace en bleu représente les solutions des contraintes sur les réels. Les points en noir représentent les solutions sur les flottants.*

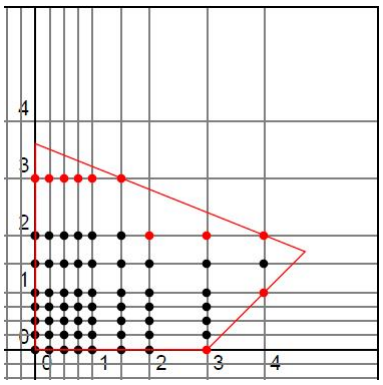

FIG. 2 – Premier niveau d'approximation. *Les points en noirs représentent les solutions des contraintes sur les flottants. Les points en rouge représentent les éléments non-solution des contraintes sur les flottants et qui appartiennent à l'approximation.*

nombres flottants suivant

$$\mathcal{CSP}_{\mathbb{F}} = \begin{cases} x & \geq 0 \\ y & \geq 0 \\ (x \oplus y) \oplus y & \leq 4 \\ x \ominus y & \leq 2 \end{cases}$$

où $x \in \mathbb{F}$ et $y \in \mathbb{F}$. Les solutions flottantes de ces contraintes sont représentées par les points de couleur noire dans la figure 1. À partir de ce système de contraintes, il est possible de construire un système sur les réels en interprétant directement chaque opération comme étant une opération sur les réels

$$\mathcal{CSP}_{\mathbb{R}} = \begin{cases} x & \geq 0 \\ y & \geq 0 \\ x + 2 \times y & \leq 4 \\ x - y & \leq 2 \end{cases}$$

où $x \in \mathbb{R}$ et $y \in \mathbb{R}$. La zone bleu de la figure 1 représente les solutions réelles de $\mathcal{CSP}_{\mathbb{R}}$. Notez que toutes les solutions flottantes de $\mathcal{CSP}_{\mathbb{F}}$ ne sont pas incluses dans l'espace des solutions réelles de $\mathcal{CSP}_{\mathbb{R}}$. Par exemple, le point $(3, 0.75)$ satisfait les contraintes sur les flottants, puisque $(3 \oplus 0.75) \oplus 0.75 = 3 \leq 4$ et $(3 \ominus .75) = 2 \leq 2$, alors qu'il ne satisfait pas les contraintes sur les réels, puisque $(3 + 0.75) + 0.75 = 4.5 > 4$ et $(3 - .75) = 2.25 > 2$. Ceci met en évidence la nécessité d'utiliser une approche spécifique pour résoudre correctement les contraintes sur les flottants.

Bien qu'une transposition directe des opérations sur les flottants en opérations sur les réels ne soit pas conservative des solutions sur les flottants, il est possible d'utiliser de construire des approximations correctes des contraintes sur les flottants, c'est à dire, qui conservent toutes les solutions. La figure 2 présente la première approximation conservative que nous avons défini. Cette approximation englobe non seulement toutes les solutions des contraintes sur les flottants mais contient aussi des éléments non-solutions représentés ici par des points rouges. Il est toutefois possible de définir une meilleure approximation, i.e. qui contient moins de points non-solutions sur les flottants. Une telle approximation est présentée dans la figure 3.

Notez qu'avec cette seconde approximation, le nombre de points non-solutions est réduit par huit. Si le nombre d'éléments éliminés semble faible, il peut être plus significatif pour des systèmes de taille importante.

De telles approximation peuvent directement être traitées à l'aide d'algorithmes de filtrage sur les réels et permettre ainsi une réduction notable des domaines des variables. En combinant ce processus avec un solveur de contraintes sur les flottants, il devient possible d'obtenir plus rapidement des solutions correctes du système de contraintes sur les flottants.

En se basant sur ce principe, nous avons proposé une nouvelle méthode dont les étapes principales sont illustrées dans la figure 4. Les contraintes sur les flottants sont d'abord approximées par des contraintes sur les réels. Les détails de cette transformation sont donnés dans la section 4. Les contraintes obtenues sont ensuite traitées par des algorithmes de filtrage issus d'un solveur sur les réels. S'il n'existe aucune solution pour le système de contraintes sur les réels, alors les contraintes initiales sur les nombres à virgule flottante n'ont pas de solutions. Dans le cas contraire, une recherche combinée avec un processus valide d'énumération sur les flottants. Ce dernier point n'est toutefois pas abordé dans cet article.

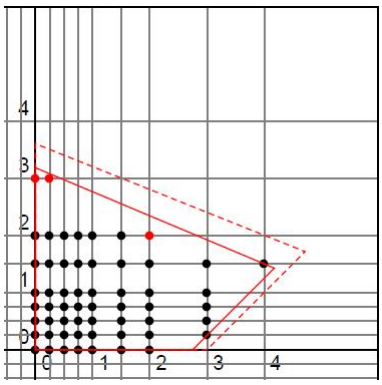

FIG. 3 – Approximation plus précise *Les points noirs représentent les solutions des contraintes sur les flottants. Les points en rouge représentent les éléments non-solution des contraintes sur les flottants et qui appartiennent à l'approximation.*

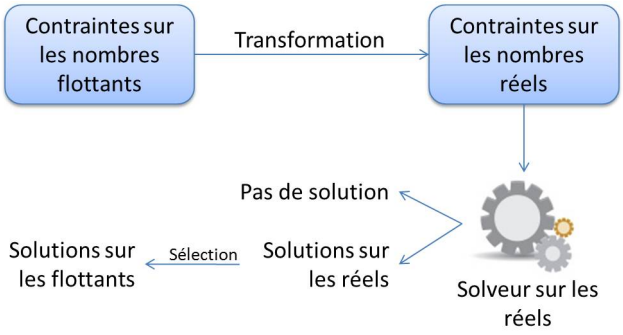

FIG. 4 – Les étapes de la méthode proposée. *La première étape fait une transformation de contraintes entre les réels et les flottants. La deuxième consiste à résoudre les contraintes obtenus. Puis Les solutions flottantes sont choisies*

4 Transformation des contraintes sur les nombres à virgule flottante vers des contraintes sur les réels

Cette étape à pour but de construire un système de contraintes sur les réels qui approxime correctement les contraintes sur les nombres à virgule flottante. Le système construit doit avoir les qualités suivantes :

- Il doit être conservatif en terme de solutions, i.e. aucune solution des contraintes sur les flottants ne doit être perdue.
- La transformation doit être la plus précise possible, i.e. elle doit minimiser le nombre de points qui ne satisfont pas les contraintes sur les nombres flottants et qui sont des solutions pour les contraintes sur les réels.

4.1 Méthode de transformation

La méthode appliquée ici consiste à construire des approximations précises et conservatives des contraintes élémentaires sur les flottants. Cependant, et dans le cas général, les contraintes sur les flottants sont composées d'expressions complexes qu'il s'agit aussi de prendre en compte. Pour cela, notre approche s'appuie sur les propriétés de l'arithmétique des intervalles. Les intervalles vont nous permettre de combiner les approximations obtenues pour les opérations élémentaires afin de traiter des expressions plus complexes. Plus formellement, considérons une contrainte C_i.

$$C_i : f(x_1, \cdots, x_n) \diamond g(x_1, \cdots, x_n)$$

où $\diamond \in \{<, \leq, \geq, >, =\}$, et f et g sont des fonctions sur les nombres flottants avec les opérations de base sur les flottants. Chaque membre de la relation doit être approximé par un intervalle sur les réels

$f(x_1, \cdots, x_n) \in [f_{inf}(x_1, \cdots, x_n), f_{sup}(x_1, \cdots, x_n)]$
$g(x_1, \cdots, x_n) \in [g_{inf}(x_1, \cdots, x_n), g_{sup}(x_1, \cdots, x_n)]$

où f_{inf}, f_{sup}, g_{inf} et g_{sup} sont des fonctions sur les réels. Pour obtenir un système conservatif des solutions, les intervalles doivent contenir toutes les solutions possibles de la fonction sur les flottants. Des approximations initiales sont d'abord définies pour chaque opération de base (cf 4.2) pour être ensuite combinées entre elles grâce à l'arithmétique par intervalle. Le système résultant offre ainsi une approximation générale du problème initial.

Après avoir traité chaque expression constitutive d'une contrainte sur les flottants, les contraintes sur les réels correspondantes peuvent être générées. Pour ce faire, à chaque expression flottante traitée est substituée une variable. Le système résultant prend alors la forme suivante

$$f_{inf}(x_1, \cdots, x_n) \leq x_f \leq f_{sup}(x_1, \cdots, x_n)$$
$$g_{inf}(x_1, \cdots, x_n) \leq x_g \leq g_{sup}(x_1, \cdots, x_n)$$
$$x_f \diamond x_g$$

4.2 Différentes approximations

Nous présentons ici une méthode pour approximer sur les réels les opérations de base sur les flottants. Sans perte de généralité, nous nous limiterons à l'addition $\oplus$ avec un mode d'arrondi vers $-\infty$. Le même raisonnement peut être suivi pour les autres opérations et les autres modes d'arrondi. Nous nous limitons également aux nombres flottants positifs normalisés.

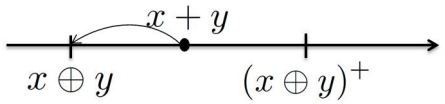

FIG. 5 – La valeur réelle est comprise entre la valeur flottante de l'arrondi et sa valeur suivante.

Approximation par l'erreur en *ulp*

Cette première approximation s'appuie sur le fait que les opérations sont correctement arrondies, i.e. le résultat flottant doit être égal à l'arrondi du résultat réel.

Pour que $(x+y)$ soit arrondi à $(x \oplus y)$ pour un mode d'arrondi vers $-\infty$, il doit être compris entre $(x \oplus y)$ et son successeur, i.e. $(x \oplus y)^+$ (voir figure figure 5). On a donc

$$x \oplus y \leq x + y < (x \oplus y)^+$$

avec $(x \oplus y)^+ = (x \oplus y) + ulp(x \oplus y)$ De cette relation on peut tirer une approximation sur $\mathbb{R}$ pour $(x \oplus y)$

$$(x + y) - ulp(x \oplus y) < x \oplus y \leq x + y$$

Cependant, dans cette relation, la valeur d'$ulp(x \oplus y)$ dépend des valeurs de x et de y. S'il est possible de trouver une borne supérieure de cet Ulp, une approximation plus précise peut cependant être construite. Elle résulte de la propriété suivante

Proposition 1. *Soient x et y des nombres flottants positifs normalisés, et pour un mode d'arrondi fixé à $-\infty$, alors $x \oplus y$ est borné par*

$$\alpha \times (x + y) < x \oplus y \leq x + y$$

avec $\alpha = \dfrac{1}{1 + 2^{-p+1}}$ et où p est le nombre de chiffres de la mantisse.

Démonstration. Pour obtenir cette relation, nous devons d'abord calculer la borne supérieure et la borne inférieure de l'erreur relative ϵ

$$\epsilon = \left| \frac{valeur_{relle} - valeur_{flottante}}{valeur_{relle}} \right|$$

$$\epsilon = \left| \frac{(x + y) - (x \oplus y)}{(x + y)} \right|$$

Sachant que, pour un mode d'arrondi fixé à $-\infty$, la valeur réelle de $(x+y)$ est toujours supérieure au résultat flottant de $(x \oplus y)$, et que le résultat réel est toujours positif puisque x et y sont positifs, ϵ est toujours positif. Notons aussi que le cas $x + y = 0$ correspond à $x \oplus y = 0$, i.e. à une erreur absolue nulle et donc une erreur relative que l'on peut considérer comme nulle. Par la suite, on suppose que $x + y > 0$, x et y étant des nombres flottants positifs normalisés. On a donc

$$\epsilon = \frac{(x + y) - (x \oplus y)}{(x + y)} \geq 0$$

$$= 1 - \frac{x \oplus y}{x + y}$$

Il nous faut maintenant trouver une borne supérieure pour ϵ.

L'addition étant correctement arrondie, on a

$$x \oplus y \leq x + y < x \oplus y + ulp(x \oplus y)$$

donc

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

En multipliant chaque membre de la relation par $(x \oplus y)$, on obtient

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

Ce qui, multiplié par -1 et en ajoutant 1 donne

$$1 - \frac{x \oplus y}{x + y} < 1 - \frac{x \oplus y}{x \oplus y + ulp(x \oplus y)}$$

donc

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

Posons $z = x \oplus y = m_z 2^{e_z}$ où m_z est la mantisse de z (elle est normalisée puisque les deux opérandes sont normalisées) et e_z l'exposant de z. On a alors

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

$$0 \leq \epsilon < \frac{ulp(z)}{z + ulp(z)}$$

La valeur de l'$ulp(z)$ est donnée par $ulp(z) = 2^{-p+1}2^{e_z}$. En remplaçant $ulp(z)$ par cette formule, on obtient

$$0 \leq \epsilon < \frac{ulp(z)}{z + ulp(z)}$$

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

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

Donc il suffit de trouver une borne supérieure pour $2^{-p+1}/(m_z + 2^{-p+1})$ qui est maximal lorsque $m_z \in [1.0, 2.0[$ est minimal i.e. $m_z = 1.0$.

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

donc

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

Il ne reste qu'à trouver une approximation pour $(x \oplus y)$. On a

$$\begin{aligned} 0 \leq \quad & \epsilon & < \frac{2^{-p+1}}{1 + 2^{-p+1}} \\ 0 \leq \quad & \frac{(x+y) - (x \oplus y)}{(x+y)} & < \frac{2^{-p+1}}{1 + 2^{-p+1}} \\ 0 \leq \quad & (x+y) - (x \oplus y) & < (x+y)\frac{2^{-p+1}}{1 + 2^{-p+1}} \end{aligned}$$

On obtient une approximation pour l'opération d'addition

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

Donc la valeur de $(x \oplus y)$ peut être approximée par l'intervalle $]\alpha(x+y), x+y]$, avec $\alpha = 1/(1 + 2^{-p+1})$. □

Avec le même raisonnement, un résultat similaire est obtenu pour les autres opérations, toujours avec un mode d'arrondi vers $-\infty$

$$\boxed{\begin{array}{rcl} (x \oplus y) & \in &]\alpha(x+y), x+y] \\ (x \ominus y) & \in &]\alpha(x-y), x-y] \\ (x \otimes y) & \in &]\alpha(x \times y), x \times y] \\ (x \oslash y) & \in &]\alpha(x/y), x/y] \\ \alpha & = & \frac{1}{1+2^{-p+1}} \end{array}}$$

Approximation plus précise

L'approximation précédente peut encore être affinée. En effet, le résultat précédent est basé sur la relation suivante

$$x \oplus y \leq x + y < x \oplus y + ulp(x \oplus y)$$

Or, $(x+y)$ n'atteindra jamais $x \oplus y + ulp(x \oplus y)$. En recherchant le pire cas qui peut réellement être atteint, il est possible de construire une approximation avec une inégalité large.

Comme dans la méthode précédente, nous recherchons une délimitation de l'erreur relative, mais, cette fois, dont les bornes soient atteignable par ϵ. La propriété suivante donne ces bornes

Proposition 2. *Si x et y sont des nombres flottants positifs normalisés, et si le mode d'arrondi est fixé à $-\infty$, alors l'erreur relative de l'opération $x \oplus y$ est bornée par*

$$0 \leq \epsilon \leq \frac{y}{x+y}$$

De plus, l'erreur relative maximal correspond à $(x \oplus y = x)$.

Démonstration. Pour un x donné, nous avons $x \leq x \oplus y$ puisque y est positif (l'addition sur les flottants étant commutative, x et y sont ici interchangeables). Par ailleurs, le résultat réel est toujours supérieur à $x \oplus y$ pour un mode d'arrondi vers $-\infty$. On a donc

$$x \leq x \oplus y \leq x + y$$

À partir de cette relation, on obtient

$$\begin{aligned} x \leq \quad & x \oplus y & \leq x + y \\ \frac{x}{x+y} \leq \quad & \frac{x \oplus y}{x+y} & \leq 1 \\ 0 \leq \quad & 1 - \frac{x \oplus y}{x+y} & \leq \frac{y}{x+y} \end{aligned}$$

Donc l'erreur relative est bornée par

$$0 \leq \epsilon \leq \frac{y}{x+y}$$

Remarquons que $y/(x+y)$ est atteignable par ϵ. En effet, lorsque $(x \oplus y = x)$, i.e. lorsque y est absorbé par x lors de l'opération d'arrondi, l'erreur relative est alors égale à

$$\epsilon = \frac{(x+y) - (x \oplus y)}{(x+y)} = \frac{y}{x+y}$$

□

Cet encadrement plus fin de l'erreur relative nous permet de construire une approximation plus précise de l'addition sur les flottants donnée par la propriété suivante

Proposition 3. *Si x et y sont des nombres flottants positifs normalisés, et si le mode d'arrondi est fixé à $-\infty$, alors $x \oplus y$ est délimité par*

$$\gamma \times (x+y) \leq x \oplus y \leq x + y$$

avec $\gamma = \dfrac{1}{1 + (2^{-p+1} - 2^{-2p+1})}$.

Démonstration. À la fin de la démonstration de la propriété précédente, nous avons établi que l'erreur relative atteint sa borne supérieure pour

$$x \oplus y = x$$

Nous savons aussi que, pour que l'addition soit correctement arrondie, il faut que

$$x \oplus y \leq \quad x + y \quad < (x \oplus y) + ulp(x \oplus y)$$

En combinant ces deux résultats, nous obtenons

$$\begin{array}{rcl} x \leq & x + y & < x + ulp(x) \\ 0 \leq & y & < ulp(x) \end{array}$$

Donc, pour que $x \oplus y = x$, on a $y \in [0, ulp(x)[$.

Revenons à l'erreur relative ϵ

$$0 \leq \epsilon \leq \frac{y}{x+y}$$

Nous devons trouver une limite supérieure pour la fonction $y/(x+y)$. La fonction $y/(x+y)$ est croissante par rapport à y. Donc elle est maximale lorsque y est maximal. Sachant que $y \in [0, ulp(x)[$ et $y \in \mathbb{F}$, la plus grande valeur flottante de y est le prédécesseur de $ulp(x)$, i.e. $(ulp(x))^-$. Posons $x = m_x 2^{e_x}$, l'$ulp(x)$ est donc

$$ulp(x) = 2^{e_x} 2^{-p+1} = 1.0 \times 2^{e_x - p + 1}$$

et son prédécesseur a pour valeur

$$\begin{array}{rcl} (ulp(x))^- & = & (1.0 \times 2^{e_x - p + 1})^- \\ & = & 1.11 \cdots 11 \times 2^{e_x - p} \\ & = & (2 - 2^{-p+1}) \times 2^{e_x - p} \\ & = & 2^{e_x - p + 1} - 2^{e_x - 2p + 1} \\ & = & 2^{e_x}(2^{-p+1} - 2^{-2p+1}) \end{array}$$

Remplaçons dans la propriété 2 y par la valeur obtenue. On a

$$\begin{array}{rcl} 0 \leq \quad \epsilon \quad & \leq & \dfrac{y}{x+y} \\ 0 \leq \quad \epsilon \quad & \leq & \dfrac{2^{e_x}(2^{-p+1} - 2^{-2p+1})}{m_x 2^{e_x} + 2^{e_x}(2^{-p+1} - 2^{-2p+1})} \\ 0 \leq \quad \epsilon \quad & \leq & \dfrac{2^{-p+1} - 2^{-2p+1}}{m_x + (2^{-p+1} - 2^{-2p+1})} \end{array}$$

Il suffit donc de trouver une borne supérieure à $2^{-p+1} - 2^{-2p+1}/(m_x + (2^{-p+1} - 2^{-2p+1}))$. Cette fonction est maximale lorsque m_x est minimal, i.e. $m_x = 1.0$. On a donc

$$0 \leq \quad \epsilon \quad \leq \frac{2^{-p+1} - 2^{-2p+1}}{1 + (2^{-p+1} - 2^{-2p+1})}$$

En reprenant le même raisonnement que celui de la fin de la preuve de la première propriété, nous pouvons maintenant trouver l'approximation qui correspond à cette erreur relative

$$\gamma \times (x+y) \leq x \oplus y \leq x + y$$

$$\gamma = \frac{1}{1 + (2^{-p+1} - 2^{-2p+1})}$$

□

Notons que $1 + 2^{-p+1} - 2^{-2p+1} < 1 + 2^{-p+1}$ et que donc $\gamma > \alpha$. Cette seconde approximation est donc meilleure que la première.

Un résultat identique peut être obtenu pour la soustraction lorsque $x \geq y$. Cependant, ce n'est pas possible pour la multiplication et la division pour lesquelles seule la première approximation est correcte.

Application à l'exemple illustratif

Revenons à l'exemple introduit en section 3. En utilisant les approximations de la première proposition, la contrainte $x \ominus y \leq 2$ est approximée par

$$\begin{array}{rcl} x \ominus y & \in &]\alpha(x-y), x-y] \\ 2 & \in & [2,2] \\ x \ominus y \leq 2 & \Rightarrow &]\alpha(x-y), x-y] \leq [2,2] \\ & \Rightarrow & \alpha(x-y) \leq 2 \\ & \Rightarrow & \frac{1}{1+2^{-1}}(x-y) \leq 2 \\ & \Rightarrow & \frac{2}{3}(x-y) \leq 2 \\ & \Rightarrow & x - y \leq 3 \end{array}$$

La contrainte $(x \oplus y) \oplus y \leq 4$ est approximée par

$$\begin{array}{rcl} x \oplus y & \in &]\alpha(x+y),\ x+y] \\ (x \oplus y) \oplus y & \in &]\alpha(x+y),\ x+y] \oplus [y,y] \\ (x \oplus y) \oplus y & \in &]\alpha^2(x+y) + \alpha y,\ x+2y] \\ (x \oplus y) \oplus y & \in &](\frac{2}{3})^2(x+y) + \frac{2}{3}y,\ x+2y] \\ (x \oplus y) \oplus y & \in &]\frac{4}{9}x + \frac{10}{9}y,\ x+2y] \\ 4 & \in & [4,4] \\ (x \oplus y) \oplus y \leq 4 & \Rightarrow &]\frac{4}{9}x + \frac{10}{9}y,\ x+2y] \leq [4,4] \\ & \Rightarrow & \frac{4}{9}x + \frac{10}{9}y \leq 4 \\ & \Rightarrow & x + 2.5y \leq 9 \end{array}$$

Nous obtenons donc le système de contraintes suivant

$$\left\{ \begin{array}{rcl} x & \geq & 0 \\ y & \geq & 0 \\ x + 2.5y & \leq & 9 \\ x - y & \leq & 3 \end{array} \right.$$

Les solutions de ce système sont représentées dans la figure 2. En utilisant la seconde approximation, nous obtenons (avec $\gamma = \frac{8}{11}$)

$$\left\{ \begin{array}{rcl} x & \geq & 0 \\ y & \geq & 0 \\ x + 2.375y & \leq & 7.5625 \\ x - y & \leq & 2,75 \end{array} \right.$$

Les solutions de ce second système de contraintes sont représentées par la figure figure 3.

Le solveur de contraintes sur les réels utilisé peut imposer des limitations sur les coefficients des contraintes. Le plus souvent, ces solveurs utilisent les flottants de type double pour effectuer leurs calculs. Les coefficients des contraintes construites ici doivent prendre en compte ces limitations. Par exemple, lors du calcul de α pour la contrainte $\alpha(x+y) < x \oplus y \leq x+y$, $1+2^{-p+1}$ est calculé avec un arrondi à $+\infty$ et son inverse avec un arrondi vers $-\infty$ afin garantir que toutes les solutions sont préservées.

5 Traitement des approximations sur les réels

L'ensemble des approximations sur les réels précédement construites forme un système de contraintes sur les réels qu'il s'agit de traiter à l'aide d'algorithmes de filtrage sur les réels. Le choix de cet algorithme dépend du problème traité. Si les contraintes sont linéaires, on peut utiliser un solveur de programmation linéaire sur les réels tel que Cplex[1]. Lorsque le système obtenu n'est pas linéaire, il est possible d'utiliser un solveur non-linéaire tel que Icos ou encore de linéariser le problème.

Dans tout les cas, si l'algorithme de filtrage sur les réels utilisé montre qu'il n'existe aucune solution pour ce système de contraintes, alors le système initial de contraintes sur les flottants n'a, lui non plus, pas de solution.

$$S_{\mathbb{R}} = \emptyset \text{ et } S_{\mathbb{F}} \subseteq S_{\mathbb{R}} \Rightarrow S_{\mathbb{F}} = \emptyset$$

Cependant, dans le cas général, l'application de l'algorithme de filtrage sur les réels va uniquement se traduire par des réductions des domaines des variables. Potentiellement, un tel système admet des solutions sur les réels, solutions qui sont rarement des solutions du système de contraintes sur $\mathbb{F}$. Il est donc nécessaire d'utiliser un processus différent pour énumérer sur les flottants.

6 Travaux connexes

Michel et al. furent les premiers à s'intéresser à la problématique de la résolution de contraintes sur les nombres à virgule flottante [13]. Ces premiers travaux adaptent l'algorithme de *Box*-consistence aux nombres à virgule flottante, un algorithme initialement conçu pour le filtrage de contraintes sur les réels. Il consiste à réduire les intervalles de variables en éliminant les parties qui ne contiennent pas de solutions. Dans une seconde approche proposée dans [12], des projections de contraintes élémentaires sur les flottants ont permis de définir une version de la $2B$-consistence dédiée aux nombres à virgule flottante. Les fonctions de projection ont également été étendues dans [3]. Si ces deux approches permettent de résoudre des systèmes de contraintes sur les flottants, elles peinent à traiter des systèmes de contraintes conséquents. La démarche proposée dans cet article peut être vue comme une approche complémentaire de ces premiers solveurs sur les flottants : grâce à ces algorithmes sur les réels, des réductions supplémentaires des domaines des variables deviennent possibles.

D'autres travaux adoptent une démarche similaire à la notre, i.e. la sur-approximation de contraintes. Par exemple, les travaux de Miné [15, 14] étendent l'interprétation abstraite au cas des nombres à virgule flottante pour prouver l'absence des erreurs d'exécution telle que la division par zéro et le débordement. Cette approche consiste aussi à faire des sur-approximations sur les réels en calculant un sur ensemble des états atteignables. Cette méthode à été mise en œuvre dans l'outil Astrée [5]. Cependant, les sur-approximations introduites sont cependant moins précise que les approximations que nous avons présentées. Il existe d'autres travaux basée sur l'interprétation abstraite qui consistent à estimer l'erreur d'arrondi pour une expression sur les nombres flottants [9]. Ces travaux requièrent, à priori, les valeurs des variables pour en estimer l'erreur. Par ailleurs, cette dernière approche ne correspond pas à nos besoins qui sont le calcul de l'ensemble des états atteignables.

7 Conclusion

Nous avons présenté dans cet article une méthode de résolution de contraintes sur les nombres à virgule flottante. L'approche consiste à construire une approximation précise et conservative sur les réels du système initial de contraintes sur les nombres à virgule flottante. L'approximation construite peut alors être traitée par des algorithmes sur les réels sans qu'aucune solution sur les flottants ne soit perdue. Si les résultats préliminaires restent encourageants, une phase de validation de notre approche reste encore nécessaire. Actuellement, nous nous concentrons sur son implémentation et sa mise en œuvre effective dans le cadre de la vérification et du test de programmes.

[1] http ://www-01.ibm.com/software/integration/ optimization/cplex/

Références

[1] Ieee standard for floating-point arithmetic. *IEEE Std 754-2008*, pages 1 –58, 29 2008.

[2] S. Bardin, B. Botella, F. Dadeau, F. Charreteur, A. Gotlieb, B. Marre, C. Michel, M. Rueher, and N. Williams. Constraint-Based Software Testing. *Journée du GDR-GPL*, 9.

[3] B. Botella, A. Gotlieb, and C. Michel. Symbolic execution of floating-point computations. *Software Testing, Verification and Reliability*, 16(2), 2006.

[4] H. Collavizza, M. Rueher, and P. Van Hentenryck. CPBPV : A Constraint-Programming Framework for Bounded Program Verification. *Proc. of CP2008*, pages 327–341.

[5] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Mine, D. Monniaux, and X. Rival. The ASTREE analyzer. *Lecture Notes in Computer Science*, 3444 :21–30, 2005.

[6] V. D'Silva, D. Kroening, and G. Weissenbacher. A survey of automated techniques for formal software verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(7) :1165–1178, 2008.

[7] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1) :5–48, 1991.

[8] A. Gotlieb. Euclide : A Constraint-Based Testing Framework for Critical C Programs. In *Proceedings of the 2009 International Conference on Software Testing Verification and Validation-Volume 00*, pages 151–160. IEEE Computer Society Washington, DC, USA, 2009.

[9] E. Goubault, M. Martel, and S. Putot. Asserting the precision of floating-point computations : a simple abstract interpreter. In *European Symposium on Programming, ESOP'02*, volume 2305. Springer.

[10] IEEE. IEEE standard for binary floating-point arithmetic. *ANSI/IEEE Standard*, 754, 1985.

[11] J.L. Lions et al. Ariane 5 flight 501 failure. *Report by the Inquiry Board, Paris*, 19, 1996.

[12] C. Michel. Exact projection functions for floating point number constraints. In *Proc. of 7th AIMA Symposium, Fort Lauderdale (US)*, 2002.

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

[14] A. Miné. Relational abstract domains for the detection of floating-point run-time errors. *Lecture Notes in Computer Science*, 2986 :3–17, 2004.

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

[16] J.M. Muller, N. Brisebarre, F. de Dinechin, C.P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, S. Torrès, S. Pion, et al. *Handbook of Floating-Point Arithmetic*. 2009.

[17] Washington DC 20548 Report Number : GAOflMTEC-92-26 Or Publisher : US General Accounting Office, GAO. Patriot Missile Defense : Software Problem Led to System Failure at Dhahran, Saudi Arabia. 1992.