Utilisation de solveurs de contraintes pour réduire les approximations produites par interprétation abstraite

Olivier Ponsini Claude Michel Michel Rueher

Université de Nice–Sophia Antipolis, I3S/CNRS
BP 121, 06903 Sophia Antipolis Cedex, France
prénom.nom@unice.fr

Résumé

Les programmes comportant des calculs avec des nombres flottants sont délicats à concevoir car l'arithmétique flottante diffère de l'arithmétique réelle par de nombreuses propriétés contre-intuitives. Une approche classique pour vérifier de tels programmes consiste à estimer la précision des calculs avec des flottants par rapport à la même séquence d'opérations effectuées avec une sémantique idéale sur les réels. Des outils comme Fluctuat, basés sur l'interprétation abstraite, ont été développés pour traiter cette question. Toutefois, ces outils sont confrontés à un problème de sur-approximation du domaine des variables, aussi bien avec la sémantique des nombres flottants qu'avec la sémantique des nombres réels. Cette sur-approximation peut-être très grossière pour certains programmes. Dans cet article, nous montrons que des solveurs de contraintes sur les flottants et sur les réels peuvent améliorer significativement les approximations calculées par Fluctuat. Nous avons réussi à fortement réduire les domaines des variables de programmes C difficiles pour les techniques d'interprétation abstraite implémentées dans Fluctuat.

Abstract

Programs with floating-point computations are tricky to develop because floating-point arithmetic differs from real arithmetic and has many counterintuitive properties. A classical approach to verify such programs consists in estimating the precision of floating-point computations with respect to the same sequence of operations in an idealized semantics of real numbers. Tools like Fluctuat—based on abstract interpretation—have been designed to address this problem. However, such tools compute an over-approximation of the domains of the variables, both in the semantics of the floating-point numbers and in the semantics of the real numbers. This over-approximation can be very coarse on some programs. In this paper, we show that constraint solvers over floating-point numbers and real numbers can significantly refine the approximations computed by Fluctuat. We managed to reduce drastically the domains of variables of C programs that are difficult to handle for abstract interpretation techniques implemented in Fluctuat.

1 Introduction

De nombreuses applications logicielles critiques emploient des calculs avec des nombres flottants : notamment, les applications de contrôle ou de simulation des systèmes physiques dans différents domaines tels que le transport, le nucléaire, le médical, ou encore l'aéronautique et le spatial. Ces applications sont souvent basées sur des modèles mathématiques et des algorithmes conçus pour les nombres réels. Les nombres flottants sont alors une source supplémentaire possible d'erreurs, comme en témoignent les nombreuses défaillances tristement célèbres dues aux flottants, e.g., l'explosion de la fusée Ariane ou l'échec de l'antimissile Patriot. En effet, pour une même séquence d'opérations, le comportement des flottants n'est pas identique à celui des réels. La nature finie des nombres flottants a des conséquences qui vont à l'encontre de l'intuition acquise sur les réels [12, 21]. Par exemple, certains nombres décimaux d'apparence simples ne sont pas des flottants (e.g., $0,1$ n'a pas de représentation exacte sur les flottants binaires), les opérateurs arithmétiques ne sont pas associatifs et sujets aux phénomènes d'absorption (e.g., $a + b$ est arrondi à a quand a est très supérieur à b) ou d'annulation (soustraction de deux nombres proches après arrondi qui ne

conserve que l'erreur d'arrondi). En dépit du standard IEEE 754, ces phénomènes intrinsèques de l'arithmétique flottante dépendent aussi de nombreux facteurs tels que les compilateurs, les systèmes d'exploitations ou les architectures matérielles.

Malgré la connaissance de ces phénomènes, s'assurer qu'un programme avec des flottants approxime convenablement le modèle mathématique sur les réels dont il est issu reste très difficile. Des outils de vérification formelle ont été développés pour estimer l'exactitude des calculs flottants et pour montrer l'absence de certaines erreurs à l'exécution comme le débordement de capacité, les opérations invalides ou la division par zéro. Les outils automatiques existants sont principalement basés sur les techniques d'interprétation abstraite. En particulier, l'analyseur statique FLUCTUAT [10] a été employé avec succès pour étudier la propagation des erreurs d'arrondi dans des programmes C industriels critiques[1]. FLUCTUAT calcule deux sur-approximations des domaines des variables d'un programme considéré respectivement avec la sémantique des réels et avec la sémantique des flottants. FLUCTUAT est alors capable de borner l'erreur commise avec les flottants. Les sur-approximations calculées sont suffisamment précises pour les besoins de l'analyse dans certains cas. Cependant, des constructions usuelles des programmes conduisent à des sur-approximations trop grandes pour permettre à l'analyse d'être conclusive.

Dans cet article, nous montrons que des solveurs de contraintes sur les flottants et sur les réels peuvent raffiner significativement les approximations calculées par FLUCTUAT. Nous exploitons les capacités de réfutation des algorithmes de filtrage pour réduire les domaines calculés par interprétation abstraite. Notre implémentation utilise des solveurs de contraintes sur intervalles : REALPAVER [15] qui est un solveur correct de contraintes sur les réels et FPCS [20, 19] qui est un solveur correct de contraintes sur les flottants. Nous avons réussi à fortement réduire les domaines des variables de programmes C difficiles pour les techniques d'interprétation abstraite implémentées dans FLUCTUAT.

La section 2 illustre notre approche sur un petit exemple et présente les principaux travaux existants. La section 3 détaille notre approche et les différents outils qui la compose : FLUCTUAT, REALPAVER et FPCS. La section 4 discute les résultats de notre approche sur plusieurs programmes représentatifs.

[1] Les applications critiques concernées par les outils de vérification formelle des calculs flottants sont souvent embarquées et majoritairement écrites en langage C.

```
1 /* Pré−condition :  x ∈ [0 .. 10] */
2 double conditionnelle(double x) {
3   double y = x*x - x;
4
5   if (y >= 0)
6     y = x/10;
7   else
8     y = x*x + 2;
9
10  return y;
11 }
```

FIG. 1 – Intersection de domaines abstraits

2 Motivations

Dans cette section, nous illustrons notre approche par un exemple motivant, puis nous positionnons notre approche par rapport aux travaux existants.

2.1 Exemple

Le programme de la figure 1 est cité dans [11] pour les difficultés qu'il pose aux analyses par interprétation abstraite. Sur les flottants, comme sur les réels, cette fonction renvoie une valeur comprise dans l'intervalle $[0 .. 3]$ alors que l'outil FLUCTUAT calcule l'intervalle $[0..102]$. En effet, nous pouvons déduire de l'instruction conditionnelle de la ligne 5 que :

- branche `si` : $x = 0$ ou $x \geq 1$ et donc $y \in [0 .. 1]$;
- branche `sinon` : $x \in]0 .. 1[$ et donc $y \in [2 .. 3]$.

Cependant, tous les domaines abstraits classiques (e.g., intervalles, polyèdres), ainsi que le domaine des *zonotopes* de FLUCTUAT, échouent à obtenir une bonne approximation de cette valeur. La difficulté pour ces analyses est de réaliser l'intersection des domaines abstraits calculés pour y en lignes 3 et 5. Ces analyses sont notamment incapables d'en déduire une contrainte sur x ; elles considèrent ainsi que dans la branche `sinon` le domaine de x est toujours l'intervalle $[0 .. 10]$.

Dans l'approche que nous proposons, nous calculons une approximation des domaines dans chacun des deux chemins d'exécution. Les techniques de filtrage de la programmation par contraintes sont suffisamment fortes pour réduire les domaines des variables des systèmes de contraintes correspondants générés. Considérons par exemple le système de contraintes sur les réels $\{y_0 = x_0 * x_0 - x_0, y_0 < 0, y_1 = x_0 * x_0 + 2, x_0 \in [0..10]\}$ qui correspond au chemin d'exécution passant par la branche `sinon` dans la fonction `conditionnelle`[2]. À

[2] Les instructions des programmes sont converties en forme SSA (*Static Single Assignment*) dans laquelle chaque variable n'est affectée qu'une seule fois dans chaque chemin d'exécution.

	Domaine	Temps
Exact (réels et flottants)	[0 .. 3]	n.d.
FLUCTUAT (réels et flottants)	[0 .. 102]	0,1 s
Zonotopes contraints (réels)	[0 .. 9,72]	n.d.
FPCS (flottants)	[0 .. 3,027]	0,2 s
REALPAVER (réels)	[0 .. 3,001]	0,3 s

TAB. 1 – Domaine de la fonction `conditionnelle`

partir des contraintes $y_0 = x_0 * x_0 - x_0$ et $y_0 < 0$, un solveur de contraintes sur intervalles peut réduire le domaine initial de x_0 à $[0 .. 1]$. Ce domaine réduit servira alors à calculer celui de y_1 via la contrainte $y_1 = x_0 * x_0 + 2$, pour produire $y_1 \in [2 .. 3{,}001]$. De la même façon, un solveur de contraintes sur les flottants réduit le domaine de x_0 à $[4{,}94 \cdot 10^{-324} .. 1{,}026]$ et celui de y_1 à $[2 .. 3{,}027]$.

En résumé, nous construisons les systèmes de contraintes correspondant à chaque chemin exécutable dans une fonction et nous utilisons des techniques de filtrage pour réduire les domaines des variables calculés par FLUCTUAT. Les chemins d'exécution sont explorés à la volée et interrompus aussitôt qu'est détectée l'inconsistance du système de contraintes associé. Sur les nombres réels, nous utilisons la combinaison des consistances de boîte (*box*) et d'enveloppe (*hull*) implémentée dans REALPAVER [15]. Sur les nombres flottants, nous utilisons la $3B$-consistance [18] implémentée dans FPCS [20]. Les résultats des différentes méthodes sur l'exemple de la fonction `conditionnelle` sont donnés dans la table 1. Sur cet exemple, contrairement à FLUCTUAT, notre approche fournit une très bonne approximation, à la fois sur les réels et sur les flottants. Les temps d'analyse sont très similaires (n.d. signifie non disponible). Dans [11], les auteurs proposent une extension aux zonotopes, appelée zonotopes contraints (*constrained zonotopes*), qui tente de pallier le problème dû aux instructions conditionnelles des programmes. Cette extension est définie sur les réels et n'est pas encore implémentée dans FLUCTUAT. L'approximation calculée avec les zonotopes contraints est meilleure que celle de FLUCTUAT, mais elle reste moins précise que celle calculée avec REALPAVER.

2.2 Travaux connexes

Différentes méthodes ont été appliquées au problème de la validation statique des programmes comportant des calculs en nombres flottants : notamment, les analyses par interprétation abstraite et les preuves de programmes avec des assistants de preuve ou avec des procédures de décision dans des solveurs automatiques.

Les méthodes d'analyse par interprétation abstraite représentent les erreurs d'arrondi des calculs en arithmétique flottante dans leurs domaines abstraits. Elles sont en général rapides, automatiques et passent bien à l'échelle. En revanche, elles peuvent manquer de précision et ne génèrent pas de contre-exemple. ASTRÉE [7] est sans doute l'un des outils les plus connus issu de cette famille de méthodes. En estimant la valeur des variables en tout point d'un programme, ASTRÉE peut prouver l'absence d'erreur à l'exécution, au sens de comportement non défini par le langage de programmation (e.g., division par zéro, débordement de capacité). FLUCTUAT, que nous détaillons en section 3.1, calcule en plus une estimation de l'exactitude des calculs, c.-à-d., une borne de la différence entre les valeurs des variables lorsque le programme est interprété avec une sémantique sur les réels et lorsqu'il est interprété avec une sémantique sur les flottants [10].

Un deuxième groupe de méthodes s'attache à formaliser l'arithmétique flottante dans des assistants de preuve tels que Coq [2] ou HOL [16]. Les preuves des propriétés des programmes sont alors effectuées interactivement dans l'assistant, qui garantit la correction des preuves. Ces formalisations ne sont pas adaptées à l'évaluation des domaines des variables des programmes. De plus, lorsque la construction d'une preuve échoue, cela n'implique pas qu'une propriété est fausse. Par conséquent, ces approches ne peuvent fournir aucun contre-exemple. L'outil Gappa [9] combine le calcul d'intervalle et l'application de théorèmes prédéfinis par réécriture. Les théorèmes permettent de réécrire les expressions arithmétiques afin de compenser certaines des faiblesses du calcul d'intervalle, e.g., perte des dépendances entre variables. Lorsque les intervalles calculés ne sont pas assez précis, il est possible d'ajouter manuellement des théorèmes ou de subdiviser les domaines d'entrée. Le coût de cette méthode semi-automatique est alors important. Dans [1], les auteurs proposent une axiomatisation de l'arithmétique flottante dans la logique du premier ordre qui permet d'automatiser les preuves dans un assistant tel que Coq en appelant des solveurs SMT (*Satisfiability Modulo Theories*) et Gappa. Leurs expérimentations montrent qu'une interaction humaine avec l'assistant de preuve demeure nécessaire. Du fait de la taille du domaine des variables flottantes, l'approche classique par vecteurs de bits des solveurs SAT est inopérante. Une technique d'abstraction a été proposée pour CBMC [4]. Cette technique est basée sur la sous et la sur-approximation d'un flottant selon une précision exprimée en nombre de bits de la mantisse. Cette technique reste cependant lente.

3 Approche proposée

L'approche que nous proposons consiste à raffiner les domaines des variables calculés par FLUCTUAT en utilisant des solveurs de contraintes sur les réels et les flottants. Avant d'exposer les détails de notre approche, nous rappelons les caractéristiques de FLUCTUAT (section 3.1), REALPAVER (section 3.2) et FPCS (section 3.3) qui sont utiles à la compréhension de la suite de cet article. L'ensemble du processus de l'approche que nous proposons est décrit en section 3.4.

3.1 Fluctuat

FLUCTUAT [10] est un analyseur statique de programmes C spécialisé dans l'évaluation de la précision des calculs numériques avec des flottants. L'outil compare le comportement du programme analysé sur les réels et sur les flottants. Concrètement, FLUCTUAT permet de spécifier des intervalles de valeurs pour les variables d'entrée d'un programme et calcule pour chaque variable du programme :

- un encadrement du domaine de la variable considérée sur les réels ;
- un encadrement du domaine de la variable considérée sur les flottants ;
- un encadrement de l'erreur maximum entre la valeur réelle et flottante ;
- la contribution de chaque instruction à l'erreur associée à la variable ;
- la contribution des variables d'entrée à l'erreur associée à la variable.

FLUCTUAT procède par interprétation abstraite et utilise le domaine abstrait faiblement relationnel des zonotopes [13], qui établit un bon compromis entre performance et précision. Les zonotopes sont des ensembles de formes affines qui préservent les corrélations linéaires entre variables. Pour améliorer la précision de l'analyse, l'outil permet d'utiliser des nombres en précision arbitraire ou de subdiviser les domaines des variables d'entrée.

FLUCTUAT est développé par le CEA-LIST[3] et a été utilisé avec succès pour des applications industrielles de plusieurs dizaines de milliers de lignes de code dans les domaines du transport, du nucléaire ou de l'aéronautique.

3.2 RealPaver

REALPAVER[4] [15] est un solveur sur intervalles pour des systèmes de contraintes numériques sur les nombres réels. Les contraintes peuvent être non-linéaires et peuvent contenir les opérateurs arithmétiques usuels ainsi que les fonctions transcendantes élémentaires. En revanche, ni les opérateurs d'inégalité stricte, ni les disjonctions de contraintes ne sont disponibles.

Le solveur calcule des approximations fiables d'ensembles continus de solutions en utilisant des méthodes du calcul des intervalles correctement arrondies et des techniques de satisfaction de contraintes. Plus précisément, les domaines calculés sont des intervalles fermés dont les bornes sont des flottants. REALPAVER implémente plusieurs consistances partielles : *box*, *hull* et $3B$ par exemple. L'approximation d'une solution est décrite par une boîte, c.-à-d., le produit cartésien des domaines des variables. Soit REALPAVER prouve qu'un système de contraintes est inconsistant, soit il calcule une union de boîtes qui contient toutes les solutions du système.

3.3 FPCS

FPCS [20, 19] est un solveur de contraintes qui a été conçu pour résoudre correctement, c.-à-d. sans perdre de solutions, un ensemble de contraintes sur les flottants. Pour ce faire, FPCS utilise une $2B$-consistance [18] ainsi que des fonctions de projections adaptées aux flottants [20, 3]. La principale difficulté de cette démarche réside dans l'élaboration de fonctions de projections inverses conservatives de l'ensemble des solutions. En effet, si les projections directes, c.-à-d. le calcul du domaine de y à partir du domaine de x pour une contrainte du type $y = f(x)$, ne nécessitent qu'une légère adaptation des résultats classiques de l'arithmétique des intervalles, les projections inverses, c.-à-d. le calcul du domaine de x à partir de celui de y, ne répondent pas aux mêmes règles en raison des propriétés de l'arithmétique flottante. L'ensemble de ces résultats est détaillé dans [20] et étendu dans [3]. FPCS implémente aussi des consistances plus fortes comme les kB-consistances [18] plus à même de traiter les classiques problèmes d'occurrences multiples, mais aussi, de réduire de manière plus conséquente les bornes des domaines des variables.

Contrairement à la plupart des approches citées en section 2.2, les domaines des flottants manipulés par FPCS ne se réduisent pas aux valeurs numériques mais incluent aussi les infinis. De plus, le solveur FPCS traite l'ensemble des opérations arithmétiques de base ainsi que la plupart des fonctions mathématiques usuelles. Les conversions de type sont aussi correctement gérées.

Le comportement des programmes qui contiennent des calculs flottants peut varier avec le langage de programmation ou le compilateur, mais aussi avec le

[3] FLUCTUAT : `http://www-list.cea.fr/labos/fr/LSL/fluctuat/index.html`

[4] REALPAVER : `http://pagesperso.lina.univ-nantes.fr/info/perso/permanents/granvil/realpaver/`

système d'exploitation ou l'architecture d'exécution. FPCS cible des programmes C, compilés avec GCC sans option d'optimisation et destinés à une architecture x86 gérée par un système d'exploitation Linux 32 bits.

3.4 Mise en œuvre

Nous pouvons maintenant détailler notre approche. Les étapes du processus mis en œuvre dans l'approche que nous proposons sont les suivantes :

1. Pour un programme C donné, nous calculons avec FLUCTUAT une première approximation des domaines des variables sur les réels et sur les flottants.
2. Nous analysons le programme C et construisons deux systèmes de contraintes pour chaque chemin d'exécution atteignable (voir détails ci-dessous) : l'un avec des variables réelles et l'autre avec des variables flottantes. Nous affectons les domaines estimés par FLUCTUAT aux variables du programme.
3. Pour chaque système de contraintes, nous filtrons les domaines avec une consistance partielle.
   Nous utilisons FPCS et une $3B(w)$-consistance sur les systèmes avec les variables flottantes.
   Nous utilisons REALPAVER et sa consistance $BC5$ sur les systèmes avec les variables réelles. La $BC5$-consistance combine la méthode de Newton sur les intervalles et les *box* et *hull*-consistances.
4. Pour chaque variable de sortie, nous calculons l'union de tous les domaines obtenus à l'étape 3.

Lors de l'étape 2, nous explorons chaque chemin d'exécution d'un programme, c.-à-d. chaque chemin du graphe de flot de contrôle du programme, par une analyse en avant (du début à la fin du programme). Les instructions sont converties en forme SSA (*Static Single Assignment*), dans laquelle chaque variable n'est affectée qu'une seule fois dans chaque chemin d'exécution [8]. La longueur des chemins est bornée car les appels récursifs de fonction ne sont pas autorisés et les boucles sont dépliées un nombre fini de fois précisé par l'utilisateur. Les états possibles du programme en tout point d'un chemin d'exécution sont représentés par un système de contraintes composé d'un ensemble fini de variables et de contraintes sur ces variables. Avec FPCS, les variables ont des domaines qui correspondent aux implémentations en machine des types du langage C (`int`, `float` et `double`) ; avec REALPAVER, les domaines sont des intervalles sur les réels. Des règles définissent comment chaque instruction d'un programme modifie les états possibles du programme en ajoutant de nouvelles contraintes et variables.

Les chemins d'exécution sont explorés à la volée et interrompus dès que l'inconsistance du système de contraintes associé est détectée : simple filtrage par $3B$-consistance avec FPCS et par *hull*-consistance $HC4$ avec REALPAVER. Ceci permet de limiter l'explosion combinatoire du nombre des chemins en n'explorant que ceux qui sont exécutables. Cette technique de représentation des programmes par des systèmes de contraintes a été introduite pour la vérification bornée des programmes avec CPBPV [5]. Son implémentation pour l'approche proposée dans cet article s'appuie sur les bibliothèques développées pour CPBPV.

Le langage de modélisation de REALPAVER ne dispose pas des opérateurs d'inégalité stricte. Ceux-ci peuvent néanmoins apparaître dans les expressions conditionnelles des programmes. Par conséquent, dans les systèmes de contraintes générés pour REALPAVER, les inégalités strictes sont remplacées par leur équivalent non strict et les contraintes contenant l'opérateur « différent de » sont ignorées. Ceci peut conduire à des sur-approximations, mais aucune solution n'est perdue.

4 Expérimentations et discussion

Dans cette section, nous présentons les résultats de notre approche sur des programmes caractéristiques qui montrent quand notre approche améliore l'approximation calculée par FLUCTUAT. Les résultats ont été obtenus avec une machine Linux sur architecture Intel Core 2 Duo à 2,8 GHz avec 4 Go de mémoire. Nous avons utilisé FLUCTUAT version 3.8.22_opt et REALPAVER version 0.4. Dans les tables qui suivent, les domaines réels et flottants sont dénotés par les symboles $\mathbb{R}$ et $\mathbb{F}$ respectivement. Les nombres sont arrondis à trois chiffres après la virgule par souci de lisibilité.

4.1 Instructions conditionnelles

Nous avons présenté le problème de l'intersection de domaines pour les analyses abstraites en section 2.1. Ici, nous illustrons le comportement de notre approche dans cette situation sur le programme du calcul des racines des équations du second degré. Ce programme est donné en figure 2 et a été extrait de la bibliothèque scientifique GNU (GSL [23]). Les racines sont calculées dans les variables `x0` et `x1`. Le programme fait appel à deux fonctions de la bibliothèque C : `fabs` et `sqrt`, pour la valeur absolue et la racine carrée d'un nombre, respectivement. La fonction `sqrt` est d'ailleurs l'une des seules fonctions à être définies par le standard IEEE 754. Ces fonctions sont directement gérées par FPCS et peuvent donc apparaître telles quelles dans des contraintes. REALPAVER dispose d'une fonction

```
int quadratic(double a, double b, double c) {
  double r, sgnb, temp, r1, r2
  double disc = b * b - 4 * a * c;
  if (a == 0) {
    if (b == 0)
      return 0;
    else {
      x0 = -c / b;
      return 1;
    }
  }
  if (disc > 0) {
    if (b == 0) {
      r = fabs (0.5 * sqrt (disc) / a);
      x0 = -r;
      x1 = r;
    } else {
      sgnb = (b > 0 ? 1 : -1);
      temp = -0.5 * (b + sgnb * sqrt (disc));
      r1 = temp / a;
      r2 = c / temp;
      if (r1 < r2) {
        x0 = r1;
        x1 = r2;
      } else {
        x0 = r2;
        x1 = r1;
      }
    }
    return 2;
  } else if (disc == 0) {
    x0 = -0.5 * b / a;
    x1 = -0.5 * b / a;
    return 2;
  } else
    return 0;
}
```

FIG. 2 – Équation du second degré

`sqrt` prédéfinie, en revanche `fabs(x)` a été remplacé par `max(x,-x)`.

Nous donnons dans la table 2 les temps d'analyse et les approximations des domaines des variables `x0` et `x1` obtenus avec deux configurations des domaines des variables d'entrée. Les deux premières lignes de la table présentent les résultats de FLUCTUAT et de REALPAVER sur les réels. Les deux lignes suivantes présentent les résultats de FLUCTUAT et de FPCS sur les flottants. Dans la première configuration, où $a \in [-1 \,..\, 1]$, $b \in [0{,}5 \,..\, 1]$ et $c \in [0 \,..\, 2]$, la sur-approximation de FLUCTUAT est tellement importante qu'elle n'apporte aucune information sur le domaine des racines, alors que notre approche est capable de réduire significativement ces domaines à la fois sur $\mathbb{R}$ et sur $\mathbb{F}$. Néanmoins, le problème d'intersection des domaines a parfois moins de conséquences sur les bornes de tous les domaines. Ceci est illustré par le domaine sur $\mathbb{F}$ de `x0` dans la seconde configuration où $a, b, c \in [1 \,..\, 1 \cdot 10^6]$. Bien que le domaine calculé par FLUCTUAT soit une sur-approximation, notre approche ne parvient pas à le réduire. En revanche, dans cette même configuration, notre approche réalise de nouveau une très bonne réduction pour le domaine de `x1`.

Pour augmenter la précision de l'analyse, FLUCTUAT offre la possibilité de diviser les domaines d'au plus deux des variables d'entrée en un nombre paramétrable de sous-domaines. Les analyses sont alors effectuées sur chaque combinaison de sous-domaines et les résultats sont rassemblés. Sans connaissance a priori des domaines à subdiviser, toutes les combinaisons à un, puis à deux domaines doivent être essayées. Le nombre de subdivisions de chaque domaine est lui aussi difficile à déterminer ; il doit être choisi le plus grand possible tout en maintenant un temps d'analyse acceptable, et ceci sans garantie d'améliorer la précision. Pour la table 3, nous avons fixé les subdivisions à 50 lorsqu'un seul domaine est divisé et à 30 pour chaque domaine sinon. Nous ne présentons que les résultats sur $\mathbb{F}$ dans la table. Sur $\mathbb{R}$, dans la première configuration, les subdivisions n'apportent aucune amélioration ; dans la seconde configuration, les résultats sont identiques à ceux sur $\mathbb{F}$. Le coût en temps de ces subdivisions peut être important au regard du gain en précision obtenu :

- Dans la première configuration, les subdivisions du domaine de a entraînent une réduction intéressante du domaine de `x0` (identique à celle obtenue avec notre approche). Aucune combinaison de subdivisions ne réduit le domaine de `x1`.
- Dans la seconde configuration, la meilleure réduction du domaine de `x1` est obtenue en subdivisant à la fois les domaines de a et de b. Le gain reste toutefois assez faible par rapport à celui obtenu par notre approche. Aucune combinaison de subdivisions ne réduit le domaine de `x0`.

Lorsqu'il est nécessaire de subdiviser tous les domaines d'entrée, le coût devient prohibitif. Notre approche s'avère bien plus efficace : elle améliore souvent la précision de l'approximation et son coût est faible, que la précision soit améliorée ou non. En outre, la technique des subdivisions serait aussi applicable à notre approche.

4.2 Expressions non-linéaires

Le domaine abstrait utilisé par FLUCTUAT est basé sur des formes affines, qui ne permettent pas une représentation exacte des opérations non-linéaires : l'image d'un zonotope par une fonction non-linéaire n'est pas

		conf. #1 : $a \in [-1 .. 1]$ $b \in [0{,}5 .. 1]$ $c \in [0 .. 2]$			conf. #2 : $a, b, c \in [1 .. 1 \cdot 10^6]$		
		`x0`	`x1`	Temps	`x0`	`x1`	Temps
$\mathbb{R}$	FLUCTUAT	$[-\infty .. \infty]$	$[-\infty .. \infty]$	0,1 s	$[-2 \cdot 10^6 .. 0]$	$[-1 \cdot 10^6 .. 0]$	0,1 s
	REALPAVER	$[-\infty .. 0]$	$[-8{,}011 .. \infty]$	1,5 s	$[-1 \cdot 10^6 .. 0]$	$[-5{,}186 \cdot 10^5 .. 0]$	0,5 s
$\mathbb{F}$	FLUCTUAT	$[-\infty .. \infty]$	$[-\infty .. \infty]$	0,1 s	$[-2 \cdot 10^6 .. 0]$	$[-1 \cdot 10^6 .. 0]$	0,1 s
	FPCS	$[-\infty .. 0]$	$[-8{,}064 .. \infty]$	0,3 s	$[-2 \cdot 10^6 .. 0]$	$[-2\ 503{,}709 .. 0]$	0,3 s

TAB. 2 – Domaine des racines de la fonction `quadratic`

	conf. #1		conf. #2	
	`x0`	Temps	`x1`	Temps
a subdivisé	$[-\infty .. -0]$	> 1 s	$[-1 \cdot 10^6 .. 0]$	> 1 s
b subdivisé	$[-\infty .. \infty]$	> 1 s	$[-5 \cdot 10^5 .. 0]$	> 1 s
c subdivisé	$[-\infty .. \infty]$	> 1 s	$[-1 \cdot 10^6 .. 0]$	> 1 s
a et b subdivisés	$[-\infty .. -0]$	> 10 s	$[-1{,}834 \cdot 10^5 .. 0]$	> 10 s
a et c subdivisés	$[-\infty .. -0]$	> 10 s	$[-1 \cdot 10^6 .. 0]$	> 10 s
b et c subdivisés	$[-\infty .. \infty]$	> 10 s	$[-5 \cdot 10^5 .. 0]$	> 10 s

TAB. 3 – Domaines de FLUCTUAT sur $\mathbb{F}$ pour la fonction `quadratic` avec subdivisions

```
double sinus(double x) {
  return x - x*x*x/6 + x*x*x*x*x/120
         + x*x*x*x*x*x*x/5040;
}
```

FIG. 3 – sinus

un zonotope en général. Les opérations non-linéaires sont alors approximées en introduisant un terme d'erreur. Les solveurs de contraintes que nous utilisons gèrent mieux les expressions non-linéaires. Ainsi, sur l'exemple du développement limité à l'ordre 7 de la fonction sinus en figure 3, notre approche améliore significativement l'approximation de FLUCTUAT (cf. table 4, colonne `sinus`).

Néanmoins, les solveurs de contraintes que nous utilisons, eux aussi, approximent les expressions non-linéaires. Ceci est illustré par le polynôme de Rump (en figure 4) extrait de [22]. Ce polynôme assez singulier a été conçu pour faire apparaître un phénomène d'annulation catastrophique sur certaines architectures quelle que soit la précision des flottants. Ni REALPAVER ni FPCS ne parviennent à réduire les domaines calculés par FLUCTUAT (cf. table 4, colonne `rump`).

4.3 Instructions itératives

FLUCTUAT déplie les boucles un nombre borné de fois avant d'appliquer l'opérateur d'élargissement (*widening*) de l'interprétation abstraite (dix dépliages par défaut). L'opérateur d'élargissement permet d'atteindre un point fixe et de terminer rapidement. Cependant, cet opérateur peut conduire à de très grandes sur-approximations. Cette situation se produit lors de l'analyse de la valeur renvoyée par le programme `sqrt`, qui calcule une valeur approchée à $1 \cdot 10^{-2}$ de la racine carrée d'un nombre supérieur à 4. L'algorithme de `sqrt` est basé sur la méthode dite babylonienne (cf. figure 5). FLUCTUAT ne réalise aucune réduction du domaine de la valeur de retour de `sqrt` sur $\mathbb{F}$ pour la configuration où $x \in [5 .. 10]$ (cf. table 5).

```
double rump(double x, double y) {
  double f;
  f = 333.75*y*y*y*y*y*y*y;
  f = f + x*x*(11*x*x*y*y - y*y*y*y*y*y*y
               - 121*y*y*y*y - 2);
  f = f + 5.5*y*y*y*y*y*y*y*y*y;
  f = f + x / (2*y);
  return f;
}
```

FIG. 4 – Polynôme de Rump

Dans notre approche, nous n'essayons pas d'analyser le comportement des boucles : nous déplions simplement les boucles N fois, où N est fixé par l'utilisateur[5]. Nous n'essayons de réduire les domaines calculés par FLUCTUAT que si les conditions d'entrée des boucles sont fausses pour un nombre de dépliages k inférieur à N.

La table 5 présente les résultats que nous obtenons avec $N = 10$. Dans notre approche, les dé-

[5] Pour estimer une borne du nombre de dépliages nécessaires, nous pouvons aussi utiliser FLUCTUAT [14].

		`sinus` $x \in [-1 .. 1]$		`rump` $x \in [7 \cdot 10^4 .. 8 \cdot 10^4]$ $y \in [3 \cdot 10^4 .. 4 \cdot 10^4]$	
		Domaine	Temps	Domaine	Temps
$\mathbb{R}$	Fluctuat	$[-1{,}009 .. 1{,}009]$	0,1 s	$[-1{,}168 \cdot 10^{37} .. 1{,}992 \cdot 10^{37}]$	0,1 s
	RealPaver	$[-0{,}842 .. 0{,}843]$	0,3 s	$[-1{,}144 \cdot 10^{36} .. 1{,}606 \cdot 10^{37}]$	1,2 s
$\mathbb{F}$	Fluctuat	$[-1{,}009 .. 1{,}009]$	0,1 s	$[-1{,}168 \cdot 10^{37} .. 1{,}992 \cdot 10^{37}]$	0,1 s
	FPCS	$[-0{,}853 .. 0{,}852]$	0,2 s	$[-1{,}168 \cdot 10^{37} .. 1{,}992 \cdot 10^{37}]$	0,2 s

Tab. 4 – Domaines de `sinus` et `rump`

		conf. #1 : $x \in [4{,}5 .. 5{,}5]$		conf. #2 : $x \in [5 .. 10]$	
		Domaine	Temps	Domaine	Temps
$\mathbb{R}$	Fluctuat	$[2{,}116 .. 2{,}354]$	0,1 s	$[2{,}098 .. 3{,}435]$	0,1 s
	RealPaver	$[2{,}121 .. 2{,}346]$	0,3 s	$[2{,}232 .. 3{,}165]$	0,5 s
$\mathbb{F}$	Fluctuat	$[2{,}116 .. 2{,}354]$	0,1 s	$[-\infty .. \infty]$	0,1 s
	FPCS	$[2{,}120 .. 2{,}347]$	1 s	$[2{,}232 .. 3{,}168]$	1,6 s

Tab. 5 – Domaine de `sqrt`

```
double sqrt(double x) {
  double xn, xn1;
  xn = x/2;
  xn1 = 0.5*(xn + x/xn);
  while (xn-xn1 > 1e-2) {
    xn = xn1;
    xn1 = 0.5*(xn + x/xn);
  }
  return xn1;
}
```

Fig. 5 – Racine carrée

pliages peuvent rapidement devenir coûteux en temps, mais le gain en précision peut aussi être important : sur $\mathbb{F}$, dans la seconde configuration, nous calculons pour `sqrt` le domaine $[2{,}232 .. 3{,}168]$ au lieu du domaine $[-\infty .. \infty]$ obtenu par Fluctuat. RealPaver est plus rapide que FPCS car nous employons avec lui une consistance plus faible pour détecter les chemins d'exécution inatteignables (*hull*-consistance pour RealPaver *versus* $3B$-consistance pour FPCS). Notons que RealPaver ne termine pas en un temps raisonnable si nous prenons $[-\infty .. \infty]$ comme domaine initial de la valeur de retour de `sqrt` au lieu du domaine calculé par Fluctuat.

4.4 Discussion

Les techniques d'interprétation abstraite calculent des approximations des domaines des variables sur une relaxation du problème initial. Dans le cas de Fluctuat, des ensembles de formes affines abstraient les expressions non-linéaires et les contraintes issues des programmes. Cette première approximation produit souvent des domaines suffisamment petits pour permettre un filtrage efficace avec des consistances partielles qui ne reposent pas sur la même relaxation.

Le filtrage par $3B$-consistance fonctionne bien avec FPCS. La $2B$-consistance n'est pas assez forte pour réduire les domaines calculés par Fluctuat, alors qu'une kB-consistance, plus forte, est trop coûteuse en temps. Nous avons aussi évalué plusieurs consistances implémentées dans RealPaver. La table 6 présente les résultats les plus significatifs. $BC5$ est une combinaison de *hull*-consistance et de *box*-consistance avec la méthode de Newton sur les intervalles. $HC4$ est une *hull*-consistance sur les contraintes utilisateurs. Le délai maximum (T.O.) était fixé à 5 minutes. La $3B$-consistance est difficile à contrôler au moyen de la spécification de la largeur des sous-domaines à réfuter, c'est pourquoi nous avons introduit la version $3B$ temporisée : il s'agit de la $3B$-consistance interrompue après 0,5 s de filtrage. Il ressort que la $3B$-consistance est trop coûteuse en temps. Une consistance plus faible comme la $BC5$ représente un meilleur compromis entre temps d'exécution et réduction des domaines.

Bien que les mêmes approximations des domaines puissent parfois être obtenues sans partir de celles calculées par Fluctuat (c.-à-d. en prenant pour domaines initiaux $[-\infty .. \infty]$), nos expérimentations montrent que notre approche donne généralement de meilleurs résultats lorsque nous utilisons les approximations calculées par Fluctuat. Par exemple, pour la fonction `sqrt` de la figure 5, RealPaver excède le délai maximum fixé pour filtrer les domaines si ceux-ci sont tous initialisés avec l'intervalle $[-\infty .. \infty]$; alors que le filtrage prend moins qu'une demi-seconde si les

	`quadratic` x1 conf. #1		`quadratic` x1 conf. #2		`sqrt` conf. #1	
	Domaine	Temps	Domaine	Temps	Domaine	Temps
$3B$	n.d.	T.O.	n.d.	T.O.	n.d.	T.O.
$3B$ temporisée (0,5 s)	$[-11{,}444 .. \infty]$	3 s	$[-749\ 999{,}721 .. 0]$	2,4 s	$[2{,}12 .. 2{,}346]$	1,3 s
weak$3B$	$[-8{,}004 .. \infty]$	9,3 s	$[-206\ 746{,}455 .. 0]$	5 s	$[2{,}121 .. 2{,}346]$	1 s
$BC5$	$[-8{,}011 .. \infty]$	1,5 s	$[-518\ 518{,}519 .. 0]$	0,5 s	$[2{,}121 .. 2{,}346]$	0,3 s
$HC4$	$[-8{,}223 .. \infty]$	0,8 s	$[-518\ 518{,}519 .. 0]$	0,5 s	$[2{,}121 .. 2{,}346]$	0,3 s

Tab. 6 – Comparaison des consistances de RealPaver

domaines sont initialisés avec les bornes calculées par Fluctuat. De même, pour la fonction `quadratic` de la figure 2, FPCS calcule le domaine $[-5{,}001 \cdot 10^{11} .. 0]$ en partant de $[-\infty .. \infty]$, alors que Fluctuat produit une meilleure approximation avec le domaine $[-2 \cdot 10^6 .. 0]$.

Bien sûr, les techniques de filtrage de la programmation par contraintes ne parviennent pas toujours à raffiner les approximations calculées par Fluctuat, en particulier lorsque la relaxation effectuée par Fluctuat se révèle appropriée pour calculer de bonnes approximations des domaines (e.g., avec les systèmes linéaires). De plus, même lorsque le filtrage serait en mesure de réduire les domaines, le temps de calcul peut être trop long, par exemple quand un grand nombre de dépliages des boucles est nécessaire ou quand des phénomènes de convergence lente se produisent dans FPCS [19].

5 Conclusion

Dans cet article, nous avons introduit une nouvelle approche pour raffiner les approximations des domaines des variables calculées par l'analyseur statique de programmes C Fluctuat. Cette approche repose sur des solveurs de contraintes, RealPaver et FPCS, qui sont corrects sur les nombres réels et les nombres flottants respectivement. Nous exploitons la capacité de réfutation des consistances partielles pour réduire les domaines calculés par Fluctuat. Nous avons montré que cette approche est rapide et efficace sur des programmes caractéristiques des difficultés de Fluctuat (constructions conditionnelles et non-linéaires).

Cependant, notre approche ne se substitue pas à Fluctuat. Les outils basés sur l'interprétation abstraite tels que Fluctuat calculent efficacement une approximation *globale* des domaines : en d'autres termes, l'interprétation abstraite d'un programme fusionne en chaque point du programme les domaines issus de l'analyse de tous les chemins du programme [6]. L'approximation globale des instructions conditionnelles et l'opération d'élargissement dans les instructions itératives facilitent le passage à l'échelle mais au prix de sur-approximations qui peuvent être très imprécises. D'autre part, les zonotopes constituent de meilleures approximations des systèmes de contraintes linéaires que les boîtes utilisées dans les solveurs de contraintes sur intervalles. Toutefois, les zonotopes sont moins bien adaptés aux systèmes de contraintes non-linéaires. Les techniques de filtrage employées dans des solveurs de systèmes de contraintes numériques offrent quant à elles un cadre flexible et extensible pour traiter les contraintes non-linéaires. L'exploration de chaque chemin d'exécution séparément est essentielle au calcul d'approximations précises. Cependant, afin de limiter l'explosion combinatoire, nous devons borner la longueur des chemins. Par conséquent, l'approche proposée dans cet article est complémentaire de celle de Fluctuat : notre approche donne de meilleurs résultats lorsque Fluctuat a réduit les domaines auparavant.

L'extension naturelle de ce travail est d'étudier comment Fluctuat peut être combiné plus étroitement avec des solveurs de contraintes. Des consistances plus fortes sont bien sûr envisageables pour améliorer encore la précision des approximations, e.g., des contraintes globales comme la *Quad* sur les expressions non-linéaires [17] ou la kB-consistance appliquée aux domaines des variables de sortie uniquement.

Remerciements

Les auteurs remercient Sylvie Putot et Éric Goubault du CEA-List pour leurs conseils et explications sur Fluctuat.

Références

[1] Ali Ayad and Claude Marché. Multi-prover verification of floating-point programs. In *IJCAR*, volume 6173 of *LNCS*, pages 127–141. Springer, 2010.

[2] Sylvie Boldo and Jean-Christophe Filliâtre. Formal verification of floating-point programs. In *18th IEEE Symposium on Computer Arithmetic*, pages 187–194. IEEE, 2007.

[3] Bernard Botella, Arnaud Gotlieb, and Claude Michel. Symbolic execution of floating-point computations. *Software Testing, Verification and Reliability*, 16(2) :97–121, 2006.

[4] Angelo Brillout, Daniel Kroening, and Thomas Wahl. Mixed abstractions for floating-point arithmetic. In *9th International Conference on Formal Methods in Computer-Aided Design*, pages 69–76. IEEE, 2009.

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

[6] Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *6th ACM Symposium on Principles of Programming Languages*, pages 269–282, 1979.

[7] Patrick Cousot, Radhia Cousot, Jérôme Feret, Antoine Miné, Laurent Mauborgne, David Monniaux, and Xavier Rival. Varieties of static analyzers : A comparison with ASTRÉE. In *1st Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering*, pages 3–20. IEEE, 2007.

[8] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems*, 13(4) :451–490, 1991.

[9] Florent de Dinechin, Christoph Quirin Lauter, and Guillaume Melquiond. Assisted verification of elementary functions using Gappa. In *ACM Symposium on Applied Computing*, pages 1318–1322. ACM, 2006.

[10] David Delmas, Eric Goubault, Sylvie Putot, Jean Souyris, Karim Tekkal, and Franck Védrine. Towards an industrial use of fluctuat on safety-critical avionics software. In *FMICS*, volume 5825 of *LNCS*, pages 53–69. Springer, 2009.

[11] Khalil Ghorbal, Eric Goubault, and Sylvie Putot. A logical product approach to zonotope intersection. In *CAV*, volume 6174 of *LNCS*, pages 212–226. Springer, 2010.

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

[13] Eric Goubault and Sylvie Putot. Static analysis of numerical algorithms. In *SAS*, volume 4134 of *LNCS*, pages 18–34. Springer, 2006.

[14] Eric Goubault and Sylvie Putot. Static analysis of finite precision computations. In *VMCAI*, volume 6538 of *LNCS*, pages 232–247. Springer, 2011.

[15] Laurent Granvilliers and Frédéric Benhamou. Algorithm 852 : Realpaver : an interval solver using constraint satisfaction techniques. *ACM Transactions on Mathematical Software*, 32(1) :138–156, 2006.

[16] John Harrison. A machine-checked theory of floating-point arithmetic. In *TPHOLs*, volume 1690 of *LNCS*, pages 113–130. Springer-Verlag, 1999.

[17] Yahia Lebbah, Claude Michel, and Michel Rueher. A rigorous global filtering algorithm for quadratic constraints. *Constraints*, 10(1) :47–65, 2005.

[18] Olivier Lhomme. Consistency techniques for numeric CSPs. In *13th International Joint Conference on Artificial Intelligence*, pages 232–238, 1993.

[19] Bruno Marre and Claude Michel. Improving the floating point addition and subtraction constraints. In *CP*, volume 6308 of *LNCS*, pages 360–367. Springer, 2010.

[20] Claude Michel. Exact projection functions for floating-point number constraints. In *7th International Symposium on Artificial Intelligence and Mathematics*, 2002. `http://rutcor.rutgers.edu/~amai/aimath02/PAPERS/21.ps`.

[21] David Monniaux. The pitfalls of verifying floating-point computations. *ACM Transactions on Programming Languages and Systems*, 30(3) :12 :1–12 :41, 2008.

[22] Siegfried M. Rump. Verification methods : Rigorous results using floating-point arithmetic. *Acta Numerica*, 19 :287–449, 2010.

[23] The GSL Team. *GNU Scientific Library Reference Manual*, 1.14 edition, 2010. `http://www.gnu.org/software/gsl/`.