Certification of Numerical Analysis Programs
(CerPAN-FOST)

Micaela Mayero*
(Sylvie Boldo\textsuperscript{1}  François Clément\textsuperscript{2}  Jean-Christophe Filliâtre\textsuperscript{1})

\textsuperscript{1}INRIA Futurs/LRI, Proval team

\textsuperscript{2}INRIA Rocquencourt, Estime team

*INRIA, ENS-Lyon, Arnaire et Université Paris 13, LIPN UMR 7030-LCR team

http://www-lipn.univ-paris13.fr/CerPAN/
http://fost.saclay.inria.fr/

September 2009, EVA-Flo
Framework

- Work in progress (supported by ANR)
- **Numerical Problems**
- **Critical Programs**
- Automated technics: transports, money, medicine, seismology, meteorology, ...
- Reliability of these technics: zero default programs, certification
Difficulties

- These methods are limited by numerical aspects of problems
- Sources of errors (computers, schemas, algorithms, humans,...)
- **floating-point** numbers
- Formal proofs of programs soundness: problems from *continuum* (vs discret world)
  - successor of float
Objectives

- To develop methods to **formally prove correctness of programs** from NA domain
- Programs which are **often** used to solve critical problems
- **NA**: methods can be useful to develop critical numerical programs
- **FP**: continue the **development** of proof systems with **real numbers** (floating-point numbers, exacts reals)
- Open these new technics to **non experts** users
Method
- Floating-Point error
- Method error
- Errors and case study

Floating-point errors
- Methodology
- Into Caduceus
- Examples

Method error
- Methodology
- What do we have to prove?
- Difficulties
Proofs of programs

- Why, Caduceus, Gappa
  - Why: software verification platform (general-purpose verification condition generator)
  - Caduceus: verif. tool for C programs; built on top of Why.
  - Gappa: Génération Automatique de Preuves de Propriétés Arithmétiques. Tool intended to help verifying and formally proving properties on programs dealing with fp.

- Existing programs
- Annotations
- Proof Obligations (Coq,...)
Proofs of programs

- Why, Caduceus, Gappa
  - Why: software verification platform (general-purpose verification condition generator)
  - Caduceus: verification tool for C programs; built on top of Why.
  - Gappa: Génération Automatique de Preuves de Propriétés Arithmétiques. Tool intended to help verifying and formally proving properties on programs dealing with fp.

- Existing programs
- Annotations
- Proof Obligations (Coq,...)

1. To deal with floating-point numbers
2. Case study
3. Automation
Different kinds of error

Computing errors (due to computers):

- round errors: \((1 + 2^{-53}) - 1 = 0\)
- representation errors: \(\frac{1}{10}\)
- exceptional behavior: \(NaN\) \((+\infty - \infty)\)
Extraction

- Formal formalization of the problem
- Proofs
- Extraction: program proved to be correct
Extraction

- **Formal formalization** of the problem
- Proofs
- Extraction : program proved to be correct

1. To deal with real numbers *(which one?)*
2. Case study
3. Automation
4. Efficiency
Different kinds of error

**Method error** (known by the programmer and controlled):

It is the intrinsic error due to the algorithm with respect to the exact mathematical value:

- cut series ($\sum_{i=0}^{N} a_i$ instead of $\sum_{i=0}^{\infty} a_i$)
- approximations ($1 + x + \frac{x^2}{2}$ for $\exp(x)$)
- neglect some terms
- ...
To bound errors

- To bound the computing error:
  If $|x| \leq 2^{-3}$, then $|y - (1 + x + \frac{x^2}{2})| \leq 2^{-52}$

- To bound the computing error and the method error:
  If $|x| \leq 2^{-3}$, then $|y - \exp(x)| \leq 2^{-51}$

- To bound all errors ($x$ is an approximation of $X$):
  If $|X| \leq 2^{-3}$ and $|X - x| \leq 2^{-50}$, then $|y - (1 + X + \frac{X^2}{2})| \leq 2^{-48}$
Case study

The analytic gradient:

Minimization of function $J(P) = \frac{1}{2} \| d - F(P) \|^2$

where $d$ is an experimental measure and $F(P)$ is the theoretical function

$\Rightarrow \text{grad}(J(P)) = 0$
Case study

The analytic gradient:

Minimization of function $J(P) = \frac{1}{2}||d - F(P)||^2$

where $d$ is an experimental measure and $F(P)$ is the theoretical function

$\Rightarrow \text{grad}(J(P)) = 0$

Decomposition into two parts:

1. Resolution of the partial differentiation equation thanks to a numerical schema
   $\rightarrow$ Proof of schema stability

2. Computing of the derivative of a simple function:
   square of Euclidian norm on $\mathbb{R}^n$
Methodology

- Proof of program
- Caduceus
- Calculus error
- Floating-point numbers
The floating-point model inside Caduceus

Flottant Why (x) =
  flottant : x
  real : x@1
  real : x@2

- x is the value of the **floating-point number** in memory
- x@1 is the **value** if all calculus would have been exacts
- x@2 is the **ideal** result (including measure errors)
Examples of annotations

1. /* @ requires |x|<=2^(-3) @ ensures |\result - \result@1| <= 2^(-52) */
2. /* @ requires |x|<=2^(-3) @ set \result@1=exp(x@1) @ ensures |\result - \result@1| <= 2^(-51) */
3. /* @ requires |x@2|<=2^(-3) and |x-x@2|<=2^(-50) @ set \result@2=exp(x@2) @ ensures |\result - \result@2| <= 2^(-48) */
Malcolm algorithm (1/2)

Annotated program:

```c
/*@ logic int my_log(real s) */
/*@ ensures \result == 2 ^^ (53) */
double malcolm1() {
    double A;
    A=2;
   /*@ assert A==2 */
   /*@ invariant A== 2 ^^ my_log(A) && 1 <= my_log(A) <= 53
        variant (53-my_log(A)) */
    while (A != (A+1)) {
        A*=2;
    }
    return A;
}
```

Examples
Malcolm algorithm (2/2)

Proofs obligations:
(*Why goal*) Lemma malcolm1_impl_po_2 :
for all (A: double),
  for all (HW_2: A = (r_to_d nearest_even (IZR 2))),
  for all (HW_3:
    (* File "Malcolm.c", line 8, characters 14-18 *)
    (eq (d_to_r A) (IZR 2)),
    (* File "Malcolm.c", line 10, characters 17-73 *)
    ((eq (d_to_r A) (Rpower (IZR 2)
      (IZR (my_log (d_to_r A)))))) \ 1 <=
    (my_log (d_to_r A)) \ (my_log (d_to_r A)) <= 53).
...

CerPAN-FOST
**Method**

- Floating-point errors
  - Method error

**Examples**

### Dirichlet

```c
#include "dirichlet.h"
/*@ requires ni >= 2 && nk >= 2
   && 1 <= is < ni && 1 <= ir < ni
   && dx > 0. && dt > 0.
   && \valid_range(f,1,nk-1)
   && \valid_range(v,1,ni-1)
   && \forall int i; 1 <= i < ni => v[i] > 0.
   && \forall int i; 1 <= i < ni => v[i]*dt/dx < 1.
*/

double **forward_prop(int ni, int nk, int is, double dx,
                     double dt, double *f, double *v) {
  ...
  /*@ invariant 1 <= i <= ni variant ni-i */
  ...
Methodology

- **Direct specification**
- **Coq**
- “Algorithm error”
- **Known by the programmer**
What do we have to prove?

## Convergency

Wave equation:
\[ \frac{1}{c^2} \frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} = 0 \]

\( u \rightarrow u_h \) (approximated solution)

method error \( \equiv \) convergency

consistency \( \land \) stability \( \rightarrow \) convergency
What do we have to prove?

## Consistency

Explicit centered schema:

\[
\varepsilon_j^n = \frac{u_{h+1}^n - 2u_h^n + u_{h-1}^n}{\Delta t^2} - C^2 \frac{u_{h+1}^n - 2u_h^n + u_{h-1}^n}{\Delta x^2}
\]

2\(^{nd}\) order consistency:

\[
\varepsilon_j^n = O(\Delta t^2 + \Delta x^2)
\]
What do we have to prove?

**Stability**

Perturbative data: how is the solution changed?

On a bounded time interval:

\[ \| u_h^n \|_{L^2} \leq C (\| u_0 \|_{L^2} + t^n \| u_1 \|_{L^2}) \]
Difficulties

- **Consistency:**
  - Definition of $O$
  - Imprecise notations (ex: $O(\Delta x^2 + \Delta t^2)$)
  - To swap quantifiers
  - Mixed time and space problems
  - Uniform Taylor series

- **Stability:**
  - 2 methods (at least):
    - Fourier
    - Energetic technics
<table>
<thead>
<tr>
<th>Method</th>
<th>Floating-point errors</th>
<th>Method error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Difficulties**

**Conclusion**

- Separated treatment of the 2 errors: floating-point and method error
- Addition of floating-point numbers into Caduceus
- Proof of floating-point error
- Addition of a new tactic “gappa” into Coq
- Specification of method error
- FOST (Formal proOfs of Scientific compuTation programs)