Résolution de contraintes sur les flottants

V3F: Validation et Vérification de logiciel avec calcul en Virgule Flottante

Claude Michel

I3S (UNSA-CNRS)

Partenaires

Partenaires du projet:

- **LIFC** - CNRS - INRIA - Bruno Legeard
- **I3S** - CNRS - INRIA - Michel Rueher
- **IRISA** - CNRS - INRIA - Thierry Jéron
- **LIST** - CEA - Bruno Marre

Savoir faire

- validation et la vérification du logiciel
- programmation par contraintes

Objectifs du projet (1)

Objectif : *fournir des outils de validation et vérification du logiciel qui permettent de traiter la représentation des réels par des flottants.*

- spécification $\longrightarrow \mathbb{IR}$, $\mathbb{IF}$
- programme $\longrightarrow \mathbb{IF}$

$\longrightarrow$ *beaucoup de questions . . .*

peu de réponses :-(

Objectifs du projet (2)

Plus immédiats

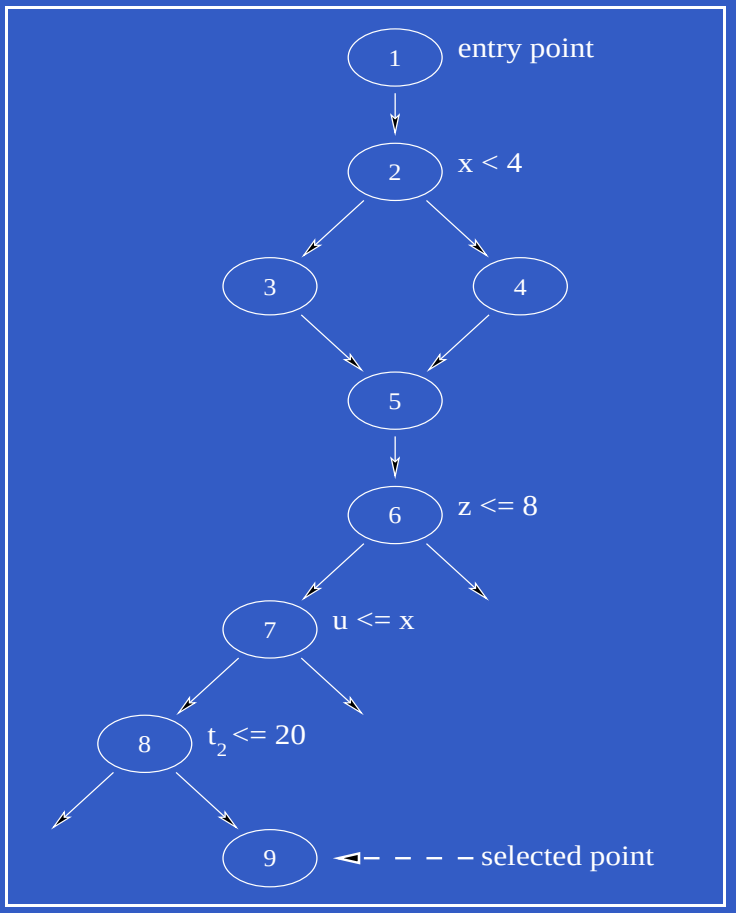
- développer des techniques de *résolution de contraintes* spécifi ques aux *nombres flottants*,

- étudier les *possibilités d'intégration* de ces techniques
  - dans des applications de vérifi cation de modèles formels de spécifi cation,
  - de génération automatique de tests,
  - ou encore, de vérifi cation statique de code.

Pourquoi des contraintes sur les flottants ?

Exemple: génération automatique de jeux de tests

```
float foo(float x, float y)
    float z, t, u ;
1.  { z = x * y ;
2.    t = 2.1 * x ;
3.    if (x < 4)
4.      u = 9.5 ;
      else
5.      u = 2.1 ;
6.    if (z ≤ 8)
7.    {  if (u ≤ x)
8.         {  t = t − y ;
9.            if (t ≤ 20)
10.             { ...
```



Génération de jeux de tests avec contraintes

```
float foo(float x, float y)
    float z, t, u ;
1.  { z = x * y ;
2.    t = 2.1 * x ;
3.    if (x < 4)
4.      u = 9.5 ;
      else
5.      u = 2.1 ;
6.    if (z ≤ 8)
7.      {  if (u ≤ x)
8.           {  t = t − y ;
9.              if (t ≤ 20)
10.                { ...
```

⇨

Système de contraintes équivalent :

$$\sigma_1 = (x, y, z, t_1, t_2, u \in (-\infty..\infty)) \wedge$$
$$(z = x * y) \wedge (t_1 = 2 * x) \wedge$$
$$(z \leq 8) \wedge (u \leq x) \wedge$$
$$(t_2 = t_1 - y) \wedge (t_2 \leq 20)$$

$+$ contraintes d' "entailment" (`if`, `while`, ...)

Solutions = n-uplet de *flottants*

ex. solution de σ_1: $x = 5.7, y = 0.5$

Motivations

- La résolution de contraintes sur les fbttants *dépend fortement de l'environnement* :
  - mode d'arrondi
  - unité de calcul en virgule fbttante
  - bibliothèque mathématique
  - ordre d'évaluation
  - ...

↪ Les techniques de filtrage sur les réels *peuvent supprimer des solutions* sur $\mathbb{F}$.

Exemple

Équation : $16.1 = 16.1 + x$ (1)

- solution de (1) sur les réels : 0
- domaine de x avec `PROLOG IV`
  $\longrightarrow x = 0$ (sur $\mathbb{R}$)
- solutions de (1) sur les fbttants * :
  $[-1.77635683940025046e{-}15, 1.77635683940025046e{-}15]$

(*) SPARC station, avec des `double` et la *libm*

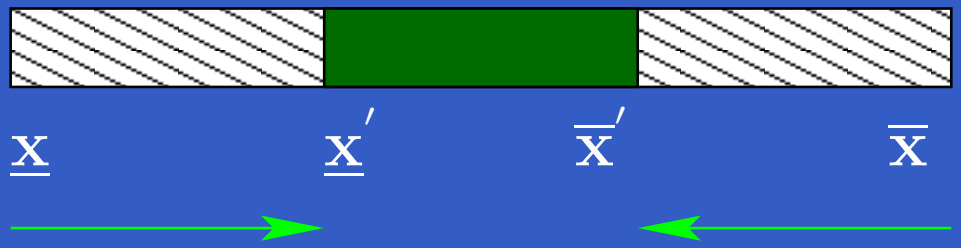
Résolution par Contraintes

Filtrage + *Recherche*

Réduction
des domaines

Choix de variables
et de valeurs
(heuristiques)

Filtrage par $2B$-consistence



$\rightarrow$ réduction du domaine $[\underline{\mathbf{x}}, \overline{\mathbf{x}}]$ de x à $[\underline{\mathbf{x}}', \overline{\mathbf{x}}']$ en considérant la contrainte n-aire

$$c(x_1, \ldots, x_{k-1}, x_k, x_{k+1}, \ldots, x_n)$$

Problèmes des techniques de filtrage sur $\mathbb{R}$

- **Fonctions de projections inverses des filtrages par $2B$**

  $16.1 = 16.1 + x$

  $\rightarrow$ fonction de projection : $x = 16.1 - 16.1$ (=0)

- **Propriétés non valides (ex: Méthode de Newton)**

  $\mathbf{f}(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \mathbf{x} + \mathbf{y} + \mathbf{z}$

  $\mathbf{x} = [-1, 1], \quad \mathbf{y} = [16.1, 16.1], \ \mathbf{z} = [-16.1, -16.1]$

  Itération d'un Newton par intervalles :

  $\mathbf{x} := \mathbf{x} \cap (m(\mathbf{x}) - \frac{f(m(\mathbf{x}),\mathbf{y},\mathbf{z})}{\frac{\partial f}{\partial x}(\mathbf{x},\mathbf{y},\mathbf{z})}) \rightarrow \mathbf{x} = [0, 0]$

CSPs sur $\mathbb{F}$

- **FCSP** (fbating-point constraint system) $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$:
  - un ensemble de *variables* $\mathcal{X} = \{x_1, ..., x_n\}$
  - un ensemble $\mathcal{D} = \{D_1, ..., D_n\}$ de *domaines* (D_i est un ensemble de valeurs de $\mathbb{F}$ possibles pour x_i)
  - un ensemble $\mathcal{C}$ de *contraintes*
- Une *contrainte* c sur l'ensemble ordonné de variables $X(c) = (x_1, ..., x_r)$ est un sous ensemble du produit Cartésien $(D_1 \times ... \times D_r)$ qui spécifi e les combinaisons *autorisées* de valeurs pour les variables $(x_1, ..., x_r)$.

L'arithmétique des intervalles est conservative sur $\mathbb{F}$

- **Propriété 1 :**

  $\mathbf{f}(\mathbf{x_1}, \ldots, \mathbf{x_n})$: extension naturelle aux intervalles de f

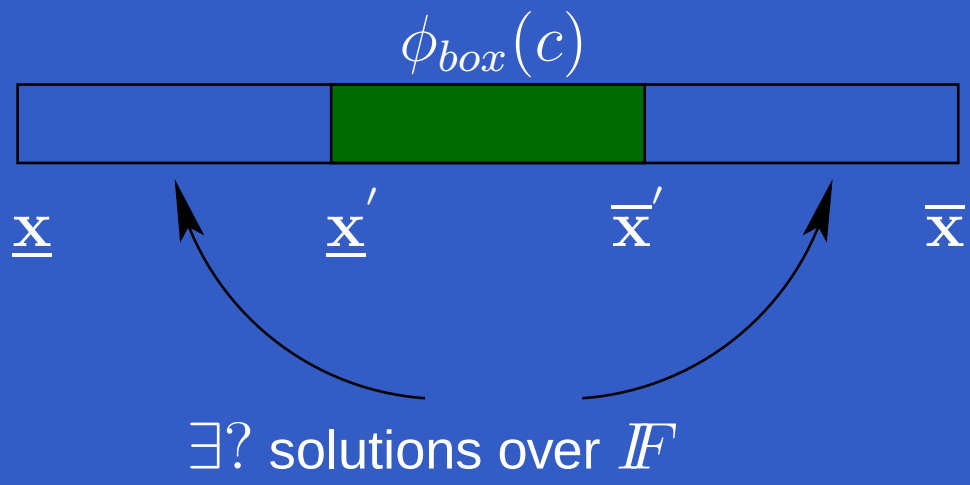
  $eval(f(x_1, ..., x_n), r)$: évaluation de f avec le mode d'arrondi r

  $$\forall r \in \{-\infty, +\infty, 0, \text{near}\},$$
  $$eval(f(x_1, ..., x_n), r) \in \mathbf{f}(\mathbf{x_1}, \ldots, \mathbf{x_n})$$

- **Composition de fonctions :** propriété 1 toujours vraie

  (*si les calculs de* f *et de* $\mathbf{f}$ *sont effectués en évaluant la même séquence d'opérations*)

Réduction des domaines sur $\mathbb{F}$: 1ère approche



$\longrightarrow$ évaluation directe de c avec $\mathbf{x} = [\underline{\mathbf{x}}, \underline{\mathbf{x}}']$

si $0 \in eval(c([\underline{\mathbf{x}}, \underline{\mathbf{x}}']))$

$\rightsquigarrow$ coupe sur $[\underline{\mathbf{x}}, \underline{\mathbf{x}}']$

La FP-$2B$-consistance

- Une contrainte n-aire c est FP-$2B$-**consistent** si :
  $\forall x_i \in X(c)$,

  $D_i = \square\{v_i \in D_i$ tel que il existe un v_k dans chaque D_k $(k \neq i)$
  pour lequel $c(v_1, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_n)$ est vrai
  avec $Round(r)\}$

  (D_i : ensemble de flottants)

  Un $FCSP$ est FP-$2B$-consistant,
  ssi toutes ses contraintes sont FP-$2B$-consistantes.

Fonctions de projections

- Soit la contrainte $c : \mathbf{x}_1 + \mathbf{x}_2 = \mathbf{x}_3$
  $\longrightarrow 3$ fonctions de projections

$$\begin{cases} f_1 : \mathbf{x}_1 \leftarrow \mathbf{x}_3 - \mathbf{x}_2 & \text{(projection inverse)} \\ f_2 : \mathbf{x}_2 \leftarrow \mathbf{x}_3 - \mathbf{x}_1 & \text{(projection inverse)} \\ f_3 : \mathbf{x}_3 \leftarrow \mathbf{x}_1 + \mathbf{x}_2 & \text{(projection directe)} \end{cases}$$

- Calcul du domaine de x_1 :

$$\mathbf{x}'_1 \leftarrow \mathbf{x}_1 \cap f_1(\mathbf{x}_1)$$

Calcul de la projection directe

- L'arithmétique des intervalles est conservative

$$\square\{y \in \mathcal{F}_\kappa \mid y = \hat{f}_r(x), x \in \mathbf{x}, \forall r \in \{-\infty, +\infty, 0, \text{near}\}\}$$
$$= [\hat{f}_{-\infty}(\underline{\mathbf{x}}), \hat{f}_{+\infty}(\overline{\mathbf{x}})]$$

(f : croissante)

($\hat{f}_r$: correctement arrondie, i.e., $\hat{f}_r(x) = \Theta_r(f(x))$)

- Lorsque r est connu :

$$\forall r \in \{-\infty, +\infty, 0, \text{near}\}$$
$$\square\{y \in \mathcal{F}_\kappa \mid y = \hat{f}_r(x), x \in \mathbf{x}\} = [\hat{f}_r(\underline{\mathbf{x}}), \hat{f}_r(\overline{\mathbf{x}})]$$

Calcul de la projection inverse

- **L'inverse de** $\hat{f}_{+\infty}(x) = y$

$$\square\{x \in \mathcal{F}_\kappa \mid \hat{f}_{+\infty}(x) \in \mathbf{y}\} = [\Theta^+(f^{-1}(\underline{\mathbf{y}}^-)), \Theta_{-\infty}(f^{-1}(\overline{\mathbf{y}}))]$$

$$\text{avec } \Theta^+(x) = \begin{cases} x^+ & \text{ssi } x \in \mathbb{F}, \\ \Theta_{+\infty}(x) & sinon. \end{cases}$$

! f croissante et $\hat{f}_r$ correctement arrondie

- Résultats similaires $\forall r \in \{-\infty, 0, \text{near}\}$ et r inconnu.

Problèmes d'implémentation

- Calcul de la projection inverse
  - $\forall r \in \{-\infty, +\infty, 0\} \rightarrow$ calculable avec IEEE 754
  - $r =$ near $\rightarrow$ nécessite $\mathbb{F}_{k+1}$
- Extensions
  - ∞ et l'arithmétique des zéros signés
  - $max(\mathbb{F})$ et $min(\mathbb{F})$ quand $r \in \{-\infty, +\infty, 0\}$

Implémentation du solveur

Objectif: un solveur *souple*

- 2 bibliothèques
  - libfpi (en C): couche intervalles
    (fonctions de projections, accès FPU, . . .)
  - libfpc (en C++): couche solveur
    (traitement des expressions . . .)
- intégrables dans des environnements
  - dynamiques (ex.: Prolog (sicstus, eclipse, . . .))
  - statiques (ex.: Ilog solver)

Choix d’environnement

Plateforme ciblée: Intel/Linux

- FPU (IEEE 754) répandu et étudié
- sources libre et accessible
- GCC/libm

Possibilité d’évolution vers Windows

Couverture (safe)

- 3 types de fbttants (`float`, `double`, `long double`)
- les différents modes d'arrondi
- les différents modes de précision
- l'affectation : =
- les opérations binaire de base : +, -, *, /
- l'opérateur unaire : -
- les opérateurs de comparaison : ==, !=, <, >, <=, >=.
- la racine carrée : sqrtf, sqrt, sqrtl
- la valeur absolue : fabsf, fabs, fabsl

Couverture (unsafe)

- Tentative de couverture de la `libm`
  - l'exponentielle et le logarithme : expf, exp, expl, logf, log, logl
  - les fonctions trigonométriques : sinf, sin, sinl, cosf, cos, cosl, tanf, tan, tanl asinf, asin, asinl, acosf, acos, acosl, atanf, atan, atanl
  - la racine cubique : cbrtf, cbrt, cbrtl
- ajout de fbttants aux bornes
- testé, mais sans garantie

Exemple d'utilisation de FPC

```
#include <fpc.h>
int main(void) {
    int nbsols = 0;
    Fpc_Csp CSP;
    Fpc_Variable x(CSP, "x", FPC_LDOUBLE);
    Fpc_Model model(CSP);

    model.add(x*x == 4.0);

    model.extract();
    Fpc_Solver solver(model);
    while (solver.searchNext()) {
      ...
```

Exemple de la racine cubique

```
int solve(double a, double b, double c) {
  double q, r, delta;
  ...
  q = (3.0*b - a*a)/3.0;
  r = (2.*a*a*a - 9.0*a*b + 27.0*c)/27.0;
  delta = q*q*q/27.0 + r*r/4.0;
  if(fabs(delta) < 1.0e-40) {
    /** point 1 **/
    ...
  } else {
    ...
```

Système de contraintes équivalent

Contraintes pour atteindre le `point 1`:

$$Q = (3.0 \times B - A \times A)/3.0$$
$$\wedge\ R = (2.0 \times A \times A \times A - 9.0 \times A \times B + 27.0 \times C)/27.0$$
$$\wedge\ DELTA = (Q \times Q \times Q/27.0 + R \times R/4.0)$$
$$\wedge\ abs(DELTA) < 1.0e - 40$$

- DeClic $\rightarrow$ 35 valeurs d'entrée (10 fausses !)
- FPC $\rightarrow$ 337 valeurs d'entrée

Conclusion

- Apports
  - traitements correct des contraintes sur $\mathbb{F}$
  - implémentation souple avec fonctionnalités étendues
- Travaux en cours
  - tests des bibliothèques
  - intégration avec d'autres solveurs (Ilog Solver, . . .)
  - mise en œuvre à plus grande échelle