RAIM 2025 – A Tribute to Jean-Michel Muller
Lyon

Mechanical and User-friendly Lemmas for Lagrange finite elements and floating-point Errors in Rocq

Sylvie Boldo (she/her)

Inria, Université Paris-Saclay

November 5th, 2025

This title could have been

Justifying Error Analysis of Numerics –

Mechanical Inference in Coq about Hardware Execution and Limits

This title could have been

Justifying Error Analysis of Numerics –

Mechanical Inference in Coq about Hardware Execution and Limits

$\hookrightarrow$ Mechanical and User-friendly Lemmas for Lagrange finite elements and floating-point Errors in Rocq

This title could have been

Justifying **E**rror **A**nalysis of **N**umerics –
Mechanical **I**nference in **C**oq about **H**ardware
Execution and **L**imits

$\hookrightarrow$ **M**echanical and **U**ser-friendly **L**emmas for
Lagrange finite elements and floating-point **E**rrors in
Rocq

Outline

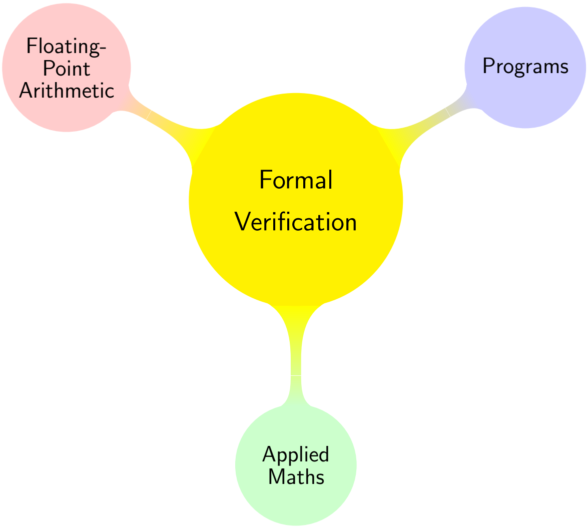
1. inTroduction
2. formalization of computer aritHmetic
3. proofs of floAting-poiNt programs
4. checKing applied mathS programs
5. conclusion and perspectives

Common work with

This is common work with

- François Clément
- Jean-Christophe Filliâtre
- Vincent Martin
- Micaela Mayero
- Guillaume Melquiond
- Houda Mouhcine
- Jean-Michel Muller

Introduction



Motivations 1/2: Why Floating-Point Arithmetic?

Motivations 1/2: Why Floating-Point Arithmetic?

1. See other talks.

Motivations 1/2: Why Floating-Point Arithmetic?

1. See other talks.

2. A long time ago, in an office not far away, Jean-Michel was my tutor and told me about proving that, even when roundings occur:

$$\boxed{-1 \leq \frac{x}{\sqrt{x^2 + y^2}} \leq 1}$$

Motivations 1/2: Why Floating-Point Arithmetic?

1. See other talks.

2. A long time ago, in an office not far away, Jean-Michel was my tutor and told me about proving that, even when roundings occur:

$$\boxed{-1 \leq \frac{x}{\sqrt{x^2+y^2}} \leq 1}$$

Spoiler: this holds when the precision is greater than 2 whatever the radix.
Note that strange things happen when the radix $\beta \neq 2$.
With $\beta = 10$ and $p = 4$, if $x = 31.66$, then $\circ\left(\sqrt{\circ\,(x^2)}\right) = 31.65$.
With $\beta = 1000$ and $p = 2$, if $x = 31.662$, then $\circ\left(\sqrt{\circ\,(x^2)}\right) = 31.654$.

Motivations 2/2: Why Formal Verification?

(some parts are $\boxed{\text{hidden}}$ for confidentiality purpose.)

Motivations 2/2: Why Formal Verification?

(some parts are $\boxed{\text{hidden}}$ for confidentiality purpose.)

On Thu, 14 Jan 2016 15:37, Sylvie Boldo wrote:

> In $\boxed{\text{Paper}}$, at the top of $\boxed{\text{page}}$ at $\boxed{\text{line}}$, the $\boxed{\text{inequality } i \text{ out of } n}$

> is wrong because $\boxed{[...]}$.

Motivations 2/2: Why Formal Verification?

(some parts are $\boxed{\text{hidden}}$ for confidentiality purpose.)

Subject :	Re: MOUHAHAHA !
Date :	Thu, 14 Jan 2016 15:43:54 +0100
From:	A very well-known researcher, co-author of Paper
To:	me
Cc:	Other remarkable researchers, co-authors of Paper

You are right. I hate you.

[...]

On Thu, 14 Jan 2016 15:37, Sylvie Boldo wrote:
> In Paper, at the top of page at line, the inequality i out of n
> is wrong because [...].

Outline

Formal proof

The proof is checked in its deep details until the computer agrees with it.

We often use formal proof checkers, meaning programs that only **check** a proof (they may also generate easy demonstrations).

Therefore the checker is a very short program (de Bruijn criteria: the correctness of the system as a whole depends on the correctness of a very small "kernel").

Examples: Rocq (ex-Coq), Lean, HOL Light, Isabelle-HOL, Mizar, PVS.

A Coq formalization of FP arithmetic: Flocq

Flocq: 66 000 lines of Rocq, 2,200 theorems,

- any radix, any format,
- both axiomatic and computable definitions of rounding,
- effective arithmetic operators,
- interface with Rocq primitive floating-point numbers,
- numerous theorems.

A Coq formalization of FP arithmetic: Flocq

Flocq: 66 000 lines of Rocq, 2,200 theorems,

- any radix, any format,
- both axiomatic and computable definitions of rounding,
- effective arithmetic operators,
- interface with Rocq primitive floating-point numbers,
- numerous theorems.

Applications:

- Frama-C/Jessie — C code certifier
- CompCert — certified C compiler

`https://flocq.gitlabpages.inria.fr/`

Flocq generic format

A FP format is only characterized by a function $\varphi : \mathbb{Z} \to \mathbb{Z}$.

Flocq generic format

A FP format is only characterized by a function $\varphi : \mathbb{Z} \to \mathbb{Z}$.

For $x \in \mathbb{R}$, we compute e such that $\beta^{e-1} \leq |x| < \beta^{e}$.
Then x is in the format iff

$$x = \left\lfloor x \beta^{-\varphi(e)} \right\rfloor \beta^{\varphi(e)}$$

In other words: if it can be written with exponent $\varphi(e)$.

Usual Formats

Definition (FIX)

Fixed-point format with exponent $e_{\min}$: $\varphi(e) = e_{\min}$.

Usual Formats

Definition (FIX)

Fixed-point format with exponent $e_{\min}$: $\varphi(e) = e_{\min}$.

Definition (FL*)

Floating-point format with precision p:

- unbounded (FLX): $\varphi(e) = e - p$,

Usual Formats

Definition (FIX)

Fixed-point format with exponent e_{min}: $\varphi(e) = e_{\text{min}}$.

Definition (FL*)

Floating-point format with precision p:

- unbounded (FLX): $\varphi(e) = e - p$,
- bounded with subnormal numbers (FLT): $\varphi(e) = \max(e - p, e_{\text{min}})$,

Usual Formats

Definition (FIX)

Fixed-point format with exponent e_{min}: $\varphi(e) = e_{\text{min}}$.

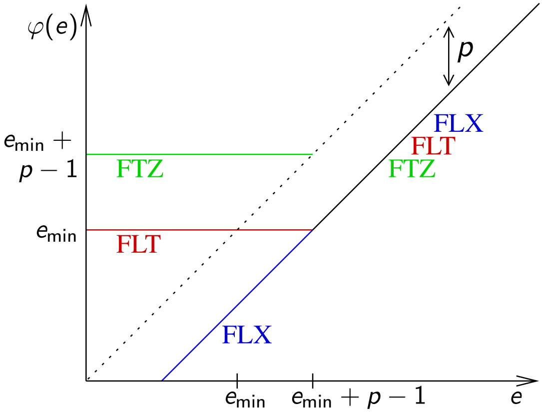
Definition (FL*)

Floating-point format with precision p:

- unbounded (FLX): $\varphi(e) = e - p$,
- bounded with subnormal numbers (FLT): $\varphi(e) = \max(e - p, e_{\text{min}})$,
- bounded without subnormal numbers (FTZ):
  $\phi(e) = \begin{cases} e - p & \text{si } e - p \geq e_{\text{min}}, \\ e_{\text{min}} + p - 1 & \text{sinon.} \end{cases}$

A random φ may not allow to define a rounding: we have a `valid` predicate for being a reasonable φ.

Usual Floating-Point Formats



Outline

Annotation language: ACSL

- ANSI/ISO C Specification Language

Annotation language: ACSL

- ANSI/ISO C Specification Language
- behavioral specification language for C programs

Annotation language: ACSL

- ANSI/ISO C Specification Language
- behavioral specification language for C programs
- pre-conditions and post-conditions to functions
  (and which variables are modified).

Annotation language: ACSL

- ANSI/ISO C Specification Language
- behavioral specification language for C programs
- pre-conditions and post-conditions to functions
(and which variables are modified).
- variants and invariants of the loops.

Annotation language: ACSL

- ANSI/ISO C Specification Language
- behavioral specification language for C programs
- pre-conditions and post-conditions to functions
  (and which variables are modified).
- variants and invariants of the loops.
- assertions

Annotation language: ACSL

- ANSI/ISO C Specification Language
- behavioral specification language for C programs
- pre-conditions and post-conditions to functions
  (and which variables are modified).
- variants and invariants of the loops.
- assertions
- In annotations, all computations are exact.

Methodology for the verification of C programs

C Program

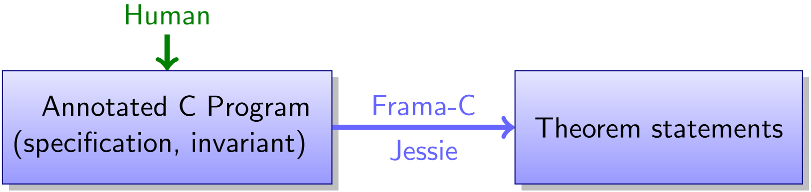
Methodology for the verification of C programs

Human

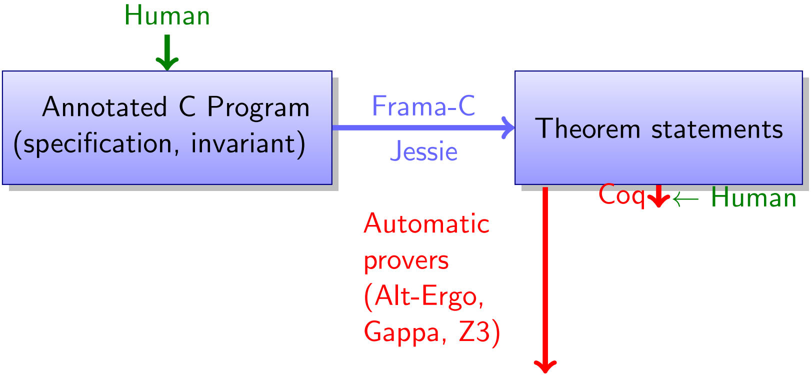
↓

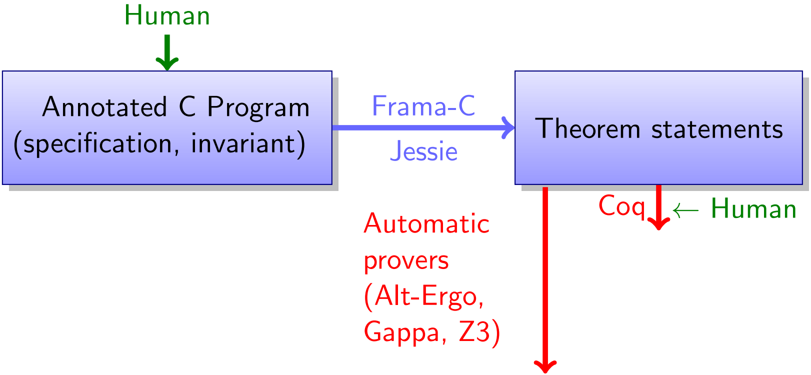
Annotated C Program
(specification, invariant)

Methodology for the verification of C programs

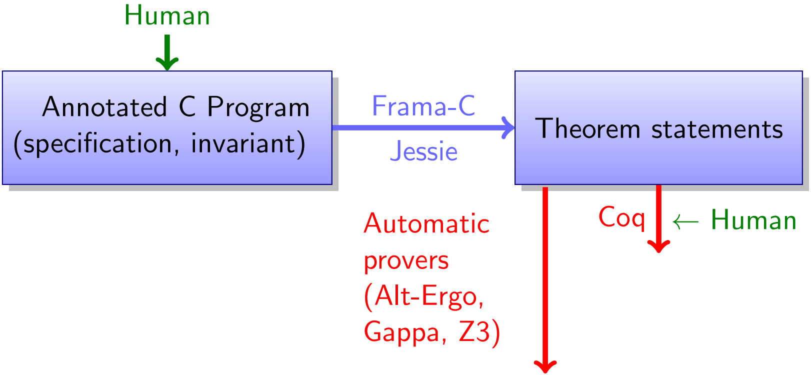


Methodology for the verification of C programs

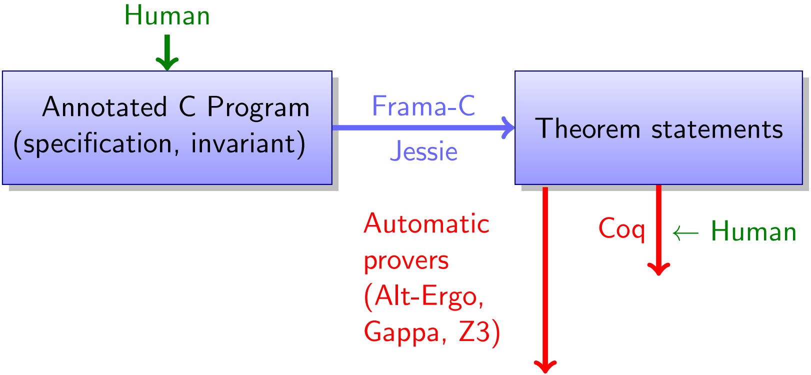



Human
Annotated C Program
(specification, invariant)
Frama-C
Jessie
Theorem statements
Coq
← Human
Automatic
provers
(Alt-Ergo,
Gappa, Z3)

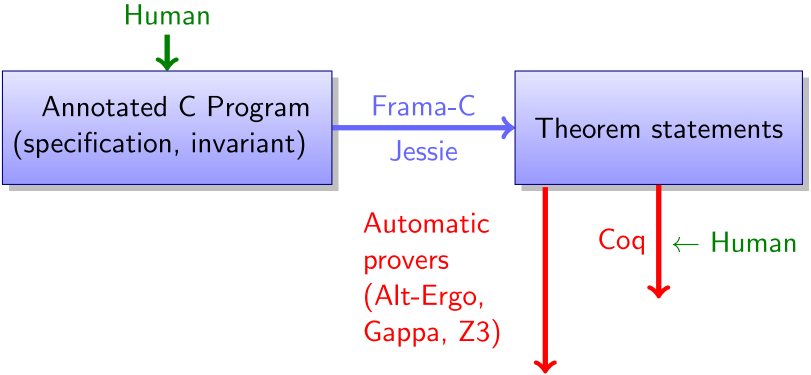
Methodology for the verification of C programs



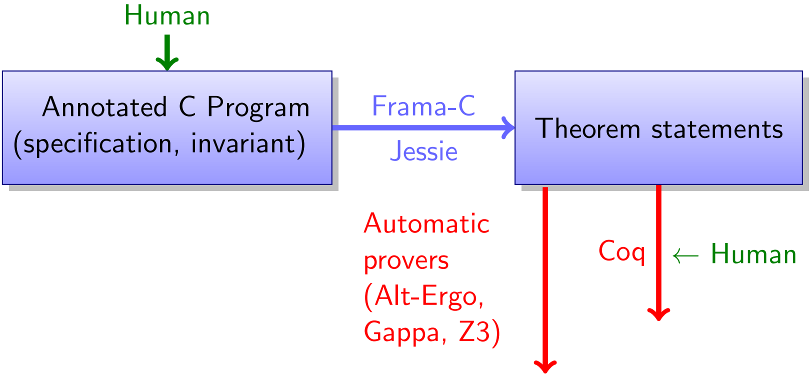
Methodology for the verification of C programs



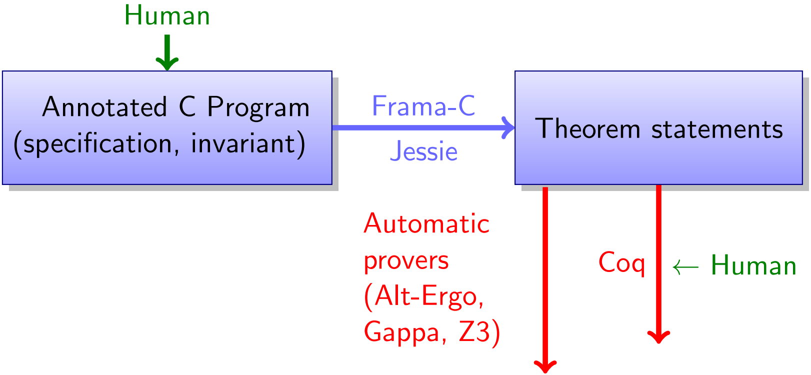
Methodology for the verification of C programs



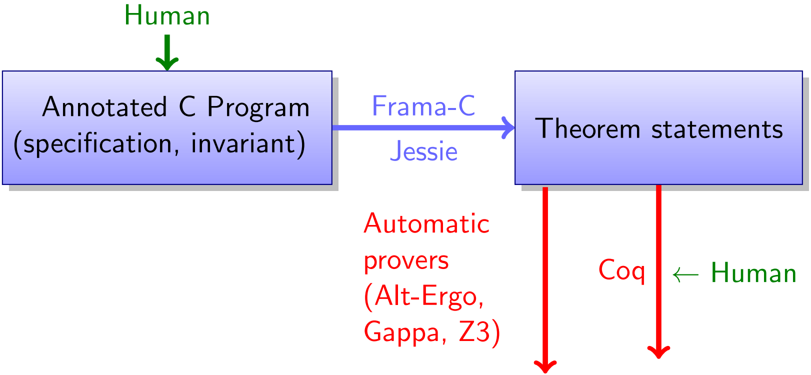
Methodology for the verification of C programs



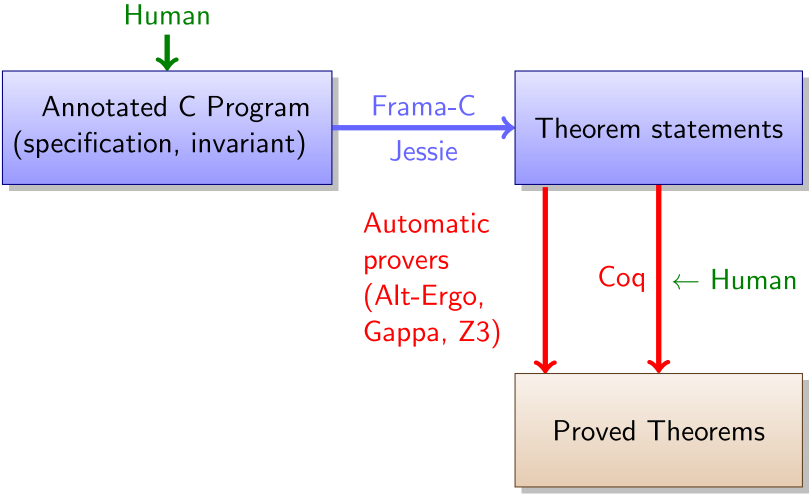
Methodology for the verification of C programs



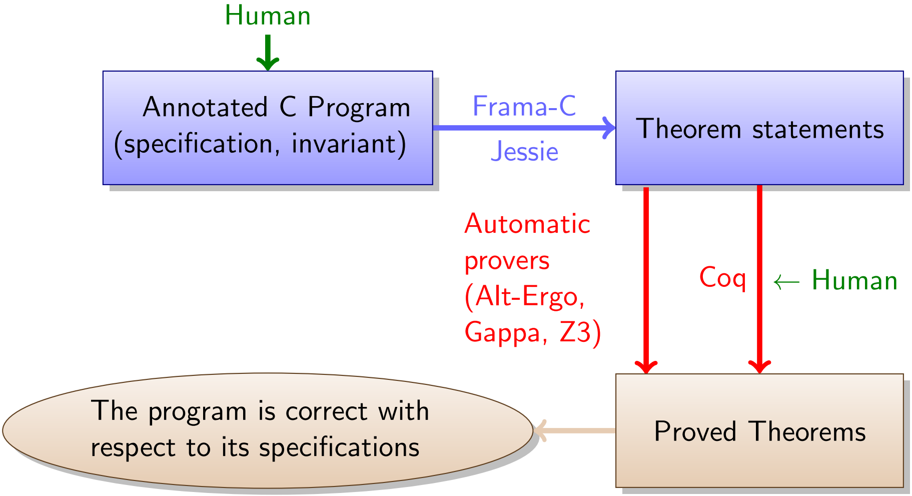
Methodology for the verification of C programs



Methodology for the verification of C programs



Methodology for the verification of C programs



Accurate Discriminant

It is known that computing accurately $b^2 - ac$ is not straightforward.

Accurate Discriminant

It is known that computing accurately $b^2 - ac$ is not straightforward.

Theorem (Kahan)

Provided there is neither Overflow, nor Underflow, ther is an algorithm that computes $b^2 - a \times c$ within 2 ulps.

Accurate Discriminant – Program

```
/*@ requires (b==0.  || 0x1.p-916 <= \abs(b*b)) &&
  @           (a*c==0. || 0x1.p-916 <= \abs(a*c)) &&
  @           \abs(b) <= 0x1.p510 &&
  @           \abs(a) <= 0x1.p995 && \abs(c) <= 0x1.p995 &&
  @           \abs(a*c) <= 0x1.p1021;
  @ ensures   \result ==0. || \abs(\result -(b*b-a*c)) <= 2.*ulp(\result);
  @ */
double discriminant(double a, double b, double c) {
  double p,q,d,dp,dq;
  p=b*b;
  q=a*c;

  if (p+q <= 3*fabs(p-q))
    d=p-q;
  else {
    dp=Dekker(b,b,p);
    dq=Dekker(a,c,q);
    d=(p-q)+(dp-dq);
  }
  return d;
}
```

Accurate Discriminant – Program

```
/*@ requires (b==0.   ||  0x1.p-916 <= \abs(b*b)) &&
  @          (a*c==0. || 0x1.p-916 <= \abs(a*c)) &&
  @          \abs(b)  <= 0x1.p510 &&
  @          \abs(a)  <= 0x1.p995 && \abs(c) <= 0x1.p995 &&
  @          \abs(a*c)  <= 0x1.p1021;
  @ ensures  \result ==0. || \abs(\result -(b*b-a*c)) <= 2.*ulp(\result);
  @ */
double discriminant(double a, double b, double c) {
  double p,q,d,dp,dq;
  p=b*b;
  q=a*c;

  if (p+q <= 3*fabs(p-q))
    d=p-q;
  else {
    dp=Dekker(b,b,p);
    dq=Dekker(a,c,q);
    d=(p-q)+(dp-dq);
  }
  return d;
}
```

Underflow

Accurate Discriminant – Program

```
/*@ requires (b==0.  || 0x1.p-916 <= \abs(b*b)) &&
  @          (a*c==0. || 0x1.p-916 <= \abs(a*c)) &&
  @          \abs(b) <= 0x1.p510 &&
  @          \abs(a) <= 0x1.p995 && \abs(c) <= 0x1.p995 &&        Overflow
  @          \abs(a*c) <= 0x1.p1021;
  @ ensures  \result==0. || \abs(\result-(b*b-a*c)) <= 2.*ulp(\result);
  @ */
double discriminant(double a, double b, double c) {
  double p,q,d,dp,dq;
  p=b*b;
  q=a*c;

  if (p+q <= 3*fabs(p-q))
    d=p-q;
  else {
    dp=Dekker(b,b,p);
    dq=Dekker(a,c,q);
    d=(p-q)+(dp-dq);
  }
  return d;
}
```

Accurate Discriminant – Program

```
/*@ requires (b==0.   || 0x1.p-916 <= \abs(b*b)) &&
  @           (a*c==0. || 0x1.p-916 <= \abs(a*c)) &&
  @           \abs(b) <= 0x1.p510 &&
  @           \abs(a) <= 0x1.p995 && \abs(c) <= 0x1.p995 &&
  @           \abs(a*c) <= 0x1.p1021;
  @ ensures  \result ==0. || \abs(\result -(b*b-a*c)) <= 2.*ulp(\result);
  @ */
double discriminant(double a, double b, double c) {
  double p,q,d,dp,dq;
  p=b*b;
  q=a*c;

  if (p+q <= 3*fabs(p-q))
    d=p-q;
  else {
    dp=Dekker(b,b,p);
    dq=Dekker(a,c,q);
    d=(p-q)+(dp-dq);
  }
  return d;
}
```

2 ulps

Accurate Discriminant – Program

```
/*@ requires (b==0.   || 0x1.p-916 <= \abs(b*b)) &&
  @          (a*c==0. || 0x1.p-916 <= \abs(a*c)) &&
  @          \abs(b)  <= 0x1.p510 &&
  @          \abs(a)  <= 0x1.p995 && \abs(c) <= 0x1.p995 &&
  @          \abs(a*c)  <= 0x1.p1021;
  @ ensures  \result ==0. || \abs(\result -(b*b-a*c)) <= 2.*ulp(\result);
  @ */
double discriminant(double a, double b, double c) {
  double p,q,d,dp,dq;
  p=b*b;
  q=a*c;

  if (p
    d=p-q;
  else {
    dp=Dekker(b,b,p);
    dq=Dekker(a,c,q);
    d=(p-q)+(dp-dq);
  }
  return d;
}
```

Function calls

⇒ pre-conditions to be guaranteed

⇒ guaranteed post-conditions

$p + dp = b^2$ and $q + dq = ac$

Accurate Discriminant – Program

```
/*@ requires  (b==0.   ||  0x1.p-916 <= \abs(b*b)) &&
  @           (a*c==0. || 0x1.p-916 <= \abs(a*c)) &&
  @           \abs(b)  <= 0x1.p510 &&
  @           \abs(a)  <= 0x1.p995 && \abs(c) <= 0x1.p995 &&
  @           \abs(a*c)  <= 0x1.p1021;
  @ ensures   \result ==0 ...(b*b-a*c)) <= 2.*ulp(\result);
  @ */
double discrimi... {
  double p,q,d...
  p=b*b;
  q=a*c;

  if (p+q <= 3*fabs(p-q))
    d=p-q;
  else {
    dp=Dekker(b,b,p);
    dq=Dekker(a,c,q);
    d=(p-q)+(dp-dq);
  }
  return d;
}
```

In the initial proof,
the test was assumed
correct.

⇒ Additional proof when the test is incorrect.

Outline

Motivations

𝔐𝔞𝔱𝔥𝔢𝔪𝔞𝔱𝔦𝔠𝔰

$\mathbb{R}$, $\int$, $\frac{\partial^2 u}{\partial t^2}$
theorems

Motivations

$\mathfrak{Mathematics}$	$\mathbb{R}$, $\int$, $\frac{\partial^2 u}{\partial t^2}$ theorems
Applied Mathematics	numerical scheme, convergence algorithms + theorems

Motivations

Mathematics	$\mathbb{R}$, $\int$, $\frac{\partial^2 u}{\partial t^2}$ theorems
Applied Mathematics	numerical scheme, convergence algorithms + theorems
Computer Science	floating-point numbers, implementation programs + ?

Motivations

Mathema | $\mathbb{R}$, $\int$, $\frac{\partial^2 u}{\partial t^2}$
theorems

Applied M | me, convergence

Computer Science | float entation
programs +

Formal Proof

Motivations

PDE (Partial Differential Equations) $\Rightarrow$ weather forecast
$\Rightarrow$ nuclear simulation
$\Rightarrow$ optimal control
$\Rightarrow$ ...

Motivations

PDE (Partial Differential Equations) $\Rightarrow$ weather forecast
$\Rightarrow$ nuclear simulation
$\Rightarrow$ optimal control
$\Rightarrow$ ...

Too complex to be solved by an exact formula
$\Rightarrow$ approximated by numerical schemes on meshes

$\Rightarrow$ mathematical proof of the convergence of the numerical scheme
(one computes values nearer to the solution when the mesh size decreases)

Motivations

PDE (Partial Differential Equations)	$\Rightarrow$	weather forecast
	$\Rightarrow$	nuclear simulation
	$\Rightarrow$	optimal control
	$\Rightarrow$	...

Too complex to be solved by an exact formula
$\Rightarrow$ approximated by numerical schemes on meshes

$\Rightarrow$ mathematical proof of the convergence of the numerical scheme
(one computes values nearer to the solution when the mesh size decreases)

$\Rightarrow$ real program that implements this scheme or this method

Motivations

PDE (Partial Differential Equations) $\Rightarrow$ weather forecast
$\Rightarrow$ nuclear simulation
$\Rightarrow$ optimal control
$\Rightarrow$ ...

Too complex to be solved by an exact formula
$\Rightarrow$ approximated by numerical schemes on meshes

$\Rightarrow$ mathematical proof of the convergence of the numerical scheme
(one computes values nearer to the solution when the mesh size decreases)

$\Rightarrow$ real program that implements this scheme or this method

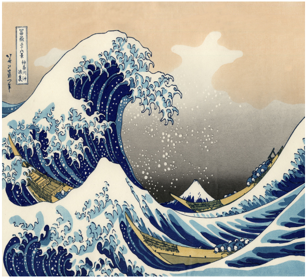
$\Rightarrow$ **towards verifying these programs!**
$\Rightarrow$ rocq-num-analysis library

The wave equation

Looking for $u : \mathbb{R}^2 \to \mathbb{R}$ regular enough such that:

$$\frac{\partial^2 u(x,t)}{\partial t^2} - c^2 \frac{\partial^2 u(x,t)}{\partial x^2} = s(x,t)$$

with given values for the initial position $u_0(x)$ and initial velocity $u_1(x)$.



$\Rightarrow$ rope oscillation, sound, radar, oil prospection...

Scheme?

We want $u_j^k \approx u(j\Delta x, k\Delta t)$.

$$\frac{u_j^k - 2u_j^{k-1} + u_j^{k-2}}{\Delta t^2} - c^2 \frac{u_{j+1}^{k-1} - 2u_j^{k-1} + u_{j-1}^{k-1}}{\Delta x^2} = s_j^{k-1}$$

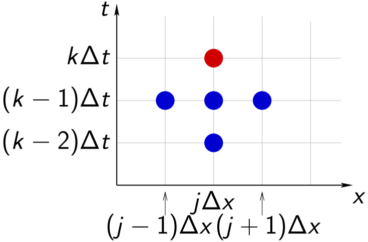
And other horrible formulas to initialize u_j^0 and u_j^1.

Scheme?

We want $u_j^k \approx u(j\Delta x, k\Delta t)$.

$$\frac{u_j^k - 2u_j^{k-1} + u_j^{k-2}}{\Delta t^2} - c^2 \frac{u_{j+1}^{k-1} - 2u_j^{k-1} + u_{j-1}^{k-1}}{\Delta x^2} = s_j^{k-1}$$

And other horrible formulas to initialize u_j^0 and u_j^1.



Three-point scheme: u_j^k depends on u_{j-1}^{k-1}, u_j^{k-1}, u_{j+1}^{k-1} and u_j^{k-2}.

Program

```
// initialization of p[i][0] and p[i][1]
for (k=1; k<nk; k++) {
  p[0][k+1] = 0.;
  for (i=1; i<ni; i++) {
    dp = p[i+1][k] - 2.*p[i][k] + p[i-1][k];
    p[i][k+1] = 2.*p[i][k] - p[i][k-1] + a*dp;
   }
  p[ni][k+1] = 0.;
}
```

Program

```
// initialization of p[i][0] and p[i][1]
for (k=1; k<nk; k++) {
  p[0][k+1] = 0.;
  for (i=1; i<ni; i++) {
    dp = p[i+1][k] - 2.*p[i][k] + p[i-1][k];
    p[i][k+1] = 2.*p[i][k] - p[i][k-1] + a*dp;
   }
  p[ni][k+1] = 0.;
}
```

Two different errors:

- round-off errors

  due to floating-point roundings

- method errors

  the scheme only approximates the exact solution

Rounding error

With ε_i^k the local error at each step, we have:

$$p_i^k - exact(p_i^k) = \sum_{l=0}^{k} \sum_{j=-l}^{l} \alpha_j^l \, \varepsilon_{i+j}^{k-l}$$

1. We have an analytical expression of the rounding error with known constants α_i^k.

Rounding error

With ε_i^k the local error at each step, we have:

$$p_i^k - exact(p_i^k) = \sum_{l=0}^{k} \sum_{j=-l}^{l} \alpha_j^l \, \varepsilon_{i+j}^{k-l}$$

1. We have an analytical expression of the rounding error with known constants α_i^k.
2. It is not that complicated!
   (we cannot get rid of the pyramidal double summation)

Rounding error

With ε_i^k the local error at each step, we have:

$$p_i^k - exact(p_i^k) = \sum_{l=0}^{k} \sum_{j=-l}^{l} \alpha_j^l \, \varepsilon_{i+j}^{k-l}$$

1. We have an analytical expression of the rounding error with known constants α_i^k.
2. It is not that complicated!
   (we cannot get rid of the pyramidal double summation)
3. The rounding error is bounded by $\bigcirc(k^2 \, 2^{-53})$:

$$\left| \mathbf{p_i^k} - \mathbf{exact}\left(\mathbf{p_i^k}\right) \right| \leq \mathbf{78} \times \mathbf{2^{-53}} \times (\mathbf{k}+\mathbf{1}) \times (\mathbf{k}+\mathbf{2})$$

Convergence

We proved that:

$$\left\| e_h^{k_{\Delta t}(t)} \right\|_{\Delta x} = O_{\substack{t \in [0, t_{\max}] \\ (\Delta x, \Delta t) \to 0 \\ 0 < \Delta x \;\wedge\; 0 < \Delta t \;\wedge \\ \zeta \leq c \frac{\Delta t}{\Delta x} \leq 1 - \xi}} (\Delta x^2 + \Delta t^2).$$

with a uniform big O and a naive proof (consistency, stability by energy)

Note that the constants hidden in the big O may be extracted.

Program verification

- 154 lines of annotations for 32 lines of C
- 150 verification conditions:
  - 44 about the behavior
  - 106 about the safety (runtime errors)

Program verification

- 154 lines of annotations for 32 lines of C
- 150 verification conditions:
  - 44 about the behavior
  - 106 about the safety (runtime errors)
- About 90 % of the safety goals (matrix access, Overflow, and so on) are proved automatically.
- 33 theorems are interactively proved using Coq for a total of about 15,000 lines of Coq and 30 minutes of compilation.

Type of proofs	Nb spec lines	Nb lines	Compilation time
Convergence	991	5 275	42 s
Round-off + runtime errors	7 737	13 175	32 min

Let us think bigger!

http://www.ima.umn.edu/~arnold/disasters/sleipner.html

The sinking of the Sleipner A offshore platform

Excerpted from a report of SINTEF, Civil and Environmental Engineering:

The Sleipner A platform produces oil and gas in the North Sea and is supported on the seabed at a water depth of 82 m. It is a Condeep type platform with a concrete gravity base structure consisting of 24 cells and with a total base area of 16 000 m^2. Four cells are elongated to shafts supporting the platform deck. The first concrete base structure for Sleipner A sprang a leak and sank under a controlled ballasting operation during preparation for deck mating in Gandsfjorden outside Stavanger, Norway on 23 August 1991.

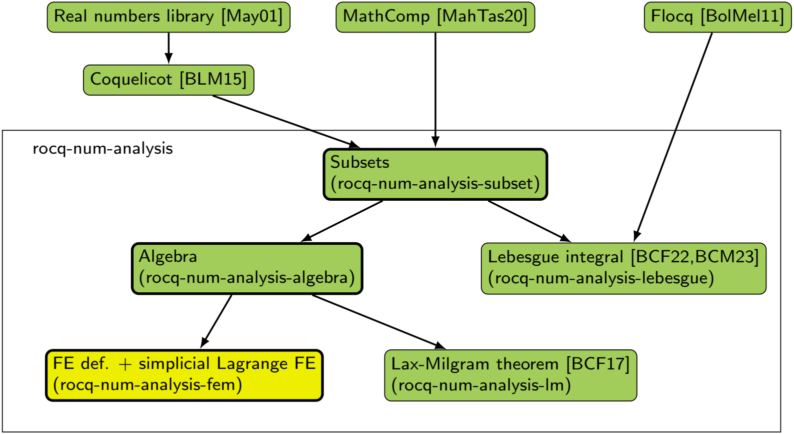
Immediately after the accident, the owner of the platform, Statoil, a Norwegian oil company appointed an investigation group, and SINTEF was contracted to be the technical advisor for this group.

The investigation into the accident is described in 16 reports...

The conclusion of the investigation was that the loss was caused by a failure in a cell wall, resulting in a serious crack and a leakage that the pumps were not able to cope with. The wall failed as a result of a combination of a serious error in the finite element analysis and insufficient anchorage of the reinforcement in a critical zone.

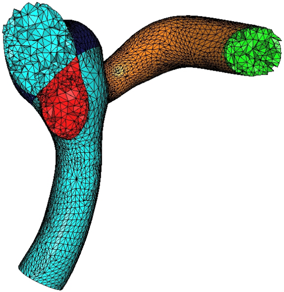
A better idea of what was involved can be obtained from this photo and sketch of the platform. The top deck weighs 57,000 tons, and provides accommodation for about 200 people and support for drilling equipment weighing about 40,000 tons. When the first model sank in August 1991, the crash

State of the lib: rocq-num-analysis



(155 files, 85 kLoC)

Mesh



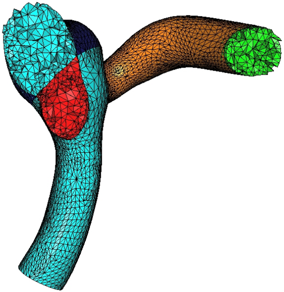
@ V. Martin

On a complex geometry, one has a mesh, made of triangles / tetrahedrons.

The Finite Element Method aims to approximate the PDE on each element, while constructing a continuous solution.

A Finite Element is a geometric element + ??? to solve the PDE on it.

Mesh



@ V. Martin

On a complex geometry, one has a mesh, made of triangles / tetrahedrons.

The Finite Element Method aims to approximate the PDE on each element, while constructing a continuous solution.

A Finite Element is a geometric element + ??? to solve the PDE on it.

A **Finite Element** is how to approximate a function (solution of the PDE) by a simpler function (eg. polynomial) on a small simple geometric cell (eg. tetrahedron), by retaining particular values (eg. point values, mean values, fluxes through interfaces) to reduce to a finite-dimension problem.

Definition of a Finite Element

A Finite Element is a triple $(\mathcal{K}, \mathcal{P}, \Sigma)$:

1. $\mathcal{K}$ is a geometric cell, such as a simplex (segment in $\mathbb{R}$, triangle in $\mathbb{R}^2$, tetrahedron in $\mathbb{R}^3$)
2. $\mathcal{P}$ is a vector space of functions on $\mathcal{K}$, of dimension n_{dof} (often polynomials with bounded degree)
3. Σ is composed of n_{dof} linear forms on $\mathcal{P}$.

A Finite Element must have the unisolvence property:
For $\Sigma := \{\sigma_i\}_{i \in \{0:n_{dof}-1\}}$, let $\Phi_\Sigma : \mathcal{P} \to \mathbb{R}^{n_{dof}}$ be
$\Phi_\Sigma(p) := (\sigma_i(p))_{i \in \{0:n_{dof}-1\}}$. Unisolvence means that Φ_Σ is a bijection.

Corresponding Rocq Definition

A FE is either a simplex or a cuboid with the correct number of vertices:

```
Inductive shape_type := Simplex | Cuboid.

Definition nvtx_of_shape (d : ℕ) (shp : shape_type) : ℕ :=
  match shp with Simplex ⇒ d.+1 | Cuboid ⇒ 2^d end.
```

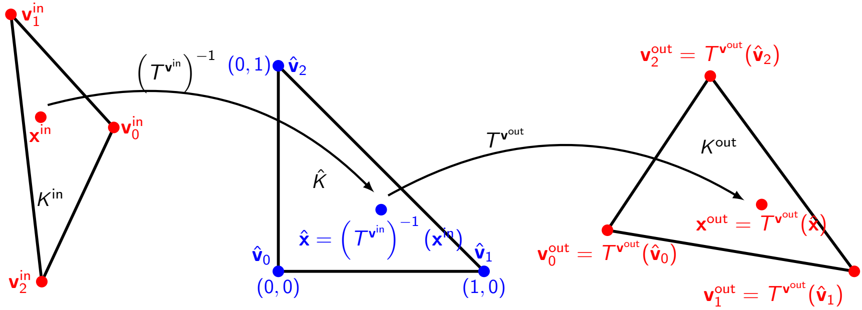
Corresponding Rocq Definition

A FE is either a simplex or a cuboid with the correct number of vertices:

```
Inductive shape_type := Simplex | Cuboid.

Definition nvtx_of_shape (d : ℕ) (shp : shape_type) : ℕ :=
  match shp with Simplex ⇒ d.+1 | Cuboid ⇒ 2^d end.
```

```
Record FE (d : ℕ) : Type := mk_FE {
  shape : shape_type;
  nvtx : ℕ := nvtx_of_shape d shape;
  K_vertices : [0..nvtx) → 'ℝ^d;
  K_geom : 'ℝ^d → Prop := convex_hull K_vertices;(* geometric element K
  ndof : ℕ;
  P_approx : FRd d → Prop;                    (* approximation space P *)
  P_approx_has_dim : has_dim P_approx ndof;
  S_dof : [0..ndof) → FRd d → ℝ;               (* degrees of freedom Σ *)
  S_dof_lm : ∀ i, lin_map (S_dof i);
  unisolvence_inj : KerS0 P_approx (gather S_dof);
}.
```

Geometric Transformation of a Finite Element



From an input finite element and the output vertices $\mathbf{v}^{\text{out}}$, we can build an output finite element, relying on the geometric transformation $T_{\mathbf{v}}$ (based on Lagrange polynomials).

Instantiation!

Let us define from scratch a FE to check there is no inconsistency.

We build a d-FE depending on a variable $k \in \mathbb{N}$. So we need:

Instantiation!

Let us define from scratch a FE to check there is no inconsistency.

We build a d-FE depending on a variable $k \in \mathbb{N}$. So we need:

- a shape: `Simplex`, ✓

Instantiation!

Let us define from scratch a FE to check there is no inconsistency.

We build a d-FE depending on a variable $k \in \mathbb{N}$. So we need:

- a shape: `Simplex`, ✓
- vertices: the right triangle/tetrahedron/... of dimension d with sides of length 1, ✓

Instantiation!

Let us define from scratch a FE to check there is no inconsistency.

We build a d-FE depending on a variable $k \in \mathbb{N}$. So we need:

- a shape: `Simplex`, ✓
- vertices: the right triangle/tetrahedron/... of dimension d with sides of length 1, ✓
- `P_approx` is the vector space of polynomials from $\mathbb{R}^d$ to $\mathbb{R}$ of total degree $\leq k$, ✓

Instantiation!

Let us define from scratch a FE to check there is no inconsistency.

We build a d-FE depending on a variable $k \in \mathbb{N}$. So we need:

- a shape: `Simplex`, ✓
- vertices: the right triangle/tetrahedron/... of dimension d with sides of length 1, ✓
- `P_approx` is the vector space of polynomials from $\mathbb{R}^d$ to $\mathbb{R}$ of total degree $\leq k$, ✓
- `S_dof` are the linear forms that take a function and evaluate it at given points for evenly distributed points (called Lagrange nodes), ✓

Instantiation!

Let us define from scratch a FE to check there is no inconsistency.

We build a d-FE depending on a variable $k \in \mathbb{N}$. So we need:

- a shape: `Simplex`, ✓
- vertices: the right triangle/tetrahedron/... of dimension d with sides of length 1, ✓
- `P_approx` is the vector space of polynomials from $\mathbb{R}^d$ to $\mathbb{R}$ of total degree $\leq k$, ✓
- `S_dof` are the linear forms that take a function and evaluate it at given points for evenly distributed points (called Lagrange nodes), ✓
- a unisolvence proof (Φ_Σ is a bijection). ✓

$\mathcal{A}_d^k$

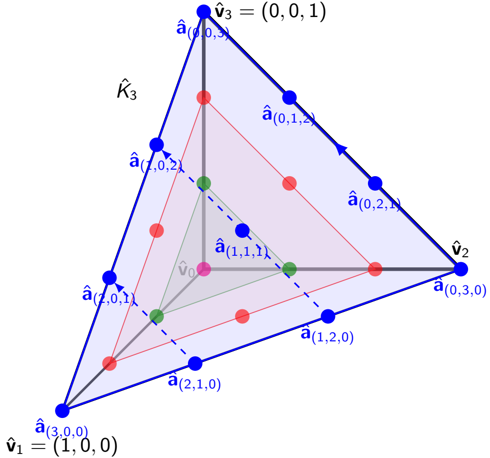
Given d and k, I want the family of families of $\mathbb{N}^d$ with a sum $\leq k$.

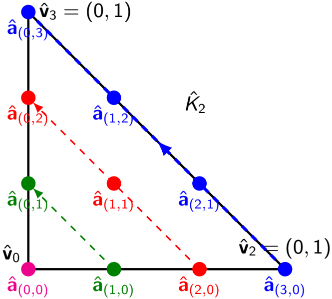
$\hookrightarrow$ useful in geometry for Lagrange nodes,
$\hookrightarrow$ useful for the monomials on $\mathbb{R}^d$ of degree $\leq k$: $(\mathbf{X}^\alpha)_{\alpha \in \mathcal{A}_d^k}$.

We define $\mathcal{A}_d^k$ (of size $\binom{d+k}{d}$) by concatenation and induction.

```
Lemma Adk_sum : forall d k idk, (sum (Adk d k idk) ≤ k).

Lemma Adk_surj : forall d k (b : ℕ^d), (sum b ≤ k) → { idk | b = Adk d k idk }

Lemma Adk_inj : forall d k, injective (Adk d k).

Lemma Adk_sortedF : forall d k, sortedF grsymlex_lt (Adk d k).
    (* a special order near the grevlex order on monomials *)
```

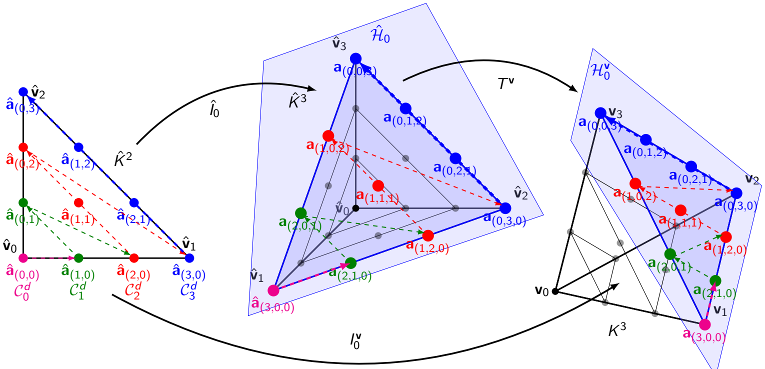
Reference tetrahedron and triangle with Lagrange nodes





Unisolvence

- long and hard proof,
- begins with a double induction on d and k,
- needs factorization of polynomials,
- needs injection onto a face hyperplane:



Outline

Conclusion

- Formal verification is tedious, but brings high guarantee to lemmas, algorithms, and programs.
- Floating-point arithmetic may be formally proved (even hard proofs) – see `coq-flocq`.
- Applied mathematics may be formally proved (even hard proofs) – see `rocq-num-analysis`.

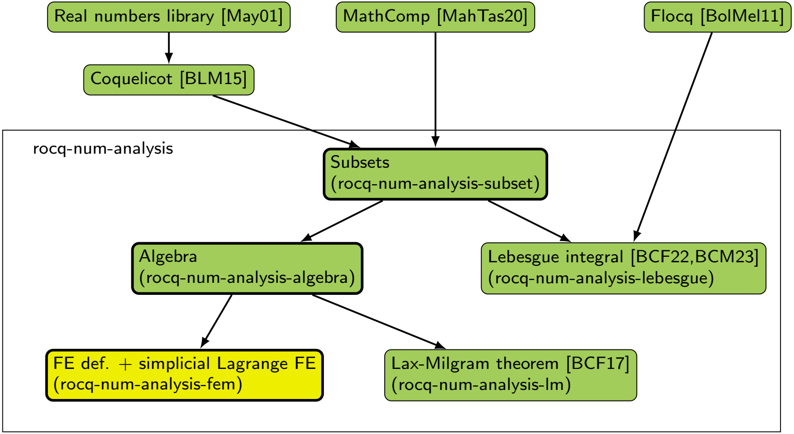
Nevertheless,

- Sometimes, we choose the proof path depending on what is available in the/my Rocq libraries.
- Libraries may be incompatible.
- Technical problems may jump at you unexpectedly (handling of substructures, several paths for canonical structures instantiation).

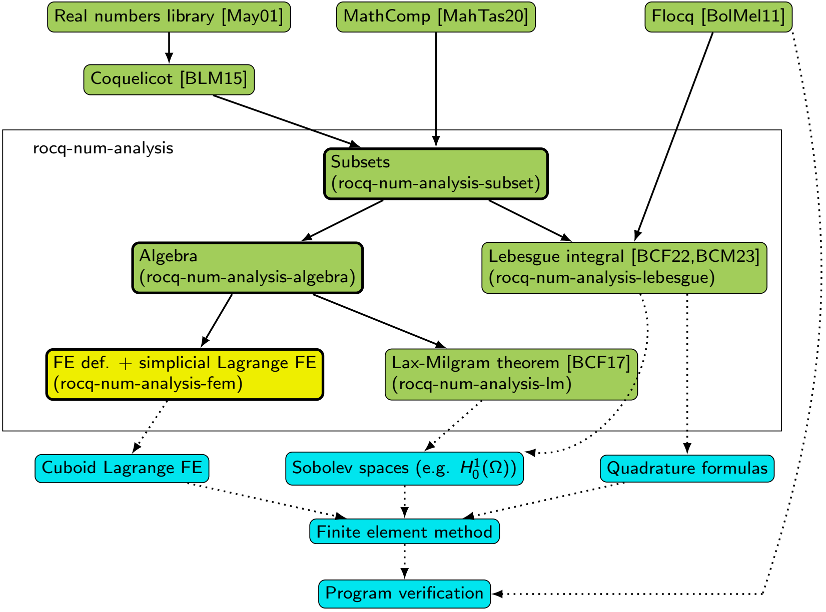
Perspectives 1/2

- More proofs! (FP programs, algorithms, results)
- Convince users!
  - more user-friendly libraries,
  - more comprehensive libraries,
  - convince that the (formal) error bounds are tight $\hookrightarrow$ D. Hamelin's talk
- Long-term work on Finite Elements (cf next slide)

Perspectives 2/2



Perspectives 2/2



Ending with a table of contents?

1. inTroduction
2. formalization of computer aritHmetic
3. proofs of floAting-poiNt programs
4. checKing applied mathS programs
5. conclusion and perspectives

Thanks a lot, Jean-Michel!

1. in**T**roduction
2. formalization of computer arit**H**metic
3. proofs of flo**A**ting-poi**N**t programs
4. chec**K**ing applied math**S** programs
5. conclusion and perspectives