Journée fédérations MIRES/MARGAUx (Poitiers)
EDP : fiabilité/vérification de codes, étude de convergence

Analyse numérique en Rocq
Vers une formalisation de la méthode des éléments finis
Partie I : Introduction aux preuves formelles et à la formalisation

Sylvie Boldo, **François Clément**, Vincent Martin, **Micaela Mayero**, Florian Faissole, Louise Leclerc, Houda Mouhcine

Inria, LIPN - USPN (U. Paris 13)

19/11/2025

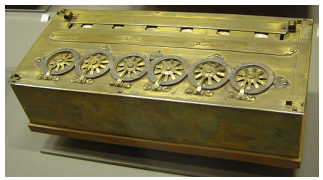
What are we going to talk about?

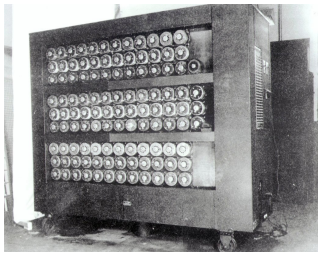
I - Introduction to formal proofs and formalisation
- history, motivation
- what are formal proofs
- Rocq
- real analysis proof assistants
- demos

II - Simplicial Lagrange finite elements in Rocq
- formalisation process
- about measure theory
- simplicial Lagrange finite elements
- perspectives

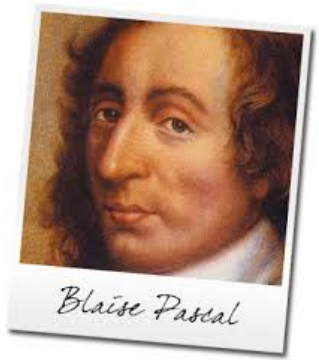
What is a Proof Assistant Tool? (1/6)

History: computers





to compute (1645, 1834, 1936)




Pascal Babbage Turing

What is a Proof Assistant Tool? (2/6)

History: algorithms

1843, Ada Lovelace 

first implementation of an algorithm as a program for the analytic machine of Babbage.

```
1  ALGORITHME "la somme de 10 valeurs"
2  Variables i,somme:Entier
3            tableau t(10):Entier
4  Debut
5  Somme<----0
6  Pour i=1 a 10 Faire
7   Lire t(i)
8   somme<----somme+t(i)
9   Finpour
10  Ecrire "La somme Est :",  somme
11  Fin
12  ********************************************
13  Pour i=1 a 10 Faire
14   Ecrire "donner une valeur"
15   Lire a
16  somme<----somme+a
17  Finpour
```

```
let rec pgcd(i, j) =
  if i = j then i else
  if i > j then pgcd(i-j, j)
  else pgcd(i, j-i);;

let (a,b) = (int_of_string(Sys.argv.(1)),
             int_of_string(Sys.argv.(2)));;

Printf.printf "%i\n" (pgcd(a,b));;

exit 0;;
```

What is a Proof Assistant Tool? (3/6)

History: proofs

1967

De Bruijn, Automath

1984

Coquand, Coq

What is a Proof Assistant Tool? (4/6)

Why Formally Prove? A concrete example 1/2

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...

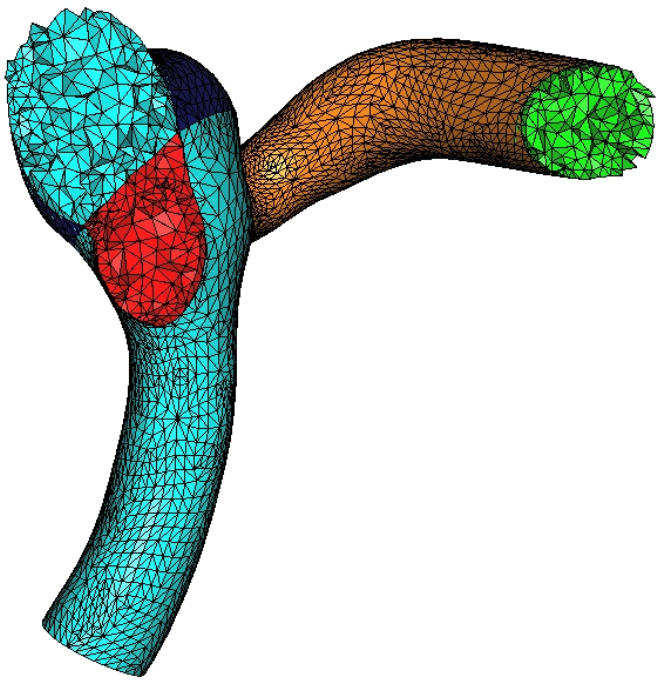
The conclusion of the investigation was that the loss was caused by a failure in a cell wall, resulting in a serious crack and a leakage that the pumps were not able to cope with. The wall failed as a result of a combination of a serious error in the finite element analysis and insufficient anchorage 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 caused a seismic event registering 3.0 on the Richter scale, and left nothing but a pile of debris at 220m of depth. The failure involved a total economic loss of about $700 million.

The 24 cells and 4 shafts referred to above are shown to the left while at the sea surface. The cells are 12m in diameter. The cell wall failure was traced to a tricell, a triangular concrete frame placed where the cells meet, as indicated in the diagram below. To the right of the diagram is pictured a portion of tricell undergoing failure testing.

What is a Proof Assistant Tool? (5/6)

Why Formally Prove? A concrete example 2/2



mesh of a cerebral aneurysm

@V.Martin

Finite Element Method (FEM): widely used, sound mathematical foundations, broad spectrum of PDE, complex 3D geometries + real life applications...

Objective

Prove correctness of the mathematical Finite Element method and of (parts of) a C++ FEM code, in Rocq.

What is a Proof Assistant Tool? (6/6)

Examples of Provers and their Logic

Propositional logic, Peano integers

Automath (de Bruijn, 1968)

Arithmetic (integers)

Mizar (Trybulec, 1970), HOL (Milner, 1970), Isabelle (Paulson, 1990), Coq (Thierry Coquand, 1984), PVS (1992), Agda (Catarina Coquand,1999), ...

Reals numbers

Mizar (Hryniewiecki, 1989), HOL-Light (Harrison, 1996), Coq (Mayero, Niqui, 1998, 2000), PVS (Dutertre, 1996), Isabelle/HOL (Fleuriot, 2000), Lean (Ayers, 2019), ...

Floating numbers

PVS (Miner, 1995), HOL-Light (Harrison, 2000), Coq (Théry, Boldo-Melquiond, 2003, 2011)...

Rocq(1/3)

Logic, tactics

Rocq (ex Coq) is based on CIC (type theory, inductive types, Curry-Howard isomorphism, intuitionistic logic built-in,...).

```
Theorem plus_n_O :
    ∀ n:nat, n + 0 = n.
Proof.
induction n;simpl.
reflexivity.
rewrite IHn. (* Induction Hyp*)
reflexivity.
Qed.
```

```
Theorem plus_n_O :
    ∀ n:nat, n + 0 = n.
Proof.
easy.
Qed.
```

Rocq(2/3)

Proof λ-term

```
Inductive nat : Set :=
  O : nat | S : nat → nat.

nat_ind : ∀ P : nat → Prop,
  P O →
  (∀ n : nat, P n → P (S n)) →
  ∀ n : nat, P n
```

```
f: N → N (* maths *)
   x → 2x
fun x:nat ⇒ 2*x (* Rocq *)
λ x:nat, 2*x (* λ-calculus *)
```

$f(3) = 6$

$(\texttt{fun x:nat} \Rightarrow 2*x)\ 3 \rightsquigarrow_\beta 6$

$(\lambda\, x{:}\texttt{nat},\ 2*x)\ 3 \rightsquigarrow_\beta 6$

then

```
Print plus_n_O.
plus_n_O = fun n : nat ⇒
   nat_ind (fun n0 : nat ⇒ n0 + 0 = n0)
           (eq_refl : 0 + 0 = 0)
           (fun (n0 : nat) (IHn : n0 + 0 = n0) ⇒
              eq_ind_r (fun n1 : nat ⇒ S n1 = S n0)
              eq_refl IHn : S n0 + 0 = S n0) n
   : ∀ n : nat, n + 0 = n
```

Rocq(3/3)

Demo

Formalisation of Real Numbers (1/8)

What is the problem with real numbers ?

To compute with "real numbers" (floating points) in a computer (processor):

$$\begin{cases} A = (1000 + -1000) + 7.501 = 7.501 \\ B = 1000 + (-1000 + 7.501) = 7.5009999999999997635 \end{cases}$$

$$\begin{cases} A = (10e16 + -10e16) + 7.501 = 7.501 \\ B = 10e16 + (-10e16 + 7.501) = 0 \end{cases}$$

$$\begin{cases} A = (10e15 + -10e15) + 7.501 = 7.501 \\ B = 10e15 + (-10e15 + 7.501) = 8 \end{cases}$$

But what about a proof of A = B?

In maths, A = B means that the addition is associative for real numbers. And it's true!

In theorem provers: we want "true" real numbers, to keep the mathematical properties.

Formalisation of Real Numbers (2/8)

What kind of "real numbers" ?

- exact real numbers
- nonstandard real numbers
- constructive real numbers
- classical real numbers
- floating point numbers
- Interval arithmetic

Formalisation of Real Numbers (3/8)

Some ways to define real numbers

Mathematics:

Remark: real numbers are inumerable $\hookrightarrow$ no inductive type

- axiomatisation/construction
- Cauchy, Cantor, Dedekind cut
- 1st ordre/2nd order

Logic:

- classical logic / intuitionnistic logic
- analysis "usual in school"/ Bishop analysis

Formalisation of Real Numbers (4/8)

Some examples in some provers: Mizar

The logic is based on the axioms of the Tarski-Grothendieck set theory (an extension of the classical Zermelo-Fraenkel set theory).
While originally an axiomatization, the formalization of real numbers in Mizar was later changed to a construction based on Dedekind cuts (Trybulec 1998):

Formalisation of Real Numbers (4/8)

Some examples in some provers: Mizar

The logic is based on the axioms of the Tarski-Grothendieck set theory (an extension of the classical Zermelo-Fraenkel set theory).
While originally an axiomatization, the formalization of real numbers in Mizar was later changed to a construction based on Dedekind cuts (Trybulec 1998):

```
func DEDEKIND_CUTS -> Subset-Family of RAT+ equals
{ A where A is Subset of RAT+ :
for r being Element of RAT+ str in A holds
( ( for s being Element of RAT+ st s <=' r holds s in A ) &
ex s being Element of RAT+ st ( s in A & r < s ) )
} \ { RAT+};
```

A is a Dedekind cut if it is a strict subset of $\mathbb{Q}^+ \cup \{0\}$ such that $\forall r \in A$,
$(\forall s \in \mathbb{Q}^+ \cup \{0\}, s \leq r \Rightarrow s \in A) \wedge (\exists s \in \mathbb{Q}^+ \cup \{0\}, r < s \wedge s \in A)$

```
func A + B -> Element of DEDEKIND_CUTS equals
{ ( r + s ) where r , s is Element of RAT+ : ( r in A & s in B ) };
```

Formalisation of Real Numbers (6/8)

Some examples in some provers: HOL-Light

Based on classical higher-order logic with axioms of infinity, extensionality, and choice in the form of Hilbert's ϵ operator. It follows the LCF approach.
Rather than using fully-featured sequences of rational numbers, real numbers are based on nearly-additive sequences of natural numbers.
The predicate `is_nadd` characterizing such a sequence `x` is given below, with `dist` the function returning the distance between two natural numbers.

Formalisation of Real Numbers (6/8)

Some examples in some provers: HOL-Light

Based on classical higher-order logic with axioms of infinity, extensionality, and choice in the form of Hilbert's ϵ operator. It follows the LCF approach.
Rather than using fully-featured sequences of rational numbers, real numbers are based on nearly-additive sequences of natural numbers.
The predicate `is_nadd` characterizing such a sequence `x` is given below, with `dist` the function returning the distance between two natural numbers.

```
let is_nadd = new definition
`is_nadd x <=> (?B. !m n. dist (m*x(n), n*x(m))<=B*(m+n))`;;

let nadd_le = new definition
`x <<= y <=> ?B. !n. x(n) <= y(n)+B`;;
```

- existence of the least upper bound of any nonempty bounded set of nearly-additive sequences;
- this theorem is later extended to give the completeness of real numbers.
- addition of two nearly-additive sequences is performed pointwise
- the multiplication is the composition of sequences
- whole real line is built as a successive quotient types.

Formalisation of Real Numbers (7/8)

Some examples in some provers: PVS

The logic is based on classical higher-order logic.
Thanks to its pervasive support for subtyping, PVS takes a top-down approach. PVS starts from a number type that is a superset of all numerals. This set encompasses integers (from Lisp), rationals, real numbers. Below is an excerpt of the pvs reals.

Formalisation of Real Numbers (7/8)

Some examples in some provers: PVS

The logic is based on classical higher-order logic.
Thanks to its pervasive support for subtyping, PVS takes a top-down approach. PVS starts from a number type that is a superset of all numerals. This set encompasses integers (from Lisp), rationals, real numbers. Below is an excerpt of the pvs reals.

```
number : NONEMPTYTYPE
number_field : NONEMPTYTYPE FROM number
n0x : VAR nonzero_number
inverse_mult : AXIOM n0x * (1/ n0x ) = 1

real : NONEMPTYTYPE FROM number_field
nonzero_real : NONEMPTYTYPE = { r : real | r /= 0} CONTAINING 1
nzreal_is_nznum : JUDGEMENT nonzero_real SUBTYPE OF nonzero_number
x , y : VAR real
posreal_add_closed : POSTULATE x > 0 AND y > 0 IMPLIES x + y > 0
trichotomy : POSTULATE x > 0 OR x = 0 OR 0 > x
```

(`POSTULATE` means that PVS decision procedure are able to prove these properties: it is the combination of the explicit axioms from theories and the implicite ones from decision procedures that defines what real numbers are in PVS.)

Formalisation of Real Numbers (8/8)

Some examples in some provers: LEAN

The logic is based on Calculus of Inductives Constructions extended with quotient types.
The type of real numbers constructed as equivalence classes of Cauchy sequences of rational numbers.

Formalisation of Real Numbers (8/8)

Some examples in some provers: LEAN

The logic is based on Calculus of Inductives Constructions extended with quotient types. The type of real numbers constructed as equivalence classes of Cauchy sequences of rational numbers.

```
structure real := of_cauchy ::
(cauchy : cau_seq.completion.Cauchy (abs : Q → Q))

def Cauchy := @quotient (cau_seq _ abv) cau_seq.equiv

instance equiv : setoid (cau_seq B abv) :=...
```

Mathlib: a unified library of mathematics formalized. It is distinguished by dependently typed foundations, is by design fully classical mathematics, with extensive hierarchy of structures.

Real Numbers in Rocq

```
**************
*  In Rocq!  *
**************
```

Real Numbers in Rocq(1/4)

Real numbers libraries

Reals

(Mayero, 2000) are purely axiomatized. The operations are total functions. The comparison between reals is decidable (classical aspect).

C-CoRN

(Niqui, 2000) is a constructive real numbers library.

Coquelicot

(Boldo, Lelay, Melquiond, 2013) a conservative user-friendly extension of stdlib Reals, easier way of writing formulas and theorem statements, by relying on total functions in place of dependent types for limits, derivatives, integrals, power series,...

Reals

(Semeria, 2020) has been conservatively updated introducing a commun constructive and classical part.

Real Numbers in Rocq(2/4)

Reals: standard library of real numbers

17 axioms:

```
Parameter ℝ: Set.
Parameter Rplus: ℝ→ ℝ→ ℝ.
Parameter Rlt: ℝ→ ℝ→ Prop.
Parameter Rinv: ℝ→ ℝ.
Axiom Rplus_com: ∀ r1 r2:ℝ, r1 + r2 = r2 + r1.
Axiom Rplus_lt_compat_l: ∀ r r1 r2 :ℝ , r1 < r2 → r + r1 < r + r2.
Axiom Rinv_l : ∀ r :ℝ , r ≠ 0 → / r * r = 1.
Axiom total_order_T: ∀ r1 r2 :ℝ , {r1 < r2} + {r1 = r2} + {r1 > r2}.
...
Definition is_upper_bound (E:R → Prop) (m:R) := ∀ x:ℝ, E x → x <= m.
Definition bound (E:R → Prop) := ∃ m : R, is_upper_bound E m.
Definition is_lub (E:R → Prop) (m:R) :=
  is_upper_bound E m ∧ (∀  b:R, is_upper_bound E b → m <= b).
Axiom completeness :
    ∀ E:R → Prop,
      bound E → (∃  x : R, E x) → { m:R | is_lub E m }.
```

Real Numbers in Rocq(2/4)

Reals: standard library of real numbers

17 axioms:

```
Parameter ℝ: Set.
Parameter Rplus: ℝ→ ℝ→ ℝ.
Parameter Rlt: ℝ→ ℝ→ Prop.
Parameter Rinv: ℝ→ ℝ.
Axiom Rplus_com: ∀ r1 r2:ℝ, r1 + r2 = r2 + r1.
Axiom Rplus_lt_compat_l: ∀ r r1 r2 :ℝ , r1 < r2 → r + r1 < r + r2.
Axiom Rinv_l : ∀ r :ℝ , r ≠ 0 → / r * r = 1.
Axiom total_order_T: ∀ r1 r2 :ℝ , {r1 < r2} + {r1 = r2} + {r1 > r2}.
...
Definition is_upper_bound (E:R → Prop) (m:R) := ∀ x:ℝ, E x → x <= m.
Definition bound (E:R → Prop) := ∃ m : R, is_upper_bound E m.
Definition is_lub (E:R → Prop) (m:R) :=
  is_upper_bound E m ∧ (∀  b:R, is_upper_bound E b → m <= b).
Axiom completeness :
    ∀ E:R → Prop,
      bound E → (∃  x : R, E x) → { m:R | is_lub E m }.
```

Real Numbers in Rocq(2/4)

Reals: standard library of real numbers

17 axioms:

```
Parameter ℝ: Set.
Parameter Rplus: ℝ→ ℝ→ ℝ.
Parameter Rlt: ℝ→ ℝ→ Prop.
Parameter Rinv: ℝ→ ℝ.
Axiom Rplus_com: ∀ r1 r2:ℝ, r1 + r2 = r2 + r1.
Axiom Rplus_lt_compat_l: ∀ r r1 r2 :ℝ , r1 < r2 → r + r1 < r + r2.
Axiom Rinv_l : ∀ r :ℝ , r ≠ 0 → / r * r = 1.
Axiom total_order_T: ∀ r1 r2 :ℝ , {r1 < r2} + {r1 = r2} + {r1 > r2}.
...
Definition is_upper_bound (E:R → Prop) (m:R) := ∀ x:ℝ, E x → x <= m.
Definition bound (E:R → Prop) := ∃ m : R, is_upper_bound E m.
Definition is_lub (E:R → Prop) (m:R) :=
  is_upper_bound E m ∧ (∀  b:R, is_upper_bound E b → m <= b).
Axiom completeness :
    ∀ E:R → Prop,
      bound E → (∃  x : R, E x) → { m:R | is_lub E m }.
```

Real Numbers in Rocq(2/4)

Reals: standard library of real numbers

17 axioms:

```
Parameter ℝ: Set.
Parameter Rplus: ℝ→ ℝ→ ℝ.
Parameter Rlt: ℝ→ ℝ→ Prop.
Parameter Rinv: ℝ→ ℝ.
Axiom Rplus_com: ∀ r1 r2:ℝ, r1 + r2 = r2 + r1.
Axiom Rplus_lt_compat_l: ∀ r r1 r2 :ℝ , r1 < r2 → r + r1 < r + r2.
Axiom Rinv_l : ∀ r :ℝ , r ≠ 0 → / r * r = 1.
Axiom total_order_T: ∀ r1 r2 :ℝ , {r1 < r2} + {r1 = r2} + {r1 > r2}.
...
Definition is_upper_bound (E:R → Prop) (m:R) := ∀ x:ℝ, E x → x <= m.
Definition bound (E:R → Prop) := ∃ m : R, is_upper_bound E m.
Definition is_lub (E:R → Prop) (m:R) :=
  is_upper_bound E m ∧ (∀  b:R, is_upper_bound E b → m <= b).
Axiom completeness :
    ∀ E:R → Prop,
      bound E → (∃  x : R, E x) → { m:R | is_lub E m }.
```

Real Numbers in Rocq(3/4)

Example: derivative

```
Definition D_x (D:R → Prop) (y x:R) : Prop := D x ∧ y ≠ x.          (* domaine *)

Definition continue_in (f:R → R) (D:R → Prop) (x0:R) : Prop :=
  limit1_in f (D_x D x0) (f x0) x0.

Definition D_in (f d:R → R) (D:R → Prop) (x0:R) : Prop :=          (* derivative *)
  limit1_in (fun x:R ⇒ (f x − f x0) / (x − x0)) (D_x D x0) (d x0) x0.

Lemma cont_deriv : ∀ (f d:R → R) (D:R → Prop) (x0:R),
   D_in f d D x0 → continue_in f D x0.

Lemma Dconst : ∀ (D:R → Prop) (y x0:R),
   D_in (fun x:R ⇒ y) (fun x:R ⇒ 0) D x0.

Lemma Dx : ∀ (D:R → Prop) (x0:R),
   D_in (fun x:R ⇒ x) (fun x:R ⇒ 1) D x0.
```

Real Numbers in Rocq(3/4)

Example: derivative

```
Definition D_x (D:R → Prop) (y x:R) : Prop := D x ∧ y ≠ x.          (* domaine *)

Definition continue_in (f:R → R) (D:R → Prop) (x0:R) : Prop :=
  limit1_in f (D_x D x0) (f x0) x0.

Definition D_in (f d:R → R) (D:R → Prop) (x0:R) : Prop :=          (* derivative *)
  limit1_in (fun x:R ⇒ (f x − f x0) / (x − x0)) (D_x D x0) (d x0) x0.

Lemma cont_deriv : ∀ (f d:R → R) (D:R → Prop) (x0:R),
   D_in f d D x0 → continue_in f D x0.

Lemma Dconst : ∀ (D:R → Prop) (y x0:R),
   D_in (fun x:R ⇒ y) (fun x:R ⇒ 0) D x0.

Lemma Dx : ∀ (D:R → Prop) (x0:R),
   D_in (fun x:R ⇒ x) (fun x:R ⇒ 1) D x0.
```

Real Numbers in Rocq(4/4)

Demo from teaching

Double Licence Maths-Info, L1, Institut Galilée, USPN

Real Analysis in Rocq(1/4)

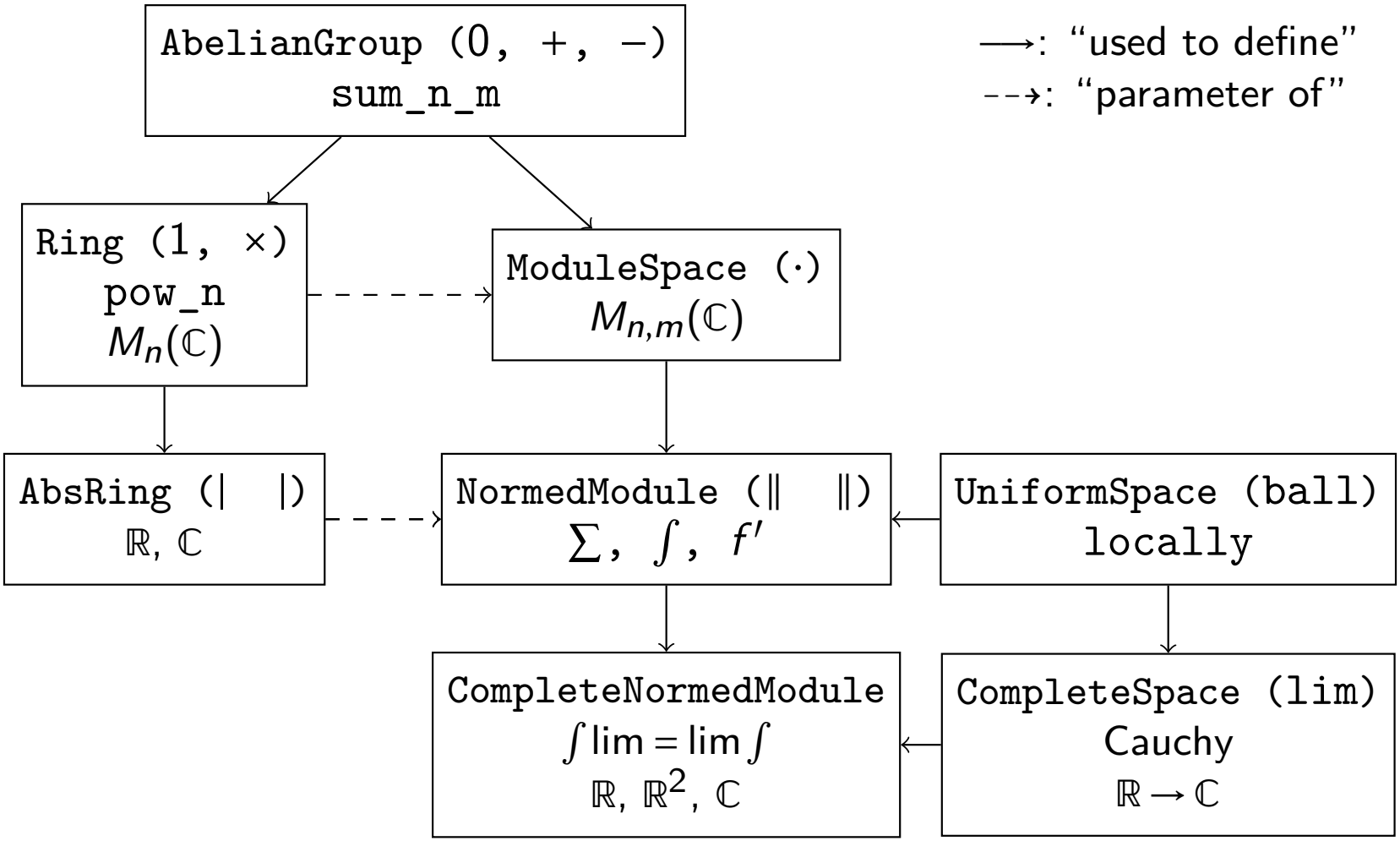
Coquelicot: generalities

Main contributions compared to Reals:

- Compatible with stdlib Reals
- Full total functions (limit, derivative, integral)
- Tactic dedicated to derivative proofs
- Algebraic hierarchy
- Abstraction (filters...)
- $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$

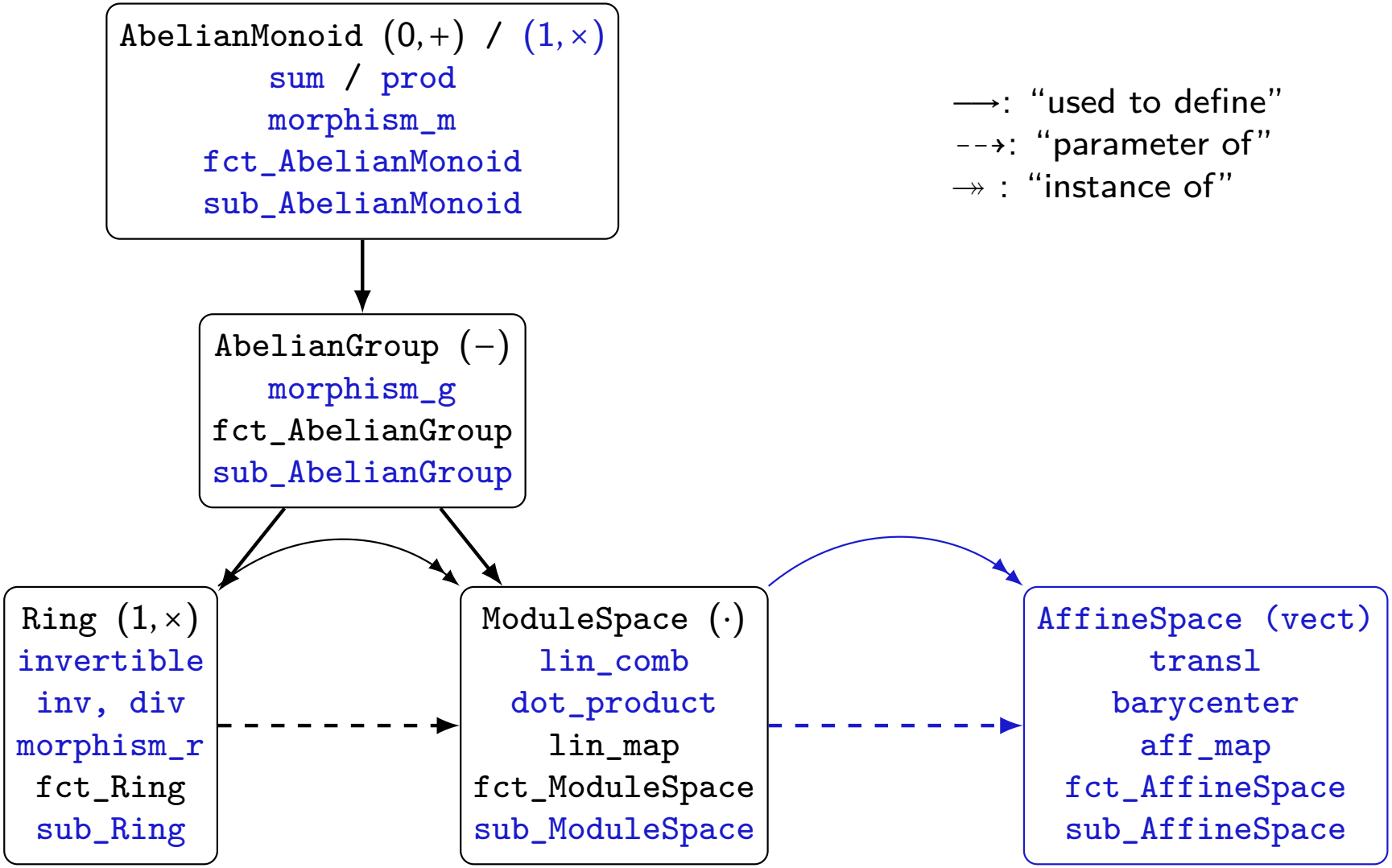
Real Analysis in Rocq(2/4)

Coquelicot: hierarchy



Real Analysis in Rocq(3/4)

Rocq-num-analysis: hierarchy of algebraic structures (Coquelicot in black)



Real Analysis in Rocq(4/4)

Demo: GIT of rocq-num-analysis

https://lipn.univ-paris13.fr/rocq-num-analysis/
https://lipn.univ-paris13.fr/rocqdoc-num-analysis/2.0/

Short conclusion, toward part II

- In 25 years: from real numbers to numerical analysis
- Increasingly useful for analysis
- We have the tools for part II, adding some notions:
  - forall: `∀`, exists: `∃`, function: `fun ... ⇒`,
    logic implication: `→`, negation: `¬`
  - `Definition`, `Lemma`, `Notation`, `Inductive`
  - comment: `(*...*)`
  - sorts: `Type`, `Prop`
  - implicit declaration: `Context{...}`
    explicit declaration: `Context(...)`

To be continued

Enjoy!