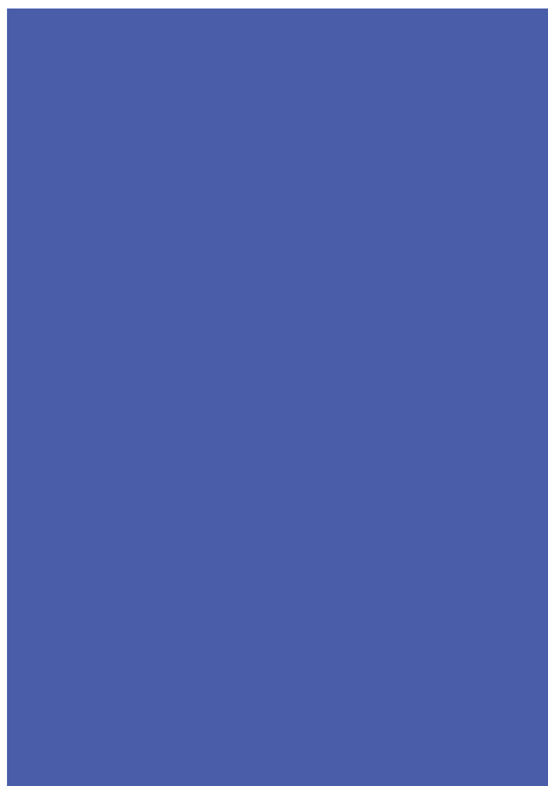
INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

École CEA-EDF-INRIA
Calcul Numérique Certifié

Guillaume Hanrot — Vincent Lefèvre — Sylvie Putot — Paul Zimmermann

Octobre 2007

Thème SYM



École CEA-EDF-INRIA
Calcul Numérique Certifié

Guillaume Hanrot, Vincent Lefèvre, Sylvie Putot, Paul Zimmermann

Thème SYM — Systèmes symboliques
Projet CACAO

— Octobre 2007 —

Résumé : Ce document rassemble les supports de cours de l'école « Calcul Numérique Certifié » organisée au Centre de Recherche INRIA Nancy - Grand Est les 25 et 26 octobre 2007.

Mots-clés : calcul flottant, norme IEEE 754, analyseur FLUCTUAT, bibliothèque MPFR

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy (France)
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

CEA-EDF-INRIA School
Certified Numerical Computation

Abstract: This document includes the slides and other support media from the school on "Certified Numerical Computation" organized at INRIA Nancy - Grand Est on October 25 and 26, 2007.

Key-words: floating-point computation, IEEE 754 standard, FLUCTUAT analysis tool, MPFR library

Le but de cette école est de sensibiliser aux problèmes posés par les calculs numériques via l'utilisation d'arithmétique flottante (erreur d'arrondi, précision limitée, débordement de capacité, etc.) et de proposer quelques outils innovants sur ce thème, avec en particulier une formation à la bibliothèque MPFR de calcul flottant en précision arbitraire.

Contrairement aux calculs entiers, les calculs sur des nombres flottants sont par nature inexacts. Il est donc primordial de contrôler a priori ou a posteriori les erreurs d'arrondi effectuées dans un calcul donné, et cela indépendamment des possibles erreurs de méthode dues au schéma numérique utilisé (maillage, approximation polynomiale, troncature). Nous présenterons plusieurs solutions permettant de contrôler ces erreurs d'arrondi (arithmétique d'intervalles, précision arbitraire, méthodes stochastiques, etc.). Parmi ces outils, nous mettrons l'accent sur la bibliothèque MPFR.

L'objectif est qu'à la fin de l'école, les participants aient acquis le b-a-ba leur permettant de maîtriser par eux-mêmes MPFR, pour résoudre leurs vrais problèmes, via des travaux pratiques sur machine, pour une prise en main effective de cette bibliothèque, et la résolution de problèmes jouets. L'école comprend aussi une session ouverte, dont l'objectif est d'utiliser MPFR sur de vrais problèmes rencontrés par les participants dans leur travail d'ingénieur ou de chercheur. (Les participants sont vivement invités à apporter de tels problèmes, éventuellement communiqués à l'avance aux organisateurs.)

Les organisateurs tiennent à remercier les personnes suivantes, qui ont contribué au succès de cette école. Luc Bouganim, membre du comité directeur des écoles CEA-EDF-INRIA, a constamment soutenu notre projet, bien qu'il sorte quelque peu de l'ordinaire. Julie Paul (INRIA Paris-Rocquencourt) s'est chargée de l'envoi des affiches. Anne-Lise Charbonnier, responsable Cours et Colloques de l'INRIA Nancy - Grand Est, a accepté de prendre en charge l'organisation locale. L'installation des machines de la salle de travaux dirigés a pu se faire grâce à Lionel Maurice, ainsi qu'à Emmanuel Thomé qui a aussi contribué à la page de couverture de ce document.

Nancy, octobre 2007

Arithmétique flottante : principes, méthodes, outils

G. Hanrot

INRIA Lorraine - LORIA

25 octobre 2007

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
INRIA
centre de recherche NANCY - GRAND-EST

Plan

1. Nombres réels et ordinateurs
2. Un point de vue atomique : IEEE754
3. De l'atomique aux calculs complexes : stratégies
4. Outils logiciels

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
INRIA
centre de recherche NANCY - GRAND-EST

Nombres réels et ordinateurs

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
INRIA
centre de recherche NANCY - GRAND-EST

Nombres réels et ordinateurs

- Problème intrinsèque : types informatiques *finis*.
- Typiquement, au plus 64 bits = 2^{64} valeurs ;
- "Faire rentrer" de l'infini dans du fini : modèle des nombres réels ;
- Trois aspects :
  - Choisir un sous-ensemble fini de l'ensemble à représenter (ici réels) ;
  - Définir des opérations *sur ce sous-ensemble* ;
  - Évaluer l'écart entre ce "modèle" et la réalité.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
INRIA
centre de recherche NANCY - GRAND-EST

Nombres réels et ordinateurs (2)

- Sur les entiers, facile : entiers modulo 2^w ;
  - $\hookrightarrow$ opérations “naturelles” ;
  - (presque) compatible avec les opérations entières
  - pas d'« amplification » de l'erreur
- Sur les réels, difficile ;
  - pour les raisons inverses...
- Notion de *précision p* $\rightarrow$ nombre représentable en précision p, $\mathcal{R}_p$;

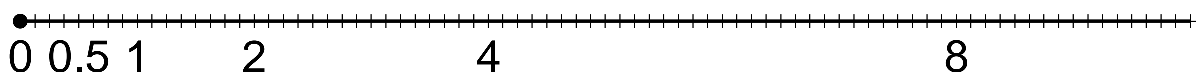
Nombres réels et ordinateurs
Un point de vue atomique : IEEE754
De l'atomique aux calculs complexes : stratégies
Outils logiciels

Nombres réels et ordinateurs (3)

- Idée 1 : précision *absolue*.

$$R_{p,q} = \{\text{nombres} \leq 2^q \text{ avec } p \text{ chiffres binaires après la virgule}\}.$$

- Chaque nombre correspond à un intervalle réel de taille $\Delta x \approx 2^{-p}$; l'erreur commise est *constante*.
- Beaucoup de chiffres significatifs quand x grand, très peu quand x petit.
- Allure de $R_{p,q}$:



virgule fixe.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE INRIA centre de recherche NANCY - GRAND-EST
G. Hanrot Arithmétique flottante : principes, méthodes, outils

Nombres réels et ordinateurs (4)

- Idée 2 : précision relative : augmente quand le nombre diminue.

$$R_{p,q} = \{(x \cdot 2^{-p}) \cdot 2^e, |e| \leq q, x \in [2^{p-1}, 2^p[\text{ entier}\}.$$

- x = mantisse, e = exposant.
- Toujours p chiffres après la virgule, mais la virgule “flotte” : elle se déplace en tête du nombre (position e).
- $x \neq 0 \leftrightarrow$ intervalle de taille Δx avec $\Delta x / x \approx 2^{-p}$. L'erreur commise est *relative* ;
- Nombre de chiffres significatifs constant ;
- Allure de $R_{p,q}$:

virgule flottante.

Un point de vue atomique :

la norme IEEE-754

La norme IEEE-754 (1)

- Virgule flottante = choix de fait sauf applications ciblées ;
- Nécessité d'une *sémantique précise* : fiabilité ;
- Nécessité d'*uniformiser les choix* : portabilité.

Philosophie **atomique** :

- Ensemble des nombres représentables ;
- Opérations sur cet ensemble, sémantique.

La norme IEEE754 se situe au niveau de l'opération individuelle.

- À chaque nouvelle opération, opérandes considérés comme *exacts*.
- Pas de notion de "propagation d'erreur" ;
- $\Rightarrow$ Utiliser une des techniques de la partie 3.

La norme IEEE-754 (2) - nombres représentables

- $\mathcal{R}_p \approx \{\pm(m \cdot 2^{-p}) \cdot 2^e, m \in [2^{p-1}, 2^p[, e \in [e_{\min}, e_{\max}]\} \cup \{0\}$.
- Types standard :

type	w	p	$\approx$ dd	$e_{\max}$	max. $\approx$
`float`	32	24	7	128	$3.40 \cdot 10^{38}$
`double`	64	53	16	1024	$1.80 \cdot 10^{308}$
`extended`	≥ 80	≥ 64	24	≥ 16384	$1.19 \cdot 10^{4932}$
`quad` $\notin$ IEEE754	128	113	34	16384	$1.19 \cdot 10^{4932}$

Multiprécision : w, p, dd quelconques, potentiellement $[e_{\min}, e_{\max}]$ aussi.

La norme IEEE-754 (3) - arrondi

- Un nombre représentable représente tout un intervalle de $\mathbb{R}$ qui le contient ; lequel ?
- Inversement : soit $x \in \mathbb{R}$, $\exists y \in \mathcal{R}_p$ qui le représente ?

Nombre/intervalle sont déterminés par arrondi

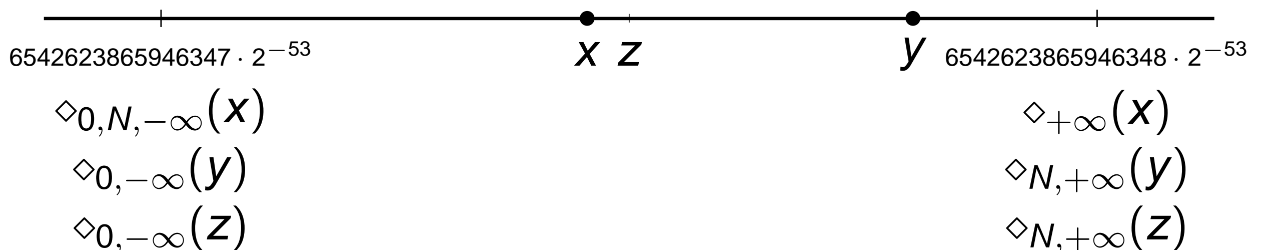
IEEE-754 définit 4 types ("modes") d'arrondi :

- vers $\pm\infty$: $\diamond_{+\infty}, \diamond_{-\infty}$;
- vers 0 : $\diamond_0$;
- au plus proche : $\diamond_N$.

Nombres réels et ordinateurs
Un point de vue atomique : IEEE754
De l'atomique aux calculs complexes : stratégies
Outils logiciels

La norme IEEE-754 (4) - arrondi

- Arrondi pair : pour $\diamond_N$, en cas d'équidistance entre deux éléments de $\mathcal{R}_p$, on choisit celui de mantisse paire.



INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE — INRIA — centre de recherche NANCY - GRAND-EST
G. Hanrot — Arithmétique flottante : principes, méthodes, outils

La norme IEEE-754 (5) - sémantique des opérations

- Guidée par le principe d'atomicité ;
- On suppose les opérandes **exacts**, x, $y \in \mathbb{R}$.
- Le résultat de $f(x)$, de x op y en flottant est

$$\diamond(f(x)), \diamond(x \operatorname{op} y) \in \mathcal{R}_p.$$

- "évaluer f, ou op en précision infinie, puis arrondir" ;
- IEEE754 **impose** cette sémantique pour $\{+, -, \times, /, \sqrt{}\}$;
- Rarement respecté par les `libm` dans les implantations pour d'autres f... sauf `CRlibm`, cf. infra.

La norme IEEE-754 (6) - Exemple

- Calcul de $x \cdot y$?
- $x = 7774910112710892 \cdot 2^{-53}$
- $y = 8934387281065303 \cdot 2^{-52}$
- $xy = 69464058022430194870055011380276 \cdot 2^{-105} \approx 1.712421240371849084352509952$
- Exposant = 1, et

$$2^{53-1} xy \approx 7712059660039982.097463063953$$

- $xy = 7712059660039982 \cdot 2^{-52}$ ou $7712059660039983 \cdot 2^{-52}$ selon le mode d'arrondi.

La norme IEEE-754 (7) - exceptions et valeurs exceptionnelles

- Nécessité d'ajouter des *valeurs exceptionnelles* :
  0/0, 1/0, log(0), $\sqrt{-1}$, etc.
- `Inf`, `-Inf`, `NaN`
- Permet parfois la poursuite du calcul malgré l'erreur ; eg. $1/\texttt{Inf}$, arctan(`Inf`), `NaN` $==$ x, `hypot`(`NaN`, $\pm$ `Inf`).
- **Rem :** 0 a un signe, mais 0 == -0.
- Flags : `INVALID`, `DIVIDE_BY_ZERO`, `OVERFLOW`, `UNDERFLOW`, `INEXACT`.

Nombres réels et ordinateurs
Un point de vue atomique : IEEE754
De l'atomique aux calculs complexes : stratégies
Outils logiciels

La norme IEEE-754 (8) - limites

- IEEE-754 est pensé au niveau **atomique**.
- **Anti-théorème**. La plupart des identités algébriques classiques sont fausses.
- Ex. $(x+y)+z \neq x+(y+z)$ en général.
- Un piège grave classique : avec la norme telle que décrite jusqu'ici,

```
if (x <> y)  z = 1/(x - y);
```

  peut produire une exception `DIVIDE_BY_ZERO` ($x =$ `FLOAT_MIN`, y immédiatement supérieur à x, $\diamond_0$). Cf. transparent (10).

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE INRIA centre de recherche NANCY - GRAND-EST
G. Hanrot Arithmétique flottante : principes, méthodes, outils

La norme IEEE-754 (9) - pièges

- Csq. L'ordre des opérations, les formules choisies pour évaluer **jouent un rôle critique** dans la qualité du résultat.
- Règle de base : **proscrire** les soustractions de nombres proches, qui provoquent des pertes **colossales** de précision : si $x \approx 2^{-54}$,

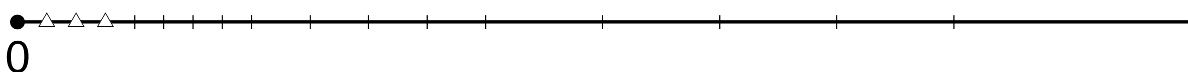
$$(x+1)-1=0 \neq x = x+(1-1)$$

Réf : D. Goldberg, *What every computer scientist should know about floating-point arithmetic.*

Nombres réels et ordinateurs
Un point de vue atomique : IEEE754
De l'atomique aux calculs complexes : stratégies
Outils logiciels

La norme IEEE-754 (10) - dénormalisés

- Principe : on veut éviter le phénomène $x \neq y$ et $x - y = 0$.
- Phénomène inhérent à la virgule flottante ;
- $\Rightarrow$ basculer en virgule fixe au voisinage de 0 !
- Ajouter $\{\pm x \cdot 2^{-p} \cdot 2^{e_{\min}}, x \in [0, 2^{p-1}[\}$
- dessin :



- Longtemps controversés, mais indispensables.
- Compliquent souvent l'analyse...

La norme IEEE-754 (11) - évolutions

- Révision en cours (presque finie) ;
- Principales nouveautés :
  - Introduction de formats décimaux ;
  - Une annexe (non normative) recommandant la prise en compte de l'arrondi exact pour les fonctions spéciales ;
  - Ce dernier point est implanté dans `CRlibm` ;
    - bibliothèque math. avec arrondi correct (projet Arénaire, ENS-Lyon) ;
    - LGPL ;
    - algorithmes spécifiés et prouvés ;
    - efficacité comparable aux meilleures `libm`.

De l'atomique aux calculs complexes : stratégies

Problématique

- IEEE-754 permet de contrôler l'erreur au niveau de l'opération ;
- On veut contrôler l'ensemble d'un algorithme numérique ;
- $\Rightarrow$ "recoller" les estimations d'erreur pour obtenir une estimation globale ;
  - Manuellement : fastidieux, difficile, sujet à erreurs ;
  - Automatiquement : quelques pistes
    - Solutions heuristiques : la foi ; précision multiple ; arithmétique stochastique.
    - Solutions exactes : Analyse par intervalles, RealRAM, méthodes formelles.

Analyse mathématique

- Pour chaque opération, on a une estimation

$$\left|\frac{\Delta(x \,\mathrm{op}\, y)}{x \,\mathrm{op}\, y}\right| \leq \epsilon$$

  si x op y non dénormalisé ;
- Écrire la chaîne d'inégalités, combiner ;
- Traiter les cas dénormalisés.
- Difficile, technique : on découvre souvent *a posteriori* de nombreuses erreurs.
- Méthodes formelles : prouver cela à l'aide d'un assistant de preuve. Fastidieux, automatisation difficile.

Précision multiple

- $p = 53, 64, 113$ peut être insuffisant pour certaines applications ;
  - Parce qu'on veut une très grande précision ;
  - Parce qu'on a identifié une section comme très instable numériquement ;
- Pour p "assez grand", on aura toujours une bonne approximation ;
- Stratégie heuristique : augmenter la précision jusqu'à ce que le résultat se stabilise ;
- Pas une solution en soi, plus une façon de repousser le problème.
- Avantages : facile à déployer.
- Inconvénients : coûteux en temps de calcul, pas garanti.

Analyse par intervalles

- Variable = *intervalle* ;
- Règles de calcul sur les intervalles :

$$\begin{aligned} [x_0, x_1] + [y_0, y_1] &= [\diamond_{-\infty}(x_0 + y_0), \diamond_{+\infty}(x_1 + y_1)] \\ [x_0, x_1] - [y_0, y_1] &= [\diamond_{-\infty}(x_0 - y_1), \diamond_{+\infty}(x_1 - y_0)] \end{aligned}$$

- Multiplication plus difficile (exercice) ;
- Division problématique : x/y pas défini si $0 \in y$;
- Avantages : résultats *rigoureux* si les opérations de base obéissent à la sémantique IEEE754.
- Inconvénients :
  - purement dynamique ;
  - atomicité $\Rightarrow$ importantes pertes de précision "décorrélation" : $x \in [0, 1] \Rightarrow x(1 - x) \in [0, 1] \times [0, 1] = [0, 1]$.

RealRAM

- Réalisation concrète d'un modèle de calcul théorique = **machine à calculer avec des vrais nombres réels, sans comparaison ni test d'égalité.**
- Variable = intervalle + *expression*.
- Vision paresseuse : l'intervalle est raffiné "à la demande", par réévaluation de l'expression en précision supérieure ;
- Compromis entre multiprécision et arithmétique d'intervalles.
- Avantages : garanti, transparent (gestion de la précision automatique et paresseuse) ;
- Inconvénient : souvent moins efficace en moyenne que multiprécision bien utilisée ; peu adapté (stockage de l'expression) à un long calcul.

Arithmétique stochastique

- Modèle probabiliste de l'arrondi ;
- Un calcul donné est effectué N fois ;
- Chaque arrondi est effectué aléatoirement ;
- $\hookrightarrow$ N résultats différents du même calcul ;
- $\hookrightarrow$ estimation de la distribution et des valeurs extrêmes possibles pour une suite d'opérations.
  - Avantages : analyse automatique, détection des instabilités numériques ;
  - Inconvénients : pas rigoureux, dynamique.

Méthodes formelles

- Constat : les preuves "manuelles" sont souvent incorrectes ;
- Utilisation d'un "assistant de preuve" (Coq, PVS, Hol, etc.) ;
- Formalisation du modèle IEEE-754 de calcul flottant ;
- Preuves d'estimations d'erreur et de corrections d'algorithme ;
- Avantage : complètement rigoureux ;
- Inconvénients ; peu automatisé, très lourd, passage à l'échelle ?

Nombres réels et ordinateurs
Un point de vue atomique : IEEE754
De l'atomique aux calculs complexes : stratégies
Outils logiciels

Outils logiciels

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE INRIA
centre de recherche NANCY - GRAND-EST
G. Hanrot Arithmétique flottante : principes, méthodes, outils

Précision multiple – Haut niveau

Panorama :

- GP : multiprécision efficace. Sémantique “Read the code Luke”. “Cache” éventuellement des chiffres supplémentaires. [GPL]
- Maple : interne en décimal (IEEE854), peu efficace. [Commercial]
- Mathematica : concept de “significance arithmetic”. [Commercial]

Nombres réels et ordinateurs
Un point de vue atomique : IEEE754
De l'atomique aux calculs complexes : stratégies
Outils logiciels

Précision multiple – Bas niveau

- Pari (version bibliothèque de GP). Gestion mémoire délicate. [C, GPL]
- MPF : précision multiple de w, sémantique “à la GP/Pari”, peu de fonctions spéciales. [C, LGPL]
- MPFR : IEEE-754 compliant (au sens : si vous avez un contre-exemple, c'est bien un bug), **sauf** dénormalisés. [C, LGPL]
- NTL : IEEE-754 pour opérations et racine carrée, arrondi fidèle sinon, quelques fonctions spéciales [C++, GPL].

Autres outils

- Arithmétique d'intervalles :
  - `boost` [C++, licence “maison”, flottants natifs],
  - `MPFI` [C, LGPL, basé sur MPFR].
- RealRAM : iRRAM [C++, LGPL] ;
- Arithmétique stochastique : Cadna [Fortran, licence “maison”].

Deux exemples d'instabilité numérique

- Un exemple dû à Rump :

$$1335y^6/4 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 11y^8/2 + x/(2y)$$

  pour $x = 77617$, $y = 33096$ [“cancellation”].
- Un exemple dû à J-M. Muller : la suite

$$x_0 = 11/2, x_1 = 61/11, x_{n+1} = 111 - (1130 - 3000/x_{n-1})/x_n$$

  converge vers 6, mais de façon très instable :
  - la solution générale pour x_n est

$$x_n = \frac{\alpha \cdot 5^n + \beta \cdot 6^n + \gamma \cdot 100^n}{\alpha \cdot 5^{n-1} + \beta \cdot 6^{n-1} + \gamma \cdot 100^{n-1}}$$

  - les conditions initiales données correspondent à un cas où $\gamma = 0$, $\beta \neq 0$ et la limite est 6 ;
  - toute erreur numérique ramène à un cas où $\gamma \neq 0$, et la limite est 100.

Exemple de Rump en GP

[En GP/Pari]

```
rump(n) =
  {
    default(realprecision, n);
    x=77617.0; y=33096.0;
    1335*y^6/4+x^2*(11*x^2*y^2-y^6-121*y^4-2)\
    +11*y^8/2+x/(2*y)
  }
```


Nombres réels et ordinateurs
Un point de vue atomique : IEEE754
De l'atomique aux calculs complexes : stratégies
Outils logiciels

Résultats en GP

```
gp> rump(5)
%1 = -2.4759 E27
gp > rump(10)
%2 = 5.76460752 E17
gp > rump(15)
%3 = 5.76460752 E17
gp > rump(20)
%4 = 0.E8
gp > rump(25)
%5 = 0.E8
gp > rump(30)
%6 = -0.827396060
```

Exemple de Rump en MPFI

```
void rump(int p)
{
    mpfi_t x, x2, y, y2, y4, y6, y8, tmp, tmp2;
    mpfr_set_default_prec(p);
    mpfi_init_set_ui(x, 77617); mpfi_init_set_ui(y, 33096);
    mpfi_init(tmp); mpfi_init(tmp2); mpfi_init(y2); mpfi_init(y4);
    mpfi_init(y6); mpfi_init(y8); mpfi_init(x2);
    mpfi_sqr(y2, y); mpfi_sqr(y4, y2); mpfi_mul(y6, y4, y2); mpfi_sqr(y8, y4);
    mpfi_sqr(x2, x);
    mpfi_mul(tmp, y2, x2);
    mpfi_mul_ui(tmp, tmp, 11);
    mpfi_sub(tmp, tmp, y6); /* tmp <- 11x^2y^2-y^6 */
    mpfi_mul_ui(y4, y4, 121); mpfi_sub(tmp, tmp, y4);
    mpfi_sub_ui(tmp, tmp, 2); mpfi_mul(tmp, tmp, x2);
    /* tmp<-x^2(tmp-121y^4-2) */
    mpfi_mul_ui(y6, y6, 1335); mpfi_div_2exp(y6, y6, 2); mpfi_add(tmp, tmp, y6);
    /*tmp<-tmp+1335y^6/4 */
    mpfi_mul_ui(y8, y8, 11); mpfi_div_2exp(y8, y8, 1); mpfi_add(tmp, tmp, y8);
    //tmp<-tmp+11y^8/2

    mpfi_div(x, x, y); mpfi_div_2exp(x, x, 1); mpfi_add(tmp, tmp, x);
    // tmp <- tmp + x/(2y)

    printf("%d ", p); mpfi_out_str(stdout, 10, 10, tmp); printf("\n");
    mpfi_clear(x); mpfi_clear(y); mpfi_clear(x2); mpfi_clear(y2);
    mpfi_clear(y4); mpfi_clear(y6); mpfi_clear(y8); mpfi_clear(tmp);
```


Résultats en MPFI

```
macaron $ for i in 20 40 60 80 100 120 121 122
140; do ./rump $i; done
20 [-1.318356625e32,1.318358172e32]
40 [-6.769984590e25,6.769984590e25]
60 [-5.534023223e19,4.611686019e19]
80 [-4.398046512e13,1.759218605e13]
100 [-8.388606828e6,8.388609173e6]
120 [-6.827396060,1.172603941]
121 [-2.827396060,1.172603941]
122 [-8.273960600e-1,-8.273960599e-1]
140 [-8.273960600e-1,-8.273960599e-1]
```

Exemple de Rump en iRRAM

Résultats en iRRAM

Exemple de Rump en Cadna

```
        program rump
        use cadna
        type (double_st) : : x, y, res
C
        call cadna_init(-1)
        x = 77617.d0
        y = 33096.d0
        res = 1335*y**6/4+x**2*(11*x**2*y**2-y**6-121*y**4-2)
        res = res + 11*y**8/2 + x/2/y
        write(*,*) str(res)
        call cadna_end()
C
        end
```


Nombres réels et ordinateurs
Un point de vue atomique : IEEE754
De l'atomique aux calculs complexes : stratégies
Outils logiciels

Résultats

```
-----------------------------------
CADNA software -- University P. et M. Curie -- LIP6
Self-validation detection : ON
Mathematical instabilities detection : ON
Branching instabilities detection : ON
Intrinsic instabilities detection : ON
Cancellation instabilities detection : ON
-----------------------------------
@.0
-----------------------------------
CADNA software -- University P. et M. Curie -- LIP6
There is 1 numerical instability
0 UNSTABLE DIVISION(S)
0 UNSTABLE POWER FUNCTION(S)
0 UNSTABLE MULTIPLICATION(S)
0 UNSTABLE BRANCHING(S)
0 UNSTABLE MATHEMATICAL FUNCTION(S)
0 UNSTABLE INTRINSIC FUNCTION(S)

1 UNSTABLE CANCELLATION(S)
```

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE INRIA centre de recherche NANCY - GRAND-EST
G. Hanrot Arithmétique flottante : principes, méthodes, outils

Exemple de Muller en iRRAM

```
void jmm_iRRAM(int count){
    REAL a= REAL(11)/2, b=REAL(61)/11, c;
    for (long i=0;i<count;i++ ) {
      cout << REAL(a) << " " << i <<"\n";
      c=111-(1130-3000/a)/b;
      a=b; b=c;
    }
    cout << REAL(a) << " " << count «"\n";
}
```


Nombres réels et ordinateurs
Un point de vue atomique : IEEE754
De l'atomique aux calculs complexes : stratégies
Outils logiciels

Résultats en iRRAM

Exemple de Muller en MPFI

```
void jmm_seq(int count)
{
    mpfi_t a, b, c; int i;
    mpfi_init_set_d(a, 5.5); mpfi_init_set_ui(b, 61);
    mpfi_init(c); mpfi_div_ui(b, b, 11);
    for(i = 1; i <= count; i++)
       {
         mpfi_ui_div(c, 3000, a); /* c <- 3000/a */
         mpfi_ui_sub(c, 1130, c); /* c <- 1130 - c */
         mpfi_div(c, c, b); /* c <- c / b */
         mpfi_ui_sub(c, 111, c); /* c <- 111 - c */
         printf("%d : ", i); mpfi_out_str(stdout, 10, 0, c);
         printf("\n");
         mpfi_set(a, b); mpfi_set(b, c); /* a <- b, b <- c */
       }
    mpfi_clear(a); mpfi_clear(b); mpfi_clear(c);
}
```


Nombres réels et ordinateurs
Un point de vue atomique : IEEE754
De l'atomique aux calculs complexes : stratégies
Outils logiciels

Les résultats

```
1 : [5.5901639344262150,1.0540983606557379e2]
2 : [5.6334310850437248,1.0536656891495628e2]
3 : [5.6746486205054793,1.0532535137949453e2]
4 : [5.7133290522982207,1.0528667094770178e2]
5 : [5.7491209182627898,1.0525087908173722e2]
6 : [5.7818108954496381,1.0521818910455037e2]
7 : [5.8113138054168587,1.0518868619458315e2]
8 : [5.8376491021732590,1.0516235089782675e2]
9 : [5.8608239892737401,1.0513917601072626e2]
10 : [5.8792016587649698,1.0512079834123504e2]
11 : [5.8621566008234822,1.0513784339917652e2]
12 : [5.2835069805909142,1.0571649301940909e2]
13 : [-6.0137461590588402,1.1701374615905885e2]
14 : [-@Inf@,@Inf@]
```

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
INRIA
centre de recherche NANCY - GRAND-EST
G. Hanrot
Arithmétique flottante : principes, méthodes, outils

Exemple de Muller en Cadna

```
      program jmm_seq
      use cadna
      parameter (nbit = 14)
      type (double_st) : : x0, x1, x2
C
      call cadna_init(-1, 0, 3)
      x0 = 5.5d0
      x1 = 61.d0/11.d0
      do i=1,nbit
         x2 = 111.d0 - (1130.d0 - 3000.d0/x0)/x1
         x0 = x1
         x1 = x2
         write (*, *)i,' :',str(x2)
      enddo
      call cadna_end()
C
      end
```


Résultats

```
-----------------------------------
CADNA software -- University P. et M. Curie -- LIP6
[ ... ]
-----------------------------------
U( 3) = 0.55901639344262E+001
U( 4) = 0.5633431085044E+001
U( 5) = 0.567464862051E+001
U( 6) = 0.57133290524E+001
U( 7) = 0.574912092E+001
U( 8) = 0.57818109E+001
U( 9) = 0.5811314E+001
U(10) = 0.583766E+001
U(11) = 0.5861E+001
U(12) = 0.588E+001
U(13) = 0.59E+001
U(14) = 0.6E+001
U(15) =@.0
U(16) =@.0
-----------------------------------
CADNA software -- University P. et M. Curie -- LIP6
BE CAREFUL : the self-validation detects major problem(s).
The results are NOT guaranteed
There is 1 numerical instability
1 UNSTABLE DIVISION(S)
[ ... ]
```

Annexe : URLs

- mpfr : www.mpfr.org.
- GP/Pari : pari.math.u-bordeaux.fr
- mpfi : mpfi.gforge.inria.fr
- Maple : www.maplesoft.com
- Mathematica : www.wolfram.com
- iRRAM : www.informatik.uni-trier.de/iRRAM
- GMP, MPF : www.swox.se/gmp
- Cadna : www.lip6.fr/cadna
- NTL : www.shoup.net/ntl

Présentation de la bibliothèque MPFR : domaine d'application, limites, originalité

Paul Zimmermann

INRIA Nancy - Grand Est

École Calcul Numérique Certifié, 25 octobre 2007

Prérequis (exposé G. Hanrot)

- la norme IEEE 754 (formats, arrondis, arrondi pair)
- les différentes approches (précision arbitraire, RealRAM, arithmétique d'intervalle, calcul stochastique)
- les outils correspondants (Pari/GP, Maple, Mathematica, iRRAM, MPFI, CADNA)

Plan de l'exposé

- Historique
- Modèle de calcul
- Originalité
- "MPFR inside" : algorithmes utilisés, stratégie de Ziv
- Performances
- Quelques applications
- Premier programme
- Support

Historique de MPFR

- ARC INRIA Fiable (1998-2000)
- novembre 1998 : texte fondateur (GH, J.-M. Muller, J. van der Hoeven, PZ)
- début 1999 : premières lignes de code (GH, PZ)
- juin-juillet 1999 : stage de Sylvie Boldo (AGM, log).
- 2000-2002 : ARC AOC (Arithmétique des Ordinateurs Certifiée)
- mars 2000 : dépôt APP

- décembre 2000 : V. Lefèvre rejoint la « MPFR-team »
- 2001 : postdoc David Daney
- mars 2002 : version 2.0.1
- 2003-2005 : Patrick Pélissier
- octobre 2004 : version 2.1.0, **gfortran** utilise MPFR
- octobre 2005 : MPFR vainqueur de *Many Digits*
- 2007 : GCC 4.3 utilise MPFR, version 2.3.0, parution de l'article ACM TOMS
- 2007-2009 : Philippe Théveny

Autres contributions : Fabrice Rouillier, Emmanuel Jeandel, Mathieur Dutour, Kevin Ryde, Laurent Fousse.

Modèle de calcul

Extension de IEEE 754 à la précision arbitraire :

- formats : précision p arbitraire

$$x = \pm 0.\underbrace{b_1 b_2 \ldots b_p}_{p \text{ bits}} \cdot 2^e$$

  avec $E_{\min} \le e \le E_{\max}$;
- plus 5 nombres spéciaux ± 0, $\pm\infty$, NaN ;
- arrondis : les 4 modes IEEE 754 (cf exposé GH).

Limites du modèle

- base fixée (2) ;
- précision $p \geq 2$;
- $E_{\min}$ et $E_{\max}$ sont globaux (par défaut $E_{\min} = -2^{30} + 1$, $E_{\max} = 2^{30} - 1$) ;
- pas de dénormalisés (mais `mpfr_subnormalize`) ;
- opérations **atomiques** (idem IEEE 754).

Particularités

- chaque variable a sa propre précision :

```
mpfr_init2 (a, 17);
mpfr_init2 (b, 42);
```

- opérations avec précisions mixtes autorisées ;

```
mpfr_sqrt (a, b, GMP_RNDN);
```

- opérations en place autorisées ;

```
mpfr_sqrt (a, a, GMP_RNDN);
```

Arrondi correct

Pour toute opération, MPFR garantit l'**arrondi correct** :

- opérateurs arithmétiques $(+, -, \times, \div)$;
- fonctions algébriques : $\sqrt{\cdot}, \sqrt{x^2 + y^2}, x^n, \ldots$;
- fonctions élémentaires et spéciales : exp, log, sin, . . ., erf, Bessel, . . .
- **conversions** (types `long`, `char*`, `double`, `long double`, `mpz_t`, `mpq_t`).

Conséquence : **un et un seul résultat** correct pour tout calcul !
Corollaire : les arrondis dirigés permettent d'implanter une arithmétique d'intervalles (MPFI).

Le bug d'Excel 2007

77.1×850 s'affiche 100000 au lieu de 65535
Bug d'**affichage**.

$$77.1 \to a = \frac{2712715088048947}{2^{45}} = 77.099999999999994316$$

$$850 \to 850$$

$$a \times 850 \to \frac{9007061815787519}{2^{37}} = 65535 - \frac{1}{2^{37}}$$

Portabilité

Précision au bit près, indépendante de la machine.

Exemple : précision demandée $p = 65$ bits.
On pourrait utiliser 3 mots sur un processeur 32 bits, et 2 mots sur un processeur 64.
$\Longrightarrow$ précision apparente 96 ou 128 bits : non portable !

MPFR utilise **exactement** 65 bits dans tous les cas (31 bits ou 63 bits forcés à zéro).

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE INRIA centre de recherche NANCY - GRAND-EST

Historique, Modèle de calcul, Originalité
MPFR inside et performances
Quelques applications
Premier programme, Support

Base 2^{32} ou 2^{64} (MPF)

$$x = \pm 0.w_1 w_2 \ldots w_p \cdot \beta^e$$

où $\beta \in \{2^{32}, 2^{64}\}$, et $0 \leq w_i < \beta$, $w_1 \neq 0$.

Ex1 : $x = 1/2$, précision 64 bits. MPFR : $0.\underbrace{100\ldots000}_{64} \cdot 2^0$

MPF, $\beta = 2^{32}$: $0.\underbrace{100\ldots000}_{32}\underbrace{000\ldots000}_{32} \cdot \beta^0$

Ex2 : $x = 1$, précision 64 bits. MPFR : $0.\underbrace{100\ldots000}_{64} \cdot 2^1$

MPF, $\beta = 2^{32}$: $0.\underbrace{000\ldots001}_{32}\underbrace{000\ldots000}_{32} \cdot \beta^1$

$\Longrightarrow$ précision effective dépend de la valeur de x !

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE INRIA centre de recherche NANCY - GRAND-EST

Plage d'exposants

La double précision IEEE 754 est limitée à 2^{1024} (environ 10^{308}).
$\Longrightarrow$ insuffisant pour certaines applications.

Exposants MPFR limités à $2^{30} - 1$: jusqu'à $10^{323228496}$.
$\Longrightarrow$ largement suffisant !

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
INRIA
centre de recherche NANCY - GRAND-EST

Historique, Modèle de calcul, Originalité
MPFR inside et performances
Quelques applications
Premier programme, Support

Exceptions

- Underflow : le résultat arrondi (avec plage illimitée d'exposant) est plus petit que $0.5 \cdot 2^{E_{\min}}$;
- Overflow : le résultat arrondi (avec plage illimitée d'exposant) est plus grand que $(1 - 2^{-p}) \cdot 2^{E_{\max}}$;
- Inexact : le résultat arrondi diffère du résultat exact ;
- Invalid : le résultat est NaN (Not-a-Number) ;
- Erange : résultat hors plage, exemple `mpfr_get_ui (x, GMP_RNDN)` lorsque x arrondi n'est pas représentable via un `unsigned int`.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
INRIA
centre de recherche NANCY - GRAND-EST

Paul Zimmermann Présentation de la bibliothèque MPFR

Exercice 1

Déterminer le signe de $\sin(10^{22})$.

GCC 4.1.2 :

```
#include <stdio.h>
#include <math.h>

int
main()
{
  double x = 1e22;
  printf ("sin(1e22)=%1.16e\n", sin (x));
}

bash-3.00$ ./a.out
sin(1e22)=4.6261304076460175e-01
```


$\sin 10^{22}$

Maple 11 :

```
> sin(1E22);
                          -.8522008498
```

Mathematica 5.0 :

```
In[6]:= N[Sin[10^22]]

Out[6]= 0.462613
```

PARI/GP 2.3.0 :

```
? sin(1e22)
%1 = -0.852200749
```

MuPAD 3.2.0 :

```
>> sin(1e22);
                          -0.9873536182
```

MPFR inside

Essentiellement 3 couches :

- fonctions de bas niveau (addition, soustraction, multiplication, division, racine carrée) ;
- fonctions élémentaires de base (exemple $\exp, \log, \sin$) ;
- autres fonctions élémentaires et spéciales.

Fonctions de bas niveau

Implantation native au-dessus de la couche `mpn` de GMP, et/ou manipulation directe de mots/bits en C.

$$0.\underbrace{1101011100}_{10} + 0.\underbrace{1111011110110}_{13} \cdot 2^{-3} \to \underbrace{xxxxxx}_{6} \cdot 2^{y}$$

```
.1101011100
   .1111011110110
.111101 0111110110
```

(the first six bits 111101 of the result are shown in red)

Fonctions élémentaires de base

Exemple : exp en précision n.

- réduction d'argument :

$$x \to r = x/2^k$$

- série de Taylor :

$$\exp r \approx 1 + r + \frac{r^2}{2!} + \frac{r^3}{3!} + \cdots + \frac{r^\ell}{\ell!}$$

- reconstruction :

$$x = r2^k \Longrightarrow \exp x = (\exp r)^{2^k}$$

Choix de k et ℓ : idéalement $k \approx \ell$.

Historique, Modèle de calcul, Originalité
MPFR inside et performances
Quelques applications
Premier programme, Support

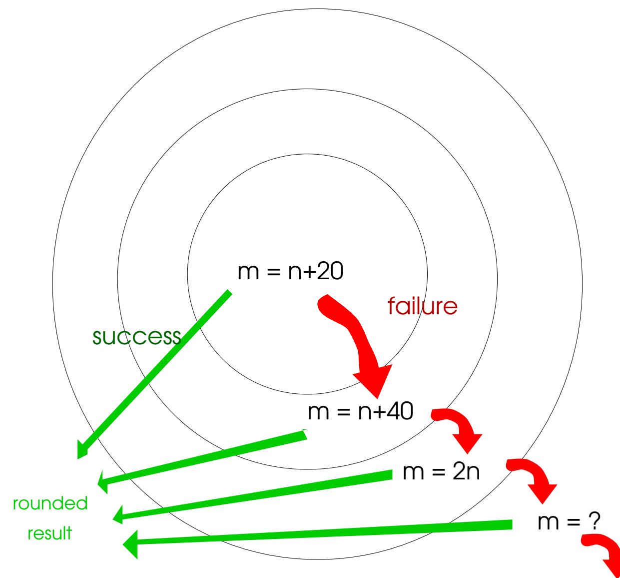
Différents algorithmes pour le calcul de

$$\exp r \approx 1 + r + \frac{r^2}{2!} + \frac{r^3}{3!} + \cdots + \frac{r^\ell}{\ell!}$$

- méthode naïve (petite précision) ;
- méthode « pas de bébé, pas de géant » (précision moyenne, $n \geq 700$ sur P4) ;
- scindage binaire pour les grandes précisions ($n \geq 24000$ sur P4).

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE INRIA centre de recherche NANCY - GRAND-EST
Paul Zimmermann Présentation de la bibliothèque MPFR

Stratégie de Ziv



- si un échec, $p \leftarrow p + 32$ ou $p + 64$;
- si ≥ 2 échecs, $p \leftarrow p + \lfloor p/2 \rfloor$.

Autres fonctions mathématiques

On se ramène aux fonctions de base, plus stratégie de Ziv :

$$\cosh x = \frac{e^x + e^{-x}}{2}$$

$$\begin{array}{lll} u & \leftarrow & \circ(e^x) \\ v & \leftarrow & \circ(u^{-1}) \\ w & \leftarrow & \circ(u + v) \\ s & \leftarrow & \frac{1}{2} w \qquad [exact] \end{array}$$

Tests

- tests de non-régression (bugs corrigés) ;
- tests aléatoires :

$$\begin{array}{rcl} y & \leftarrow & \circ_p(f(x)) \\ t & \leftarrow & \circ_{p+10}(f(x)) \\ z & \leftarrow & \circ_p(t) \end{array}$$

Si pas de problème de **double arrondi**, on doit avoir $y = z$.

- bases de données (attention . . .).

Double arrondi

Soit $y = \circ_p(f(x))$, et $t = \circ_q(f(x))$ avec $q > p$.
A-t-on toujours $y = \circ_p(t)$?

Non : $x = \frac{59}{64}$.

$$\sin x \approx 0.\underbrace{11001011111}_{11}101$$

$t = \circ_{11}(\sin x) = 0.\underbrace{11001}_{5}100000$.
$\circ_5(t) = 0.11010$ (arrondi pair).
$\circ_5(\sin x) = 0.11001 \neq t$.

Complexité

Le coût d'une opération est lié essentiellement — modulo la stratégie de Ziv — à la précision du résultat.

$\circ_{1000}(\exp x)$: si x a plus de 1000 bits, les bits de poids faible sont (en calcul amorti) ignorés.

Si x a moins de 1000 bits, le calcul peut être plus rapide.

(Au pire : dépendance linéaire en la précision des entrées.)

Efficacité en petite précision

Précision 53 bits sur Pentium 4 et Athlon (temps en cycles) :

version	machine	add	sub	mul	div	sqrt
2.0.1	Pentium 4	298	398	331	1024	1211
2.1.0	Pentium 4	211	213	268	549	1084
2.0.1	Athlon	222	323	270	886	975
2.1.0	Athlon	132	151	183	477	919

Comparaison MPFR, CLN, PARI, NTL

Athlon 1.8Ghz, temps en millisecondes, $x = \sqrt{3} - 1$, $y = \sqrt{5}$:

	chiffres	MPFR 2.2.0	CLN 1.1.11	PARI 2.2.12-beta	NTL 5.4
$x \cdot y$	10^2	**0.00048**	0.00071	0.00056	0.00079
	10^4	**0.48**	0.81	0.58	0.57
x/y	10^2	**0.0010**	0.0013	0.0011	0.0020
	10^4	**1.2**	2.4	**1.2**	**1.2**
$\sqrt{x}$	10^2	**0.0014**	0.0016	0.0015	0.0037
	10^4	**0.81**	1.58	0.82	1.23

Historique, Modèle de calcul, Originalité
MPFR inside et performances
Quelques applications
Premier programme, Support

Comparaison MPFR, CLN, PARI, NTL

	chiffres	MPFR 2.2.0	CLN 1.1.11	PARI 2.2.12-beta	NTL 5.4
$\exp x$	10^2	**0.017**	0.060	0.032	0.140
	10^4	**54**	70	68	1740
$\log x$	10^2	**0.031**	0.076	0.037	0.772
	10^4	**34**	79	40	17940
$\sin x$	10^2	**0.022**	0.056	0.032	0.155
	10^4	**78**	129	134	1860
$\text{atan}\, x$	10^2	0.28	**0.067**	0.076	NA
	10^4	610	**149**	151	NA

Quelques applications

- Analyse statique d'erreur d'arrondi (exposé Sylvie Putot)
- *constant folding* (gcc, gfortran) :

```
double x = sin (3.14);
```

- les autres MPtools : MPFI, MPC, MPCHECK
- filtres arithmétiques pour la géométrie algébrique (CGAL) : conversion `mpq_t` → `double` ;
- preuve formelle : Gappa, la conjecture de Kepler, preuve de Hales et projet FlysPecK.

Limites de MPFR

- Gestion des erreurs d'arrondi (composition) : arithmétique d'intervalles, ou modèle Real RAM
- Réglage automatique de la précision en fonction de l'exactitude (*accuracy*)
- La base est fixée à 2 : cf `decNumber` pour base 10
- Pas d'algorithmes de haut niveau d'analyse numérique : résolution d'équation polynomiale, algèbre linéaire (ALGLIB.NET), intégration numérique (CRQ)
- Les algos de MPFR sont uniquement prouvés "sur le papier" (pas de preuve formelle)

Exercice 2

Calculer 10 chiffres en arrondi au plus près de :

$$\sin(6303769153620408 \cdot 2^{971})$$

Exercice 3

Calculer 10 chiffres en arrondi au plus près de la solution $\rho \approx 2.219$ de

$$\exp(\sin x) = x.$$

Problème P21 de *Many Digits*.

A lot of code involving a little floating-point will be written by many people who have ***never attended*** *my (nor anyone else's) numerical analysis classes. We had to enhance the likelihood that* ***their programs would get correct results.*** *At the same time we had to ensure that people who really are expert in floating-point could write* ***portable software*** *and prove that it worked, since so many of us would have to rely upon it. There were a lot of almost* ***conflicting requirements*** *on the way to a balanced design.*

William Kahan, An Interview with the Old Man of Floating-Point, February 1998.

Premier programme

```
#include <stdio.h>
#include "mpfr.h"

int main ()
{
   unsigned long i;
   mpfr_t s, t;
   mpfr_init2 (s, 100); mpfr_init2 (t, 100);
   mpfr_set_ui (t, 1, GMP_RNDN);
   mpfr_set (s, t, GMP_RNDN);
   for (i = 1; i <= 29; i++)
      {
         mpfr_div_ui (t, t, i, GMP_RNDN);
         mpfr_add (s, s, t, GMP_RNDN);
      }
   mpfr_out_str (stdout, 10, 0, s, GMP_RNDN);
   printf ("\n");
   mpfr_clear (s); mpfr_clear (t);
}
```

```
#include "mpfr.h"
```
Inclut le fichier d'en-tête de MPFR (inclut aussi `gmp.h`).
```
mpfr_t s, t;
```
Déclare deux variables MPFR s et t (pointeurs).
```
mpfr_init2 (s, 100); mpfr_init2 (t, 100);
```
Initialise s et t, avec 100 bits de précision.
```
mpfr_set_ui (t, 1, GMP_RNDN);
mpfr_set (s, t, GMP_RNDN);
```
Met dans t la valeur 1, arrondie au plus proche, et copie t, arrondi au plus proche, dans s.


```
        mpfr_div_ui (t, t, i, GMP_RNDN);
```
Divise t par i, arrondi au plus proche.
```
        mpfr_add (s, s, t, GMP_RNDN);
```
Ajoute t à s, avec arrondi au plus proche.
```
mpfr_out_str (stdout, 10, 0, s, GMP_RNDN);
```
Affiche s en décimal, avec arrondi au plus proche (le nombre de décimales est déduit de la précision de s).
```
mpfr_clear (s); mpfr_clear (t);
```
Détruit la mémoire allouée pour s et t.

Si on remplace `GMP_RNDN` par `GMP_RNDZ`, on obtient une borne inférieure pour $e = \sum_{n \geq 0} \frac{1}{n!}$.

Compilation et exécution

```
$ gcc sample.c -lmpfr -lgmp
```

ou bien :

```
$ echo $MPFR
/usr/local/mpfr-2.3.0
$ gcc -I$MPFR/include sample.c $MPFR/lib/libmpfr.a
```

et puis :

```
$ ./a.out
2.7182818284590452353602874713481
```


Support

- la documentation : `info mpfr`
- le site `mpfr.org` (download, doc HTML, FAQ, patches)
- la *mailing list* `mpfr@loria.fr`
- `mpfr.gforge.inria.fr`
- le fichier `algorithms.tex` sur gforge
- le code source

Static analysis of the accuracy of floating-point computations

Sylvie Putot

CEA-LIST, MEASI (ModElling and Analysis of Systems in Interaction)

Ecole CEA-EDF-INRIA Calcul numérique certifié
25-26 octobre, LORIA - Nancy

Introduction

- FLUCTUAT: abstract interpretation based static analyzer for C
  - computes guaranteed bounds on errors between real number computation (what is expected) and the implementation in floating-point numbers
  - identifies operations responsible for the main precision losses
  - uses abstract domains based on affine arithmetic, and implemented with MPFR
- Applications :
  - safety-critical programs for instrumentation and control (mainly in aeronautics and nuclear industry) : using typically linear filters, interpolators, thresholds
  - towards verification of numerically intensive programs

Validation of accuracy "by hand" ?

- A popular way : try the algorithm with different precision (using matlab for example) and compare the results
- Example (by Rump) : in FORTRAN on an IBM S/370, computing with $x = 77617$ and $y = 33096$,

  $$f = 333.75y^6+x^2(11x^2y^2-y^6-121y^4-2)+5.5y^8+x/(2y)$$

  gives :
  - in single precision, $f = 1.172603...$
  - in double precision, $f = 1.1726039400531...$
  - in double extended precision, $f = 1.172603940053178...$
- We would deduce computation is correct ?
- True value is $f = -0.82739...$!!!

Overview of the talk

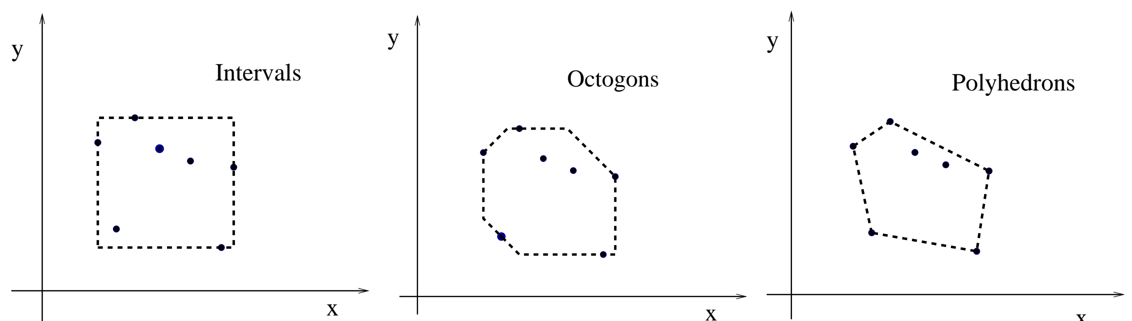
- Static analysis and abstract interpretation
- Concrete semantics for the evaluation of errors
- Abstraction (computable and implemented semantics)
  - First (simple but inaccurate) abstraction using intervals
  - Abstraction using affine arithmetic
- Examples
- Ongoing work : under-approximations (or inner-approximations)

Static analysis

- Analysis of the source code, for a set of inputs and parameters, without executing it
- Automatically infer run-time properties of programs, for sets of inputs :
  - invariant properties (true on all trajectories - for all possible inputs or parameters).
    Example : bounds on values of variables (no run-time error)
  - liveness properties (that become true at some moment on one trajectory).
    Examples : state reachability, termination

But undecidable in general

Thus abstraction to find computable over-approximations of sets of values : Abstract interpretation (Cousot/Cousot 77)



The analysis must terminate, may return an over-approximated information ("false alarm"), but never a false answer

Computing invariants in the abstract domain

- Starting from a *concrete semantics* (a model of execution), a computable *abstract semantics* is defined (for example abstraction of set of values by intervals)
- The semantics of a program (with control points $p_1, \ldots, p_n$) is given by a system of equations, of which we compute a fixpoint :

$$X = \begin{pmatrix} X_1 \\ \ldots \\ X_n \end{pmatrix} = F \begin{pmatrix} X_1 \\ \ldots \\ X_n \end{pmatrix}$$

- F is non-decreasing, least fixpoint is the limit of Kleene iteration $X^0 = \bot$, $X^1 = F(X^0)$, $\ldots$, $X^{k+1} = X^k \cup F(X^k)$, $\ldots$
- Iteration strategies, extrapolation (called widenings) to reach a fixpoint in finite time
- Abstract invariants over-approximate the concrete invariants

Example

`int x=0;`	`// 1`	$x_1 =$	$[0,0]$
`while (x<100) {`	`// 2`	$x_2 =$	$]-\infty, 99] \cap (x_1 \cup x_3)$
`  x=x+1;`	`// 3`	$x_3 =$	$x_2 + [1,1]$
`}`	`// 4`	$x_4 =$	$[100, +\infty[\cap (x_1 \cup x_3)$

- Iteration $i+1$ $(i < 100)$ [Kleene/Jacobi/Gauss-Seidl] :

$$\begin{array}{rcl} x_2^{i+1} & = & [0, i] \\ x_3^{i+1} & = & [1, i+1] \\ x_4^{i+1} & = & \bot \end{array}$$

- Fixpoint (101 Kleene iterations or widening/narrowing) :

$$x_2^\infty = [0, 99]; \quad x_3^\infty = [1, 100]; \quad x_4^\infty = [100, 100]$$

Analysis of programs with floating-point numbers

What is a correct program when using floating-point numbers ?

- No run-time error, such as division by 0, overflow, etc
- But also the program does compute something “not too far” from what is expected (=the result of the computation in real numbers, as the programmer usually thinks in real numbers)

For that, we compute :

- Bounds on floating-point and real values
- Bounds on the discrepancy error between the real and floating-point computations
- If possible, the main source of this error

Overview of the talk

- Static analysis and abstract interpretation
- Concrete semantics for the evaluation of errors
- Abstraction (computable and implemented semantics)
  - First (simple but inaccurate) abstraction using intervals
  - Second abstraction using affine arithmetic
- Examples
- Ongoing work : under-approximations (or inner-approximations)

IEEE 754 norm : correct rounding

- The result of a f.p. computation $x *_{\mathbb{F}} y$, ($*$ being $+, -, \times, /$), or of $\sqrt{x}$, is the rounded value (with chosen rounding mode) of the real result

$\rightarrow$ This specification allows to prove some properties on programs using floating-point numbers

- We note $fl(r) \in \mathbb{F}$ the rounded value of $r \in \mathbb{R}$ and $e(r) = r - fl(r) \in \mathbb{R}$ the corresponding rounding error

Concrete semantics for f.p. computation

Express difference between idealized and f.p. computation of a variable x :

$$r_x = f_x + \bigoplus_{\ell \in \mathcal{L}} \omega_x^\ell + \omega_x^{ho}$$

- $r_x \in \mathbb{R}$ is the result that would be computed if we had exact arithmetic available
- $f_x \in \mathbb{F}$ is the result of the computation in floating-point numbers
- $\mathcal{L}$ is the set of control points in the program
- $\omega_x^\ell \in \mathbb{R}$ is the contribution of point ℓ of the program to the first order error on f due to rounding
- errors of order more than 1 are grouped in $\omega_x^{ho} \in \mathbb{R}$

Concrete arithmetic operations

- Every arithmetic operation $x \Diamond_{\ell_i} y$, occurring at control point ℓ_i of the program, introduces a new rounding error $e(f_x \Diamond f_y)$ associated to label ℓ_i
- We note $\overline{\mathcal{L}} = \mathcal{L} \cup ho$

Componentwise propagation of existing errors in linear operations

$$r_x +^{\ell_i} r_y = fl(f_x + f_y) + \bigoplus_{u \in \overline{\mathcal{L}}} (\omega_x^u + \omega_y^u) + e(f_x + f_y)$$

Errors of order greater than one are agglomerated in one "higher order" term

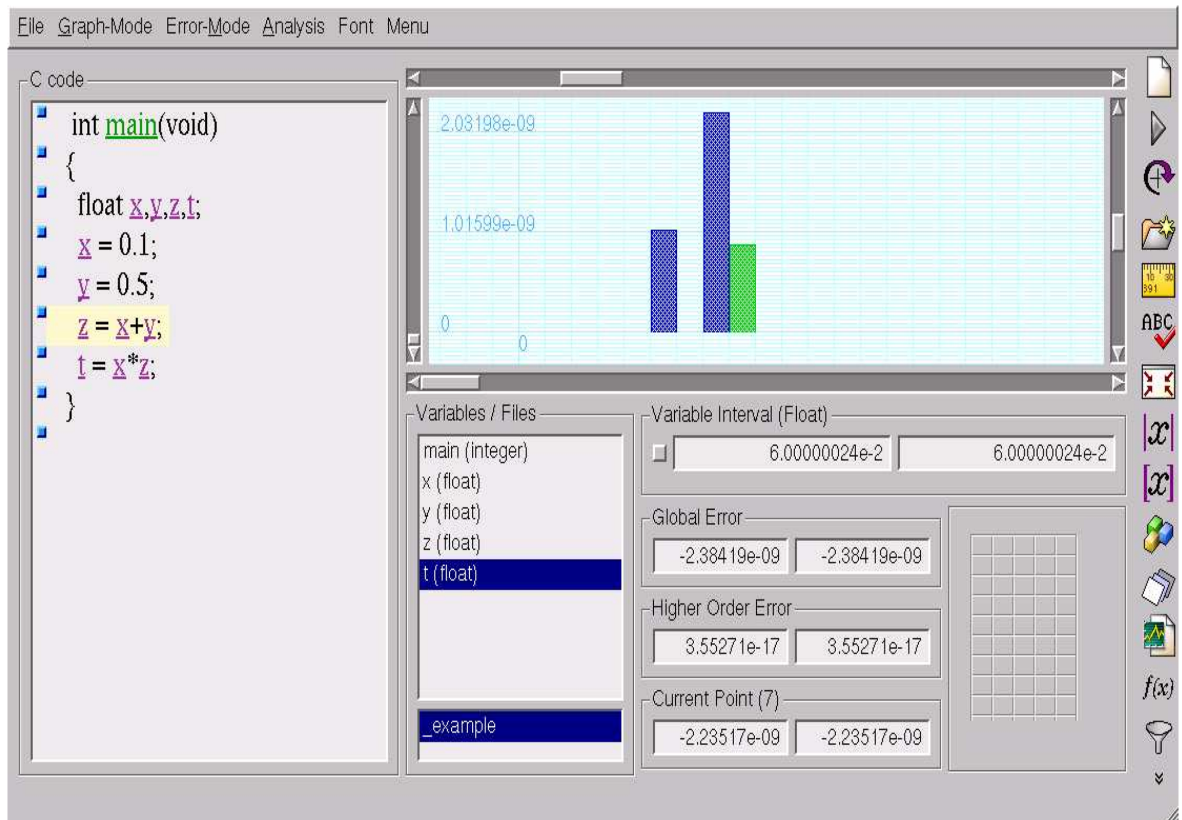
$$r_x \times^{\ell_i} r_y = fl(f_x f_y) + \bigoplus_{u \in \overline{\mathcal{L}}} (f_x \omega_y^u + f_y \omega_x^u) + (\sum_{u,v \in \overline{\mathcal{L}}} \omega_x^u \omega_y^v) + e(f_x f_y)$$

Example

```
float x,y,z,t;
x = 0.1; // [1]
y = 0.5; // [2]
z = x+y; // [3]
t = x*z; // [4]
```

$$\begin{aligned}
r_x &= 0.1 = f_x - 1.49...e^{-9}\ [1] \\
r_y &= 0.5 = f_y \\
r_z &= 0.6 = f_z - 1.49...e^{-9}\ [1] - 2.23...e^{-8}\ [3] \\
r_t &= 0.06 = f_t - 1.04...e^{-9}\ [1] - 2.23...e^{-9}\ [3] \\
&\quad +8.94...e^{-10}\ [4] + 3.55...e^{-17}\ [ho]
\end{aligned}$$

The FLUCTUAT user interface : error graph for t



Overview of the talk

- Static analysis and abstract interpretation
- Concrete semantics for the evaluation of errors
- Abstraction (computable and implemented semantics)
  - First (simple but inaccurate) abstraction using intervals
  - Second abstraction using affine arithmetic
- Examples
- Ongoing work : under-approximations (or inner-approximations)

Non relational abstract domain

Express difference between idealized and f.p. computation :

$$r = f + \bigoplus_{\ell \in \mathcal{L}} \omega^{\ell} + \omega^{ho}$$

Natural abstraction :

- f is abstracted by an interval of f.p. numbers (float or double depending the type of the variable), with rounding to the nearest
- the ω^{ℓ} are abstracted by intervals with higher-precision numbers (using MPFR library), with outward rounding

But too conservative, non relational :

- extreme example : if $X = [-1, 1]$, $X - X$ computed in interval arithmetic is not 0 but $[-2, 2]$

Computation of the rounding error with MPFR

- If the floating point operands, for example of a binary operation $x \diamond y$, are singleton (what we call *symbolic execution*)
  - floating-point result $f_x \diamond_{\mathbb{F}} f_y$ is computed exactly
  - real result $f_x \diamond_{\mathbb{R}} f_y$ can be bounded with an interval computation with arbitrary precision and outward rounding - even computed exactly if we use a large enough MPFR mantissa
  - rounding error $e(f_x \diamond_{\mathbb{R}} f_y) = (f_x \diamond_{\mathbb{R}} f_y) - (f_x \diamond_{\mathbb{F}} f_y)$ is bounded with an interval computation with (same) arbitrary precision and outward rounding
- If the operands are not singletons, the rounding error due to the operation is bounded by an interval centered on zero, of width the ulp of the result
- Default precision of the analysis in Fluctuat (for errors computation) is mpfr numbers with 60 bits mantissa

Example : a scheme by Kahan and Muller

Compute, with $x_0 = 11/2.0$ and $x_1 = 61/11.0$, the sequence

$$x_{n+2} = 111 - \frac{(1130 - \frac{3000}{x_n})}{x_{n+1}}$$

- If computed with real numbers, converges to 6. If computed with any approximation, converges to 100.
- Results with Fluctuat :
  - for x_{11} : with default precision $f_{11} = 100.007317$, with e_{11} in [-94.1261,-94.1258] (thus r_{11} in [5.8812,5.8815])
  - for x_{100} :
    - default precision of the analysis, or even up to 400 mantissa bits numbers, finds $f_{100} = 100$, $e_{100} \in [-\infty, +\infty]$ (thus $r_{100} \in [-\infty, +\infty]$) : indicates high unstability
    - with 500 mantissa bits numbers, finds $f_{100} = 100$, $e_{100} = -94$ and $r_{100} = 5.99...$

Example : a "real-world" order 2 filter

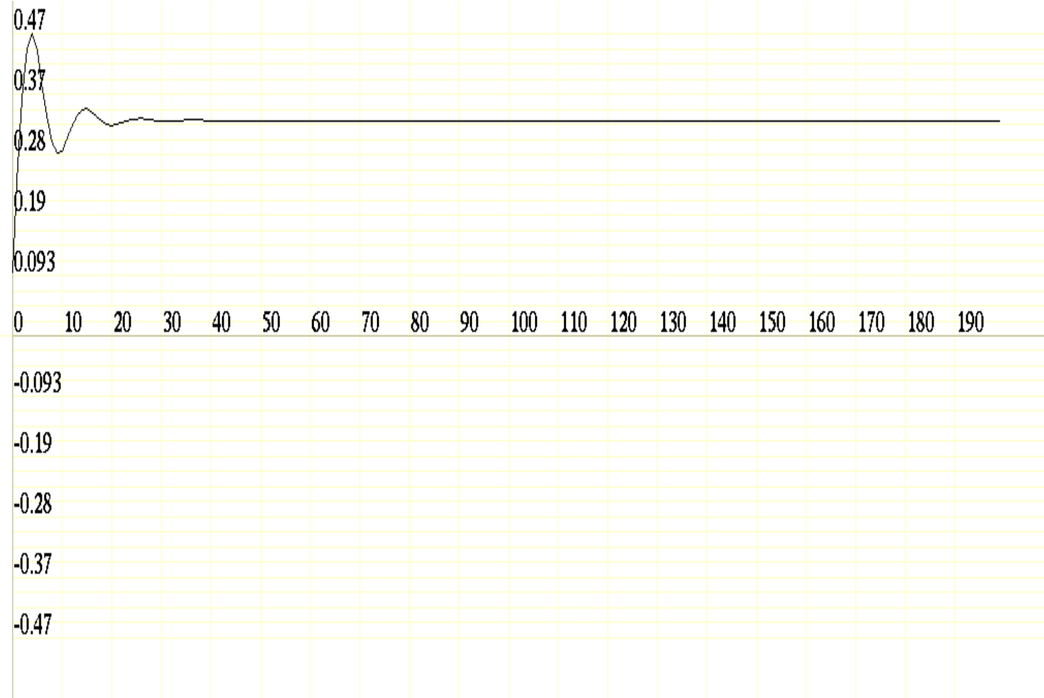
We want to compute the value and error on the 200th first iterations of

$$S_i = 0.1 + 1.4 S_{i-1} - 0.7 S_{i-2}$$

where $S_0 = S_1 = 0$.

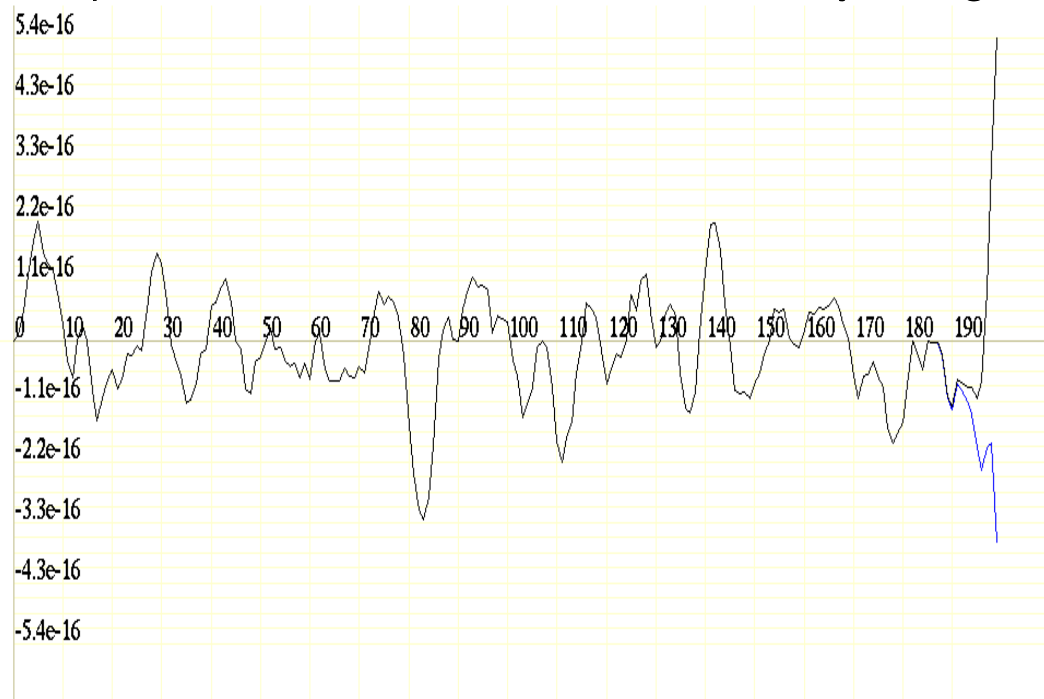
- greatest module of eigenvalue is 0,836
- default precision of the analysis (60 bits) : error on S_{200} in $[-2.5e^{32}, 2.5e^{32}]$
- takes 180 bits to have an error on S_{200} whose bounds are equal up to 3 significant digits : error in $[9.03e^{-17}, 9.031e^{-17}]$.
- another order 2 filter with greatest module of eigenvalue equal to 0.95 needs 240 bits to achieve same accuracy

Evolution of the f.p. value with the iterations



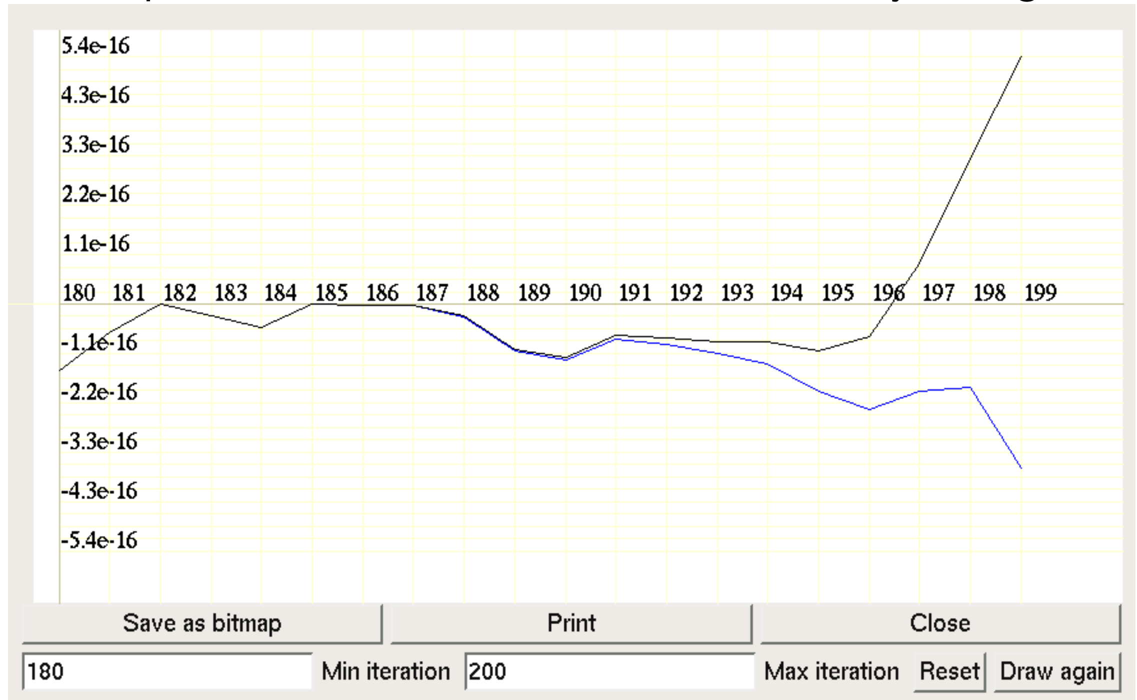
Evolution of estimated error with iterations

With a precision of 164 bits of mantissa : finally diverges



Zoom on the estimated error for last iterations

With a precision of 164 bits of mantissa : finally diverges

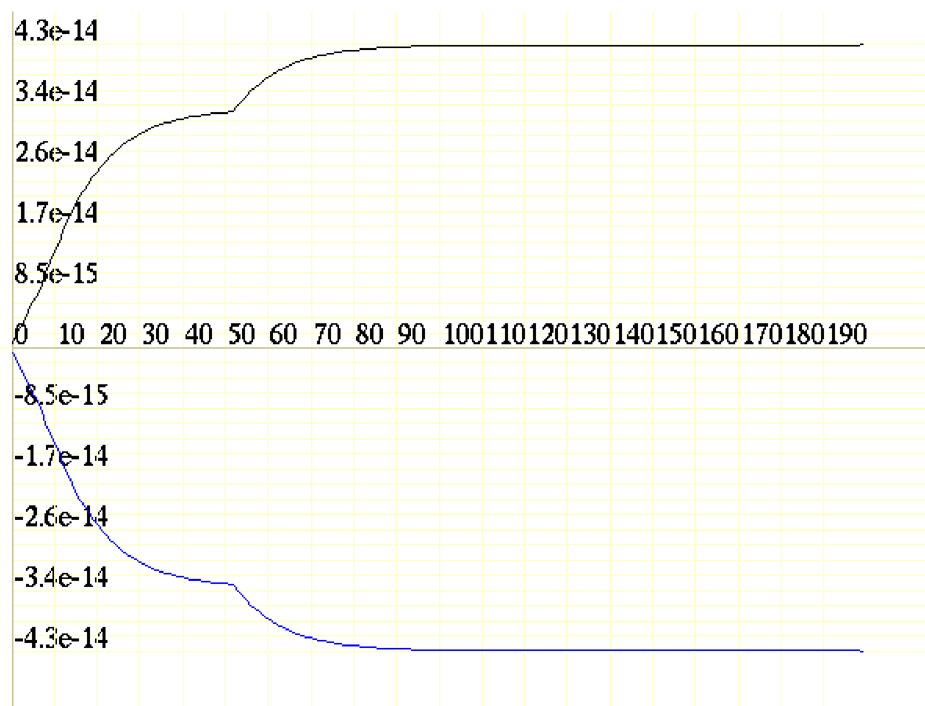


Problem of loss of correlations in interval computations !

A better behaviour here ...

Result for the error with default MPFR precision for $S_0 = S_1 = 0$, the $E_i = 0.2$, and

$$S_i = 0.7E_i - 1.3E_{i-1} + 1.1E_{i-2} + 0.75S_{i-1} - 0.15S_{i-2}.$$



More places where MPFR helps or could help

- Parameterization of the precision of intermediate computation in arithmetic expressions
  - Fluctuat supposes all intermediary computations are in the IEEE float or double format
  - however they may be stored with different format in registers
- Transcendental functions, e.g. trigonometric functions
  - MPFR gives a tight estimation of the real result
  - and we parameterize the accuracy of the library used
- Acceleration of the fixpoint computation by progressive reduction of the precision of numbers used (see next slide)

Acceleration of the fixpoint computation

The idea on a very basic example : $x_0 \in [0, 1]$, fixpoint computation by Kleene iteration (with union over iterates) of

```
while () x = 0.1*x;
```

yields for the error $\omega_0 = 0, \ \omega_n = 0.1 * \omega_{n-1} + \delta$. For more simplicity, we take $\delta = [-1, 1]$ and numbers in base 10.

- would not terminate in infinite precision
- in finite precision, #iterations depends on the precision :
  - 3 significant digits $\longrightarrow$ 4 iterations :

$$\begin{aligned}
\omega_1 &= \delta = [-1, 1] \\
\omega_2 &= [-0.1, 0.1] + [-1, 1] = [-1.1, 1.1] \\
\omega_3 &= [-0.11, 0.11] + [-1, 1] = [-1.11, 1.11] \\
\omega_4 &= [-0.111, 0.111] + [-1, 1] = [-1.12, 1.12] \\
\omega_5 &= [-0.112, 0.112] + [-1, 1] = [-1.12, 1.12]
\end{aligned}$$

  - N significant digits $\longrightarrow$ N+1 iterations

$\Rightarrow$ Acceleration by progressive reduction of the precision

Overview of the talk

- Static analysis and abstract interpretation
- Concrete semantics for the evaluation of errors
- Abstraction (computable and implemented semantics)
  - First (simple but inaccurate) abstraction using intervals
  - Second abstraction using affine arithmetic
- Examples
- Ongoing work : under-approximations (or inner-approximations)

Affine Arithmetic for real numbers

- Proposed in 1993 by Comba, de Figueiredo and Stolfi as a more accurate extension of Interval Arithmetic
- A variable x is represented by an affine form $\hat{x}$:

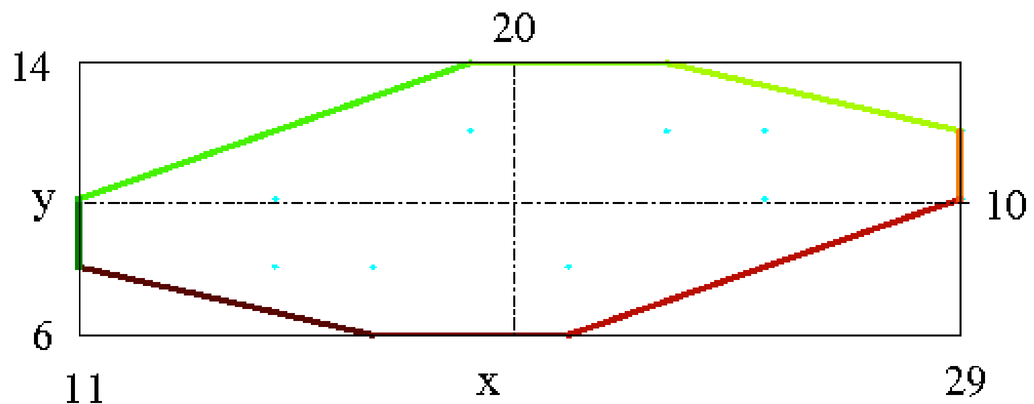
$$\hat{x} = x_0 + x_1\varepsilon_1 + \ldots + x_n\varepsilon_n,$$

  where $x_i \in \mathbb{R}$ and the ε_i are independent symbolic variables with unknown value in $[-1, 1]$.
  - $x_0 \in \mathbb{R}$ is the *central value* of the affine form
  - the coefficients $x_i \in \mathbb{R}$ are the *partial deviations*
  - the ε_i are the *noise symbols*
- The sharing of noise symbols between variables expresses *implicit dependency*

Sub-polyhedric relations

Concretization is a center-symmetric convex polytope

$$\hat{x} = 20 - 4\varepsilon_1 + 2\varepsilon_3 + 3\varepsilon_4$$
$$\hat{y} = 10 - 2\varepsilon_1 + \varepsilon_2 - \varepsilon_4$$



- Non linear operations are easily interpretable with order 1 Taylor developments
- Construction of relations is dynamic and simple
- Low complexity

Affine arithmetic : arithmetic operations

- *Assignment* of a variable x whose value is given in a range $[a, b]$ at label i, introduces a noise symbol ε_i :

$$\hat{x} = \frac{(a+b)}{2} + \frac{(b-a)}{2}\varepsilon_i.$$

- *Addition* of affine forms is computed componentwise:

$$\hat{x} + \hat{y} = (\alpha_0^x + \alpha_0^y) + (\alpha_1^x + \alpha_1^y)\varepsilon_1 + \ldots + (\alpha_n^x + \alpha_n^y)\varepsilon_n$$

  For example, with real (exact) coefficients , $f - f = 0$.
- *Multiplication* : we select an approximate linear form, the approximation error creates a new noise term :

$$\hat{x} \times \hat{y} = \alpha_0^x \alpha_0^y + \sum_{i=1}^{n} (\alpha_i^x \alpha_0^y + \alpha_i^y \alpha_0^x)\varepsilon_i + (\sum_{i=1}^{n} |\alpha_i^x| . \sum_{i=1}^{n} |\alpha_i^y|)\varepsilon_{n+1}.$$

- *More generally* : approximate linear form obtained for any operation by an order 1 Taylor form

Affine forms define implicit relations : example

Consider, with $a \in [-1, 1]$ and $b \in [-1, 1]$, the expressions

```
x = 1 + a + 2 * b;
y = 2 - a;
z = x + y - 2 * b;
```

- The representation as affine forms is $\hat{x} = 1 + \epsilon_1 + 2\epsilon_2$, $\hat{y} = 2 - \epsilon_1$, with noise symbols $\epsilon_1, \epsilon_2 \in [-1, 1]$
- This implies $\hat{x} \in [-2, 4]$, $\hat{y} \in [1, 3]$ (same as I.A.)
- It also contains implicit relations, such as $\hat{x} + \hat{y} = 3 + 2\epsilon_2 \in [1, 5]$ or $\boxed{\hat{z} = \hat{x} + \hat{y} - 2b = 3}$
- Whereas we get with intervals

$$\boxed{z = x + y - 2b \in [-3, 9]}$$

Affine forms and existing relational domains

- *More expressive* than octagons ($\pm x \pm y \leq c$) [A. Mine]
- Provides *Sub-polyhedric* relations (there is *a concretization* to center-symmetric bounded convex polytope)
- But by *some aspects* better than polyhedra [P. Cousot/N. Halbwachs]
  - for example, to interpret *non-linear computations* :
    - dynamic linearization of non-linear computations
  - much more efficient in *computation time and memory*
    - dynamic construction of relations
    - no static packing of variables needed
- Close to *dynamic templates* [Z. Manna]

From real to floating-point computation

- Affine arithmetic uses symbolic properties of real number computation, such as associativity and distributivity of $+, \times$
- These properties do not hold exactly for floating-point numbers, thus affine arithmetic can not be directly used for floating-point estimation
- Example :
  - let $x \in [0, 2]$ and $y \in [0, 2]$, we consider $((x + y) - x) - y$.
  - with affine arithmetic : $x = 1 + \varepsilon_1$, $y = 1 + \varepsilon_2$ $((x+y)-x)-y = ((2+\varepsilon_1+\varepsilon_2)-1-\varepsilon_1)-1-\varepsilon_2 = 0$
  - false in floating-point numbers : take $x = 2$ and $y = fl(0.1)$, then in simple precision $((x + y) - x) - y = -9.685755e - 08$

Relational Analyzes using affine arithmetic

- *Relational* because expresses relations between variables, and not only properties on each variable independently
- Using affine arithmetic (or variations) for linear correlations between variables :
  - for the computation of the real value (r)
  - for the computation of the errors due to the use of floating-point numbers (ω^l) :
    - each elementary rounding error introduces a new noise term
    - decomposition of errors on their provenance in the program
  - We deduce bounds for the floating-point value by the sum of real values and errors

Affine arithmetic in static analysis

- Dynamic partitioning : control of the dependencies introduced in loops we want to keep, to reduce computation time and memory
  - relations only inside one loop iteration
  - relations between n successive iterations
  - after a loop, either keep dependencies or agglomerate them
- Interpretation of non arithmetic expressions such as union, intersection, widening :
  - for example intersection for tests, union over branches after tests or at the end of loops with variable number of iterations
  - optimal union or intersection is difficult : depends on operations that follow

Join operation on affine forms

- Let $[\alpha_i^x \cup \alpha_i^y] = [\alpha_i^x, \alpha_i^y]$ if $\alpha_i^x \leq \alpha_i^y$ else $[\alpha_i^y, \alpha_i^x]$
- A natural join between $\hat{x}$ and $\hat{y}$ is

$$\hat{x} \cup \hat{y} = [\alpha_0^x \cup \alpha_0^y] + \sum_{i \in L} [\alpha_i^x \cup \alpha_i^y] \varepsilon_i$$

  Result might be greater than the union of enclosing intervals, but may be more interesting to keep correlations
- But with interval coefficients $\boxed{(\hat{x} \cup \hat{y}) - (\hat{x} \cup \hat{y}) \neq 0}$ we get back to the defects of intervals

Join operation on affine forms

For an interval **i**, we note

$$\text{mid}(\mathbf{i}) = \frac{\underline{i} + \overline{i}}{2}, \quad \text{dev}(\mathbf{i}) = \overline{i} - \text{mid}(\mathbf{i})$$

the center and deviation of the interval.

- A better join is then

$$\begin{aligned} \hat{x} \cup \hat{y} \quad = \quad & \text{mid}([\alpha_0^x, \alpha_0^y]) + \sum_{i \in L} \text{mid}([\alpha_i^x, \alpha_i^y])\, \varepsilon_i + \\ & \sum_{i \in L \cup \{0\}} \text{dev}([\alpha_i^x, \alpha_i^y])\, \varepsilon_k^u \end{aligned}$$

- Then we have affine forms with real coefficients again

Example (join)

Let $\hat{x} = 1 + 2\varepsilon_1 + \varepsilon_2$ and $\hat{y} = 2 - \varepsilon_1$.

- Join on intervals : $[x] \cup [y] \in [-2, 4]$
- First join on affine forms :
  - $\hat{x} \cup \hat{y} = [1, 2] + [-1, 2]\varepsilon_1 + [0, 1]\varepsilon_2 \subset [-2, 5]$
  - larger enclosure than on intervals but it may still be interesting for further computations to keep relations
- Second join on affine forms :
  - $\hat{x} \cup \hat{y} = 1.5 + 0.5\varepsilon_1 + 0.5\varepsilon_2 + 2.5\varepsilon_3^u \subset [-2, 5]$
  - same enclosure, but $(\hat{x} \cup \hat{y}) - (\hat{x} \cup \hat{y}) = 0$

Implementation using MPFR

- For implementation of affine forms, we do not have real numbers but arbitrary precision floating-point coefficients (using MPFR)
- One solution is to compute each coefficient of the affine form with intervals of f.p. numbers with outward rounding
  - inaccurate because we soon get intervals coefficients, and their defects
  - the precision of the MPFR numbers used has a strong influence on the results
- More accurate : keep point coefficients and handle uncertainty on the computation on these coefficients by creating new noise terms
  - we could imagine to quantify the loss of precision due to the limited precision of MPFR numbers used in the analysis by summing these particular noise terms
  - loss of precision due to the model of analysis tackled by the under-approximation

Overview of the talk

- Static analysis and abstract interpretation
- Concrete semantics for the evaluation of errors
- Abstraction (computable and implemented semantics)
  - First (simple but inaccurate) abstraction using intervals
  - Second abstraction using affine arithmetic
- Examples
- Ongoing work : under-approximations (or inner-approximations)

A non linear Newton scheme

Computes the inverse of A, that can take any value in [20,30] :

```
double xi, xsi, A, temp;
signed int *PtrA, *Ptrxi, cond, exp, i;

A = __BUILTIN_DAED_DBETWEEN(20.0,30.0);

/* initial condition = inverse of nearest power of 2 */
PtrA = (signed int *) (&A);
Ptrxi = (signed int *) (&xi);
exp = (signed int) ((PtrA[0] & 0x7FF00000) >> 20) - 1023;
xi = 1; Ptrxi[0] = ((1023-exp) << 20);

temp = xsi-xi; i = 0;
while (abs(temp) > e-10) {
  xsi = 2*xi-A*xi*xi;
  temp = xsi-xi;
  xi = xsi;
  i++;
}
```

Analysis of the Newton scheme

- We want to prove that for all A in [20,30], this Newton scheme terminates (in a reasonable number of iterations), for chosen stopping criterium eps=e-10
  - true in real numbers for all eps
  - not obvious in finite precision
- Symbolic execution
  - A = 20.0 : i = 5, xi = 5.0e-2 + [-2.82e-18,-2.76e-18]
  - A = 30.0 : i = 9, xi = 3.33e-2 + [-5.28e-18,6.21e-18]
- Static analysis for A in [20.0,30.0] :
  - Intervals : analysis *does not prove termination*
  - Affine forms for the real value (with 10000 subdivisions) : analysis finds

i in [5,9], xi in [3.33e-2,5.0e-2]+ [-4.21e-13,4.21e-13]

    (affine arithmetic for the errors not yet implemented for these computations)

In simple precision

- Replace `double` by `float`; the static analyser does not find convergence (with `epsilon=1e-10`)
- Automatic symbolic execution for 1000 input values in [20.0,30.0]
  - for the 517 first values, the number of iterations is in [5,7]
  - for the 518th value, A = 25.18, the algorithm does not converge : the difference between two successive iterates alternates between -3.725290e-09 and 3.725290e-09
- Even when convergence, what is the validity ?
  - A = 25.46999931335449219, convergence in 6 iterations, difference between the two last floating-point iterates is 0
  - but rounding error is 2.e-09, 20 times larger than stopping criterion!
- With `epsilon=1e-7`, proof of convergence with i in [4,12]

A second-order linear recursive filter

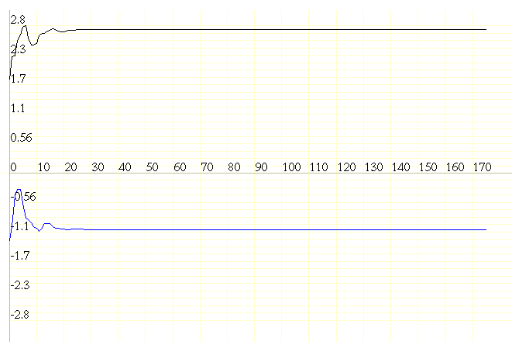
- *Linear Filter of order 2*

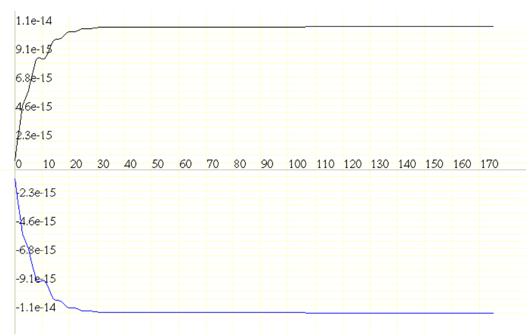
$$S_i = 0.7E_i - 1.3E_{i-1} + 1.1E_{i-2} + 1.4S_{i-1} - 0.7S_{i-2}$$

where $S_0 = S_1 = 0$, and the E_i are independent inputs in the range $[0, 1]$, that can be modelled by

$$\hat{E}_i = \frac{1}{2} + \frac{1}{2}\epsilon_i.$$

- With intervals : value and errors in $[-\infty, +\infty]$ (even increasing the number of bits of the mantissa)
- Relational analysis on values and errors :





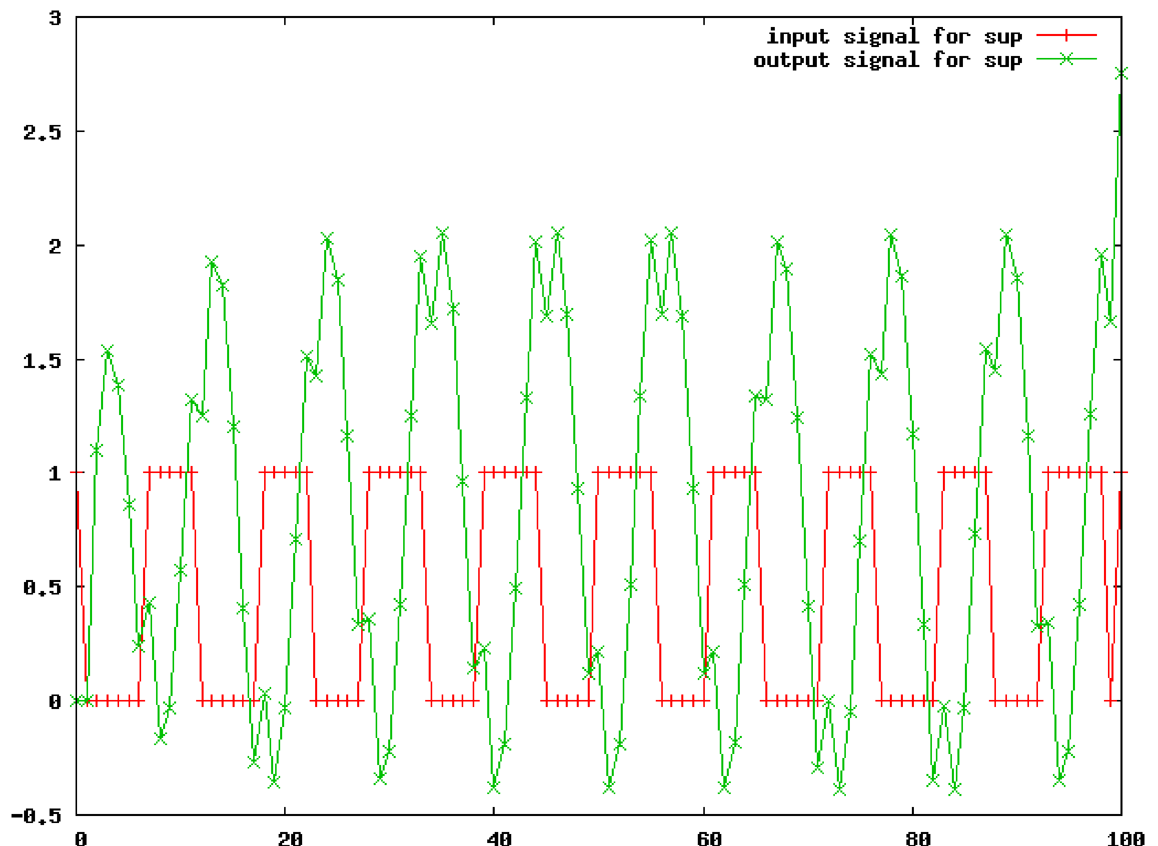
Second-order filter

- *Fixed unfolding* ($i = 99$) : affine form for the real value of the output

$$\hat{S}_{99} = 0.83 + 7.81e^{-9}\varepsilon_0 - 2.1e^{-8}\varepsilon_1 + \ldots - 0.16\varepsilon_{98} + 0.35\varepsilon_{99}$$

  - supposing coefficients computed exactly, gives the *exact enclosure of* S_{99} : $[-1.0907188500, 2.7573854753]$
  - *extreme scenario for* S_{99} : the coefficients of the affine form allow us to deduce the E_i leading to the max (or min) of the enclosure ($\alpha_i \geq 0 \Rightarrow E_i = 1$ else $E_i = 0$)
- *Real enclosure* well approached using finite sequences : $S_\infty = [-1.09071884989..., 2.75738551656...]$

The input sequence that maximizes S_{100}



Overview of the talk

- Static analysis and abstract interpretation
- Concrete semantics for the evaluation of errors
- Abstraction (computable and implemented semantics)
  - First (simple but inaccurate) abstraction using intervals
  - Second abstraction using affine arithmetic
- Examples
- Ongoing work : under-approximations (or inner-approximations)

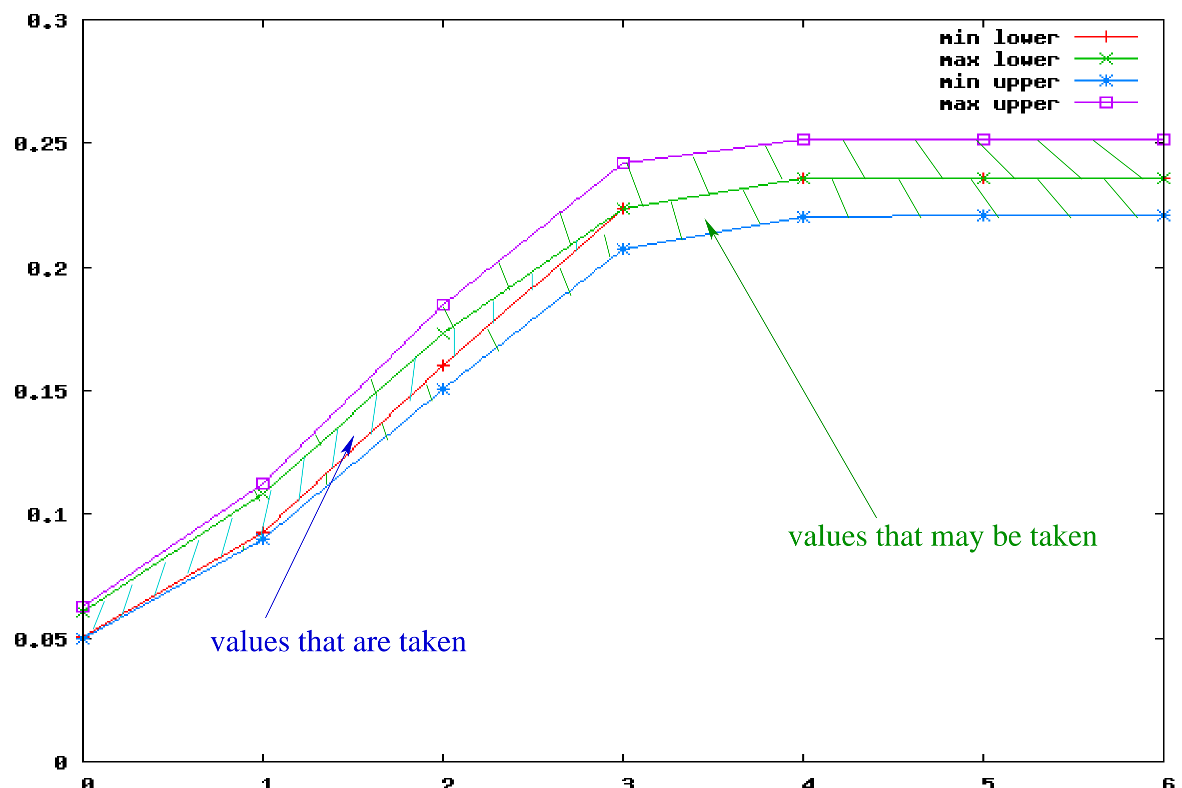
Under-approximation (with separate prototype)

- Existing abstract domains mainly for over-approximation
  - Results are sure but may be pessimistic ("false alarms")
  - How pessimistic ?
- What we mean by under-approximation : sets of values of the outputs, that are sure to be reached for some inputs in the specified ranges.
- How : extensions of computation on affine forms with Kaucher/generalized arithmetic [Goldsztejn]
- Applications
  - Joint use of under- and over-approximation to characterize the quality of analysis results
  - Extract scenarii giving extreme values in the general case (not only linear)

A square-root algorithm (Householder method)

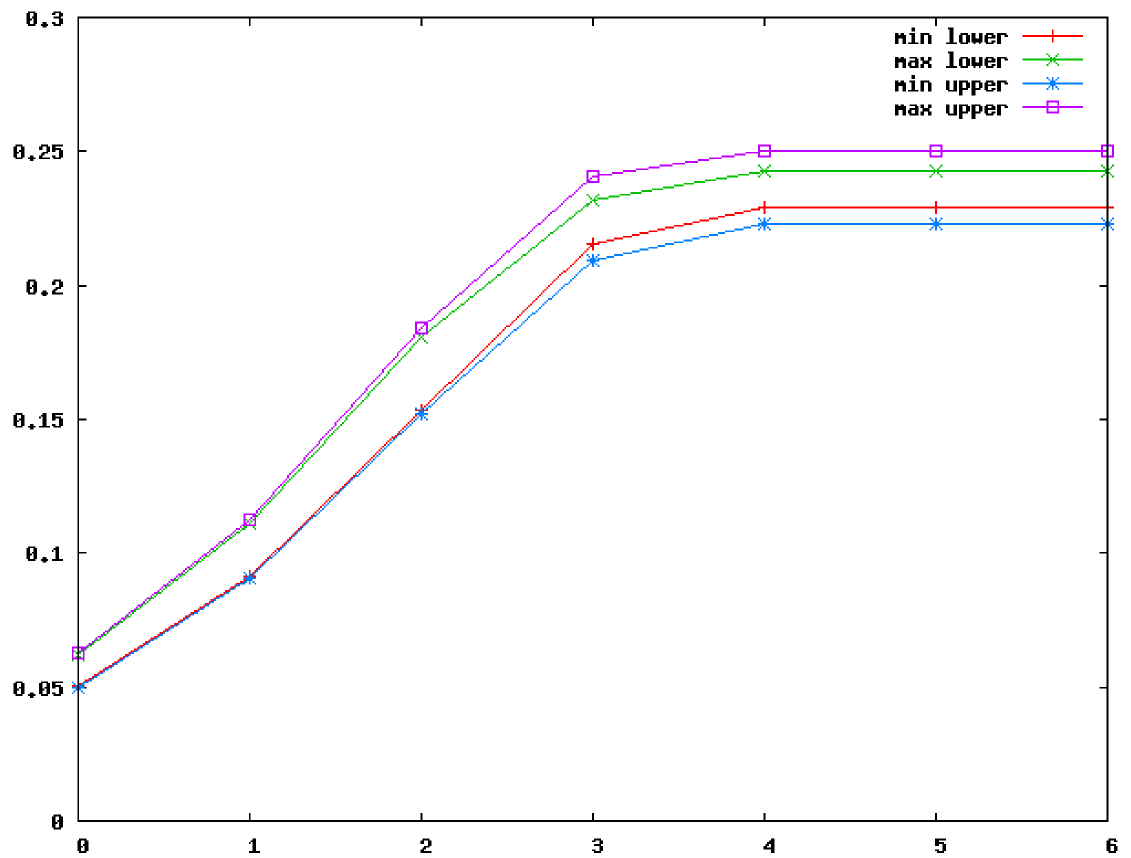
```
#define _EPS 0.00002
double Input, Output, x, xp1, residue, shouldbezero
int i = 0;

Input = __BUILTIN_DAED_DBETWEEN(16.0,20.0);
x = 1.0/Input; xp1 = x;
residue = 2.0*_EPS;
while (fabs(residue) > _EPS) {
  xp1 = x*(1.875+Input*x*x*(-1.25+0.375*Input*x*x))
  residue = 2.0*(xp1-x)/(x+xp1);
  x = xp1;
  i++;
}
Output = 1.0 / x;
shouldbezero = x*x-1.0/Input;
```

Evolution of x_i with iterations (no subdivision)



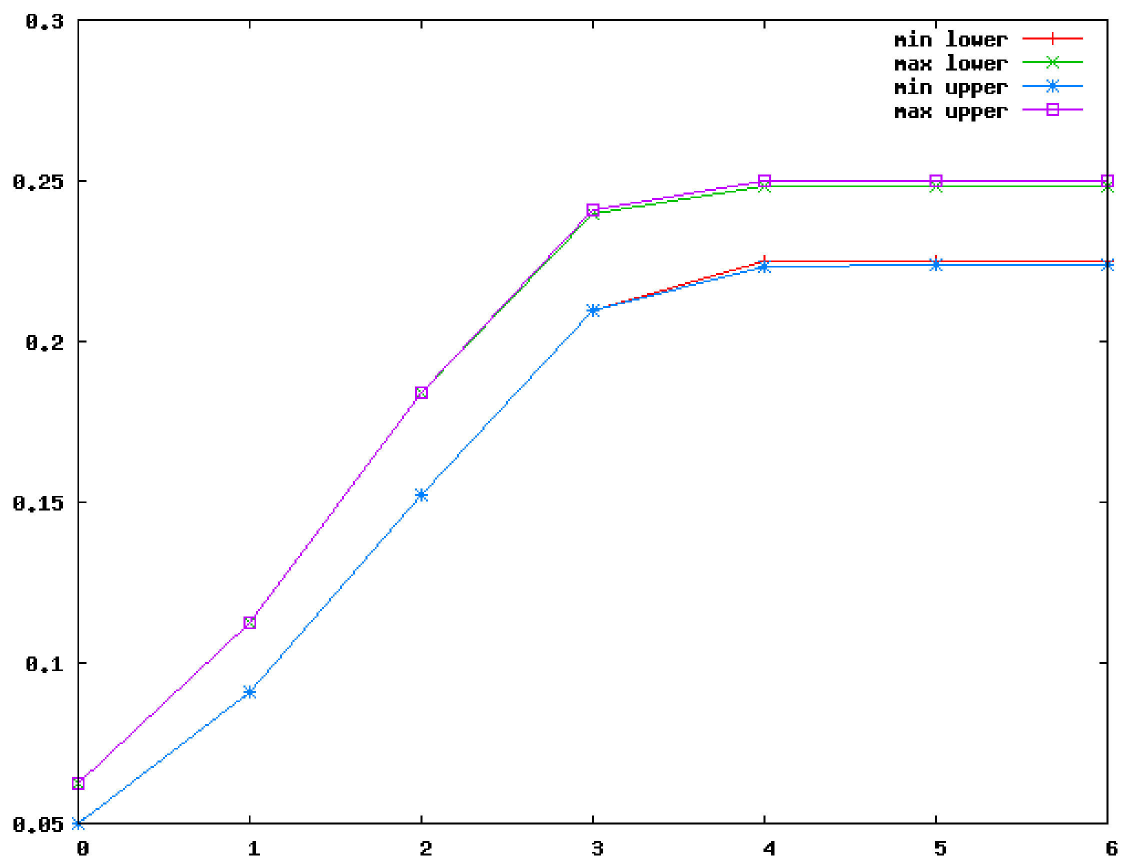
Evolution of x_i with iterations (2 subdivisions)

Over- and under-approximation



Evolution of x_i with iterations (8 subdivisions)

Over- and under-approximation



Functional proof of this numerical algorithm?

- With 32 subdivisions of the input
  - Stopping criterion of the Householder algorithm is satisfied after 5 iterations :

$$[0, 0] \subseteq \text{residue}(x_4, x_5) \subseteq [-1.44e^{-5}, 1.44e^{-5}]$$

  - Tight enclosure of the iterate :

$$[0.22395, 0.24951] \subseteq x_5 \subseteq [0.22360, 0.25000]$$

  - Functional proof :

$$[0, 0] \subseteq \text{shouldbezero} \subseteq [-1.49e^{-6}, 1.49e^{-6}]$$

Conclusion: why & how we use MPFR since 2001

- Need of an arbitrary precision library with well-defined semantics (we want to *prove* properties)
  - essentially directed rounding with arbitrary precision
  - but also correct rounding with arbitrary precision so that we have the possibility to consider (some day) different formats than IEEE float/double
- Efficiency is an issue for us (costly affine forms)
  - for example constructive reals such as often used in proof-theoretic approaches are too slow
  - mainly for small to middle-size mantissa (default precision, often sufficient, is 60 bits, but we go up to thousands of bits)

Conclusion: why & how we use MPFR since 2001

- Main functions used are mpfr_cmp, mpfr_neg, mpfr_add, mpfr_add_ui, mpfr_sub, mpfr_sub_ui, mpfr_mul, mpfr_div, mpfr_ui_div, mpfr_div_2exp, mpfr_div_ui, mpfr_add_one_ulp, mpfr_sub_one_ulp, but also mpfr_sqrt, mpfr_abs, mpfr_log, mpfr_exp, mpfr_sin_cos, mpfr_trunc, mpfr_floor, mpfr_ceil
- Very convenient to use after a short adaptation : for our (quite basic) use, the only bug we detected in the last few years was solved by updating to a newer version!
- Very quick answer to any question

Many thanks to the MPFR team !

Related work and tools

- The ASTREE static analyzer (see references)
  - Detection of run-time error for large synchronous instrumentation software
  - Using in particular octagons and domains specialized for order 2 filters (ellipsoids)
  - Taking floating-point arithmetic into account

  `http://www.astree.ens.fr/`
- CADNA : estimation of the roundoff propagation in scientific programs by stochastic testing
  `http://www-anp.lip6.fr/cadna/`
- GAPPA : automatic proof generation of arithmetic properties
  `http://lipforge.ens-lyon.fr/www/gappa/`

References

- An Introduction to Affine Arithmetic, by J. Stolfi and L.H. de Figueiredo, TEMA 2003

http://www.sbmac.org.br/tema/seletas/docs/v4_3/101_01summary.pdf

- Abstract Interpretation: Achievements and Perspectives, by P. Cousot, SSGRR 2000

http://www.di.ens.fr/~cousot/COUSOTpapers/SSGRRP-00-PC.shtml

- Static analysis-based validation of floating-point computations, by S. Putot, E. Goubault and M. Martel, Dagstuhl Seminar, LNCS 2991, Springer-Verlag, 2004.

http://www.di.ens.fr/~goubault/papers/SPutot_DagstuhlFinal.ps.gz

References

- Static Analysis of Numerical Algorithms, by E. Goubault and S. Putot, SAS 2006

http://www-ist.cea.fr/publicea/exl-php/200600004467-static-analysis-of-numerical-algorithms.html

- Modal Intervals revisited, by A. Goldsztejn, submitted to Reliable Computing

http://www.goldsztejn.com/publications/SubmittedRC2005.Goldsztejn.ModalIntervalsRevisitedPart2.pdf

- Under-approximations of computation in real numbers based on generalized affine arithmetic, by E. Goubault and S. Putot, SAS 2007

http://www.di.ens.fr/~goubault/papers/SAS07final.pdf

École CEA-EDF-INRIA Calcul numérique certifié

Travaux pratiques: Premiers pas avec MPFR

25-26 octobre 2007

1 Prise en main de MPFR

Problème 1.1 *Écrire un programme calculant* $\exp(1.5)$ *avec 150 bits de précision et affichant le résultat en décimal en arrondi au plus près. Augmentez la précision à 170 bits. Que remarquez-vous ?*

Faites varier la précision de calcul de 130 à 150 bits et affichez à chaque fois le résultat sur 41 chiffres décimaux en arrondi au plus près.

Problème 1.2 *Écrire un programme évaluant le polynôme de Rump :*

$$333.75b^6 + a^2(11a^2b^2 - b^6 - 121b^4 - 2) + 5.5b^8 + \frac{a}{2b}$$

en $a = 77617$ *et* $b = 33096$ *dans toutes les précisions de 17 à 128 bits (sans réordonner les termes), et affichant, pour chaque précision, le résultat en décimal en arrondi au plus près.*

Problème 1.3 *Écrire un programme calculant* 1.001^n *pour* n *allant de* 2 *à* 9 *et affichant le résultat sur 28 chiffres décimaux.*

Problème 1.4 *Écrire un programme calculant* $\sqrt[3]{7 + \sqrt[5]{2} - 5\sqrt[5]{8}} + \sqrt[5]{4} - \sqrt[5]{2}$ *en utilisant le mode d'arrondi au plus près. Votre programme pourra prendre en argument la précision des calculs intermédiaires.*

Problème 1.5 *Écrire un programme calculant* $\sqrt{\sin^2 x + \cos^2 x}$ *en utilisant le mode d'arrondi au plus près (il vous est juste interdit de simplifier mathématiquement). Votre programme prendra en argument la valeur de* x *et la précision des calculs intermédiaires. Testez votre programme sur diverses valeurs de* x*, y compris de grosses valeurs comme* 10^{17}*. Que remarquez-vous ?*

Problème 1.6 *Écrire un programme calculant e à l'aide de la formule* $e = \sum_{k=0}^{\infty} 1/k!$ *avec une erreur aussi petite que possible en moins en 2 secondes (cf annexe). Votre programme pourra prendre en argument un ou plusieurs paramètres (e.g. précision). Après avoir estimé l'erreur, vous pourrez comparer votre résultat avec celui de* `mpfr_exp` *et afficher une borne sur votre erreur, ainsi que le temps de calcul.*

Problème 1.7 *Écrire un programme calculant les termes de la suite :*

$$\begin{cases} u_0 & = & 2 \\ u_1 & = & -4 \\ u_{n+1} & = & 111 - \dfrac{1130}{u_n} + \dfrac{3000}{u_n u_{n-1}} \end{cases}$$

en utilisant le mode d'arrondi au plus près. Votre programme pourra prendre en argument : une précision globale et le nombre de termes à afficher. Que remarquez-vous ? Quelles sont les 11 premières décimales de u_{100} *?*

Problème 1.8 *Écrire un programme calculant les termes de la suite :*

$$\begin{cases} u_0 & = & e - 1 \\ u_n & = & n.u_{n-1} - 1 \end{cases}$$

en utilisant le mode d'arrondi au plus près. Quelles sont les 22 premières décimales de u_{998} *en utilisant une précision de 8586 bits pour tous les calculs intermédiaires ? Quelles sont les 19 premières décimales de* u_{998} *?*

Problème 1.9 *Reprendre les problèmes précédents en utilisant les modes d'arrondi dirigés pour fournir un encadrement du résultat exact.*

Problème 1.10 *Utilisez la méthode de Newton sur* $x^2 - x - 1 = 0$ *en partant de* $x = 1$ *pour calculer approximativement* n *bits du nombre d'or* φ*. Vérifier le résultat à l'aide de la formule* $\varphi = \frac{1+\sqrt{5}}{2}$ *; vous pourrez afficher la différence en ulp.*

2 Arrondi correct

Nous allons proposer deux méthodes pour implémenter une fonction simple avec arrondi correct : $f_n(x) = 1/\sqrt[n]{x} = x^{-1/n}$, où x est un nombre MPFR et n un entier de type `unsigned long`. Dans les deux cas, il faudra utiliser la stratégie de Ziv. Afin de pouvoir vérifier, vous calculerez ainsi $1717^{-1/17}$ avec 692 bits de précision en arrondi au plus près, et 693 dans les modes d'arrondi dirigés. Vous afficherez le résultat en base 2, sur 692 (resp. 693) chiffres.

Problème 2.1 *La fonction f_n sera implémentée avec arrondi correct par composition des deux fonctions $g_n(x) = \sqrt[n]{x}$ et $h(x) = 1/x$. Laquelle de ces deux fonctions g_n et h vaut-il mieux appliquer en premier ? Pourquoi ?*

Problème 2.2 *Implémenter la fonction f_n avec arrondi correct. Vous ne ferez ici pas d'analyse d'erreur : vous utiliserez simplement la monotonie des fonctions g_n et h, en faisant le calcul deux fois et en jouant sur les modes d'arrondi (en vous inspirant de l'arithmétique d'intervalles).*

Problème 2.3 *Implémenter la fonction f_n avec arrondi correct en utilisant la fonction* `mpfr_can_round` *décrite dans la section* Internals *du manuel de MPFR.*

Problème 2.4 *Modifier le code écrit pour chacune des deux méthodes de façon à retourner une valeur ternaire comme le fait MPFR. Quelles valeurs ternaires obtenez-vous sur l'exemple demandé ?*

3 Exceptions

Le mécanisme d'exceptions (implémenté par MPFR à l'aide de *flags* globaux) est utile pour écrire du code générique et faire une vérification *a posteriori* pour traiter les cas exceptionnels en réécrivant des expressions sous une autre forme.

Une fonction f est avec arrondi fidèle si pour tout x, l'évaluation de $f(x)$ retourne la valeur exacte $f(x)$ arrondie soit vers $-\infty$, soit vers $+\infty$.

Problème 3.1 *Implémenter la fonction $\sqrt{x+y}$ avec arrondi fidèle. Vous la testerez dans les précisions 32 bits et 128 bits en affichant le résultat en base 16, sur les exemples suivants, et en intervertissant x et y :*

x	y
144	145
2^{62}	$2^{32}+2$
$2^{emax}(1-2^{-34})$	17
$2^{emax}(1-2^{-34})$	2^{emax-3}
$2^{emin}(1-2^{-34})$	$-2^{emin}(1-2^{-17})$
$+\infty$	1
$+\infty$	$+\infty$

Problème 3.2 *Implémenter la fonction* $\log(\exp(x)/17-2)$ *avec arrondi fidèle. Vous la testerez dans les précisions 32 bits et 128 bits en affichant le résultat en base 16, sur les exemples suivants :* 4, 42, $-\infty$, $+\infty$, ± 0, 2^{32}, $2^{emax}(1-2^{-34})$, $\log 34$ *et* $\log 51$ *tous deux arrondis sur 512 bits vers $-\infty$ et vers $+\infty$.*

4 Émulation d'autres arithmétiques

Les fonctions `mpfr_set_emin`, `mpfr_set_emax` et `mpfr_subnormalize` peuvent servir à émuler des arithmétiques du style IEEE 754. La fonction `mpfr_check_range` peut également être utile.

Problème 4.1 *Implémenter* `void poly(mpfr_t y, mpfr_t x, int d, mpfr_t *a)` *qui évalue le polynôme* $y = P(x) = \sum_{i=0}^{d} a_i x^i$ *par le schéma de Horner, comme si vous aviez une arithmétique IEEE 754 en double précision. Bien entendu, on suppose que les valeurs* x *et* a_i *sont représentables en double précision IEEE.*

Problème 4.2 *Émuler les arithmétiques suivantes sur le code* `floor(x/y)` *(forme très souvent utilisée pour effectuer une division euclidienne lorsque l'on travaille sur des types flottants) :*

- *arithmétique IEEE 754 double précision ;*
- *idem avec précision intermédiaire étendue x86 (64 bits de mantisse), i.e. avec un double arrondi : d'abord sur 64 bits, puis sur 53 bits.*

Tester le code sur des exemples simples, ainsi que sur $x = 3 \times 2^{52} - 4$ *et* $y = 2^{52} - 1$.

Problème 4.3 *Reprendre le problème 4.1 pour cette fois implémenter une évaluation polynomiale avec arrondi correct pour le format double IEEE.*

Bien que la conversion base 2 – base 10 soit avec arrondi correct, le résultat d'un simple calcul n'est pas forcément l'arrondi correct du résultat mathématique, car MPFR calcule en base 2 (phénomène du double arrondi).

Problème 4.4 *Pour* p *et* q *entiers de type* `unsigned long`*, écrire un programme qui affiche la valeur de* p/q *correctement arrondie au plus près sur 6 chiffres en décimal. Y a-t-il des valeurs qui peuvent poser problème ? Tester sur* $500\,002\,499 \;/\; 499\,999\,999$.

5 Annexe : mesure du temps de calcul

Le temps de calcul (à la différence du temps réel) peut se mesurer avec la fonction `clock()`, déjà présente dans C89. Exemple :

```
#include <stdio.h>
#include <time.h>

int main (void)
{
  clock_t c;
  volatile int i;

  c = clock ();
  for (i = 0; i < 1000000000; i++)
    { }
  c = clock () - c;
  printf ("%g seconds\n", (double) c / CLOCKS_PER_SEC);
  return 0;
}
```

Note : sous Linux, tapez `man 7 time` pour avoir les différentes manières de mesurer le temps. Pour les calculs longs sur systèmes POSIX, il est préférable d'utiliser la fonction POSIX `times()` car sur les machines 32 bits, `clock()` reprend la même valeur au bout de 72 minutes environ.

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399