Asymptotic Reasoning in Coq

Reynald Affeldt [1] Cyril Cohen [2] Damien Rouhling [2]

[1] National Institute of Advanced Industrial Science and Technology, Japan

[2] Université Côte d'Azur, Inria, France

September 3, 2018

Credits: several slides are inspired/adapted/stolen from different talks by Cyril Cohen

Motivation

- Asymptotic reasoning involves a lot of ε/δ-reasoning.

$$\left\{\begin{array}{l} \forall\varepsilon > 0,\ \exists\delta_f > 0,\ \forall x,\ |x-a| < \delta_f \Rightarrow |f(x) - l_f| < \varepsilon \\ \forall\varepsilon > 0,\ \exists\delta_g > 0,\ \forall x,\ |x-a| < \delta_g \Rightarrow |g(x) - l_g| < \varepsilon \end{array}\right.,$$

$$\Rightarrow \forall\varepsilon > 0,\ \exists\delta > 0,\ \forall x,\ |x-a| < \delta \Rightarrow |f(x) + g(x) - (l_f + l_g)| < \varepsilon.$$

Motivation

- Asymptotic reasoning involves a lot of ε/δ-reasoning.
- In the mathematical (informal) practice, asymptotic hand-waving often gives a more convenient framework to write proofs.

$$\underset{x\to 0}{o}(x^n) + \underset{x\to 0}{o}(x^n) = \underset{x\to 0}{o}(x^n)$$

$$\underset{x\to 0}{o}(x^n) + \underset{x\to 0}{\mathcal{O}}(x^n) = \underset{x\to 0}{\mathcal{O}}(x^n)$$

$$\ldots$$

Motivation

- Asymptotic reasoning involves a lot of ε/δ-reasoning.
- In the mathematical (informal) practice, asymptotic hand-waving often gives a more convenient framework to write proofs.
- We want the best of both informal and formal reasoning.
    1. Simplicity and ease of use.
    2. Strong guarantees on the correctness of the proof.

Our goals

- Formal proofs for robotics.
  - Kinematic chains.
  - Various aspects of robot motion and control.
- Undergraduate classic textbook analysis.
- Catch up with ISABELLE/HOL and LEAN.

Context

- Formalization in Coq + SSReflect.
- We take inspiration from well-established methodologies:
  - We follow Mathematical Components's design principle: small-scale tactics, fewer definitions, more combiners and lemmas that hide the most technical parts.
  - We use filters for local reasoning, an abstraction that proved to be efficient in Coquelicot and Isabelle/HOL's analysis library.

Context

- Formalization in Coq + SSReflect.
- We take inspiration from well-established methodologies:
  - We follow Mathematical Components's design principle: small-scale tactics, fewer definitions, more combiners and lemmas that hide the most technical parts.
  - We use filters for local reasoning, an abstraction that proved to be efficient in Coquelicot and Isabelle/HOL's analysis library.

One crucial difference with Coquelicot: we use additional axioms.

Contributions

- Techniques to do asymptotic hand-waving
  1. in a rigorous (formal) way,
  2. with robust small-scale proofs,
  3. concisely,

  in the form of

  1. a set of tactics and notations that make local reasoning smoother,
  2. a small theory of little-o and big-$\mathcal{O}$ based on Bachmann-Landau notations.
- Yet another analysis library, Mathematical Components Analysis[1], compatible with Mathematical Components and that integrates these techniques.

[1] `https://github.com/math-comp/analysis`, joint work with Reynald Affeldt, Cyril Cohen, Assia Mahboubi and Pierre-Yves Strub.

Contributions

- Techniques to do asymptotic hand-waving
  1. in a rigorous (formal) way,
  2. with robust small-scale proofs,
  3. concisely,

  in the form of
  1. a set of tactics and notations that make local reasoning smoother,
  2. a small theory of little-o and big-$\mathcal{O}$ based on Bachmann-Landau notations.
- Yet another analysis library, Mathematical Components Analysis[1], compatible with Mathematical Components and that integrates these techniques.

[1] https://github.com/math-comp/analysis, joint work with Reynald Affeldt, Cyril Cohen, Assia Mahboubi and Pierre-Yves Strub.

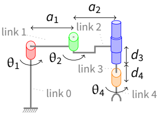
Use cases

What is a Robot Manipulator?

- E.g., SCARA (Selective Compliance Assembly Robot Arm)

Mitsubishi RH-S series

Schematic version



- Robot manipulator $\stackrel{def}{=}$ *Links* connected by *joints*
  - Revolute joint $\leftrightarrow$ rotation
  - Prismatic joint $\leftrightarrow$ translation

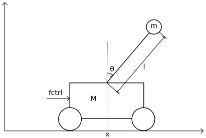
`https://github.com/affeldt-aist/coq-robot`

Use cases

The inverted pendulum

https://github.com/drouhling/LaSalle/tree/mathcomp-analysis

- The inverted pendulum is a standard example for testing control techniques.



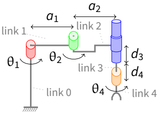
- Goal: stabilize the pendulum on its unstable equilibrium thanks to the control function fctrl.

What is a Robot Manipulator?

- E.g., SCARA (Selective Compliance Assembly Robot Arm)

Mitsubishi RH-S series

Schematic version



- Robot manipulator $\stackrel{def}{=}$ *Links* connected by *joints*
  - Revolute joint ↔ rotation
  - Prismatic joint ↔ translation

https://github.com/affeldt-aist/coq-robot

Motivating example

To prove

$$\lim_a f = l_f \wedge \lim_a g = l_g \Rightarrow \lim_a (f+g) = l_f + l_g$$

Typical ε/δ-reasoning:

$$\left\{ \begin{array}{l} \forall \varepsilon > 0,\ \exists \delta_f > 0,\ \forall x,\ |x-a| < \delta_f \Rightarrow |f(x) - l_f| < \varepsilon \\ \forall \varepsilon > 0,\ \exists \delta_g > 0,\ \forall x,\ |x-a| < \delta_g \Rightarrow |g(x) - l_g| < \varepsilon \end{array} \right. ,$$

$$\left\{ \ \forall \varepsilon > 0,\ \exists \delta > 0,\ \forall x,\ |x-a| < \delta \Rightarrow |f(x) + g(x) - (l_f + l_g)| < \varepsilon. \right.$$

Motivating example

To prove

$$\lim_a f = l_f \wedge \lim_a g = l_g \Rightarrow \lim_a (f + g) = l_f + l_g$$

Typical ε/δ-reasoning:

$$\left\{ \begin{array}{l} \forall \varepsilon > 0,\ \exists \delta_f > 0,\ \forall x,\ |x - a| < \delta_f \Rightarrow |f(x) - l_f| < \varepsilon \\ \forall \varepsilon > 0,\ \exists \delta_g > 0,\ \forall x,\ |x - a| < \delta_g \Rightarrow |g(x) - l_g| < \varepsilon \\ \varepsilon > 0 \end{array} \right. ,$$

$$\{\ \exists \delta > 0,\ \forall x,\ |x - a| < \delta \Rightarrow |f(x) + g(x) - (l_f + l_g)| < \varepsilon.$$

Motivating example

To prove

$$\lim_a f = l_f \wedge \lim_a g = l_g \Rightarrow \lim_a (f+g) = l_f + l_g$$

Typical ε/δ-reasoning:

$$\frac{\left\{\begin{array}{l}
\forall \varepsilon > 0,\ \exists \delta_f > 0,\ \forall x,\ |x-a| < \delta_f \Rightarrow |f(x) - l_f| < \varepsilon \\
\forall \varepsilon > 0,\ \exists \delta_g > 0,\ \forall x,\ |x-a| < \delta_g \Rightarrow |g(x) - l_g| < \varepsilon \\
\varepsilon > 0 \\
\delta_f > 0 \\
\forall x,\ |x-a| < \delta_f \Rightarrow |f(x) - l_f| < \frac{\varepsilon}{2} \\
\delta_g > 0 \\
\forall x,\ |x-a| < \delta_g \Rightarrow |g(x) - l_g| < \frac{\varepsilon}{2} \qquad \text{magical guess}
\end{array}\right. ,}{\left\{\ \exists \delta > 0,\ \forall x,\ |x-a| < \delta \Rightarrow\ |f(x) + g(x) - (l_f + l_g)| < \varepsilon. \right.}$$

Motivating example

To prove

$$\lim_a f = l_f \wedge \lim_a g = l_g \Rightarrow \lim_a (f+g) = l_f + l_g$$

Typical ε/δ-reasoning:

$$\left\{\begin{array}{l}
\forall \varepsilon > 0,\ \exists \delta_f > 0,\ \forall x,\ |x-a| < \delta_f \Rightarrow |f(x) - l_f| < \varepsilon \\
\forall \varepsilon > 0,\ \exists \delta_g > 0,\ \forall x,\ |x-a| < \delta_g \Rightarrow |g(x) - l_g| < \varepsilon \\
\varepsilon > 0 \\
\delta_f > 0 \\
\forall x,\ |x-a| < \delta_f \Rightarrow |f(x) - l_f| < \frac{\varepsilon}{2} \\
\delta_g > 0 \\
\forall x,\ |x-a| < \delta_g \Rightarrow |g(x) - l_g| < \frac{\varepsilon}{2}
\end{array}\right. ,$$

$$\left\{\begin{array}{l}
\forall x,\ |x-a| < \min(\delta_f, \delta_g) \Rightarrow |f(x) + g(x) - (l_f + l_g)| < \varepsilon. \\
\qquad \text{magical guess}
\end{array}\right.$$

Why ε/δ definitions are not best for formal proofs

A few aspects of typical ε/δ-reasoning:

- The (human) prover has to provide existential witnesses.
- Witnesses are (usually) explicit.
- Witnesses are (usually) given way before they are used.

Why ε/δ definitions are not best for formal proofs

A few aspects of typical ε/δ-reasoning:

- The (human) prover has to provide existential witnesses.
- Witnesses are (usually) explicit.
- Witnesses are (usually) given way before they are used.

$\Rightarrow$ Proof scripts are hard to read and hard to maintain.

Why ε/δ definitions are not best for formal proofs

A few aspects of typical ε/δ-reasoning:

- The (human) prover has to provide existential witnesses.
- Witnesses are (usually) explicit.
- Witnesses are (usually) given way before they are used.

$\Rightarrow$ Proof scripts are hard to read and hard to maintain.

$\Rightarrow$ Use an abstraction like filters.

A quick introduction to filters

A set of sets F is a filter if

$$F \neq \emptyset \quad \forall A, B \in F, A \cap B \in F \quad \forall A \in F, \forall B \supseteq A, B \in F.$$

Examples and notations:

- Neighbourhood filter of a point:

$$\mathrm{locally}(p) = \{A \mid \exists \varepsilon > 0, \mathrm{ball}_\varepsilon(p) \subseteq A\}.$$

- Neighbourhood filter of $+\infty$:

$$\mathrm{locally}(+\infty) = \{A \mid \exists M,]M; +\infty[\subseteq A\}.$$

- Image of a filter F by a function f:

$$f @ F = \left\{A \mid f^{-1}(A) \in F\right\}.$$

- Reverse filter inclusion: $F \to G = G \subseteq F$.

Filter inference and notations

Thanks to a set of canonical structures, we can equip types with a canonical filter function

```
locally : forall (U : Type) (T : filteredType U),
  T -> set (set U),
```

and use notations where the occurrences of this function are inferred:

```
f @ x --> y, lim (f @ x), cvg (f @ +oo), u --> -oo.
```

Motivating example (cont.)

To prove

$$\lim_a f = l_f \wedge \lim_a g = l_g \Rightarrow \lim_a (f+g) = l_f + l_g$$

Typical ε/δ-reasoning:

$$\frac{\left\{ \begin{array}{l} \forall \varepsilon > 0,\ \exists \delta_f > 0,\ \forall x,\ |x-a| < \delta_f \Rightarrow |f(x) - l_f| < \varepsilon \\ \forall \varepsilon > 0,\ \exists \delta_g > 0,\ \forall x,\ |x-a| < \delta_g \Rightarrow |g(x) - l_g| < \varepsilon \end{array} \right. ,}{\left\{ \ \forall \varepsilon > 0,\ \exists \delta > 0,\ \forall x,\ |x-a| < \delta \Rightarrow |f(x) + g(x) - (l_f + l_g)| < \varepsilon \right. .}$$

Motivating example (cont.)

To prove

$$f@a \to l_f \;\Rightarrow\; g@a \to l_g \;\Rightarrow\; (f+g)@a \to (l_f + l_g)$$

Filter reasoning:

$$\frac{\left\{\begin{array}{l} \mathrm{locally}(l_f) \subseteq f@a \\ \mathrm{locally}(l_g) \subseteq g@a \end{array}\right.,}{\left\{\ \mathrm{locally}(l_f + l_g) \subseteq (f+g)@a\right. .}$$

Motivating example (cont.)

To prove

$$f@a \to l_f \;\Rightarrow\; g@a \to l_g \;\Rightarrow\; (f+g)@a \to (l_f + l_g)$$

Filter reasoning:

$$\dfrac{\left\{\begin{array}{l} \mathrm{locally}(l_f) \subseteq f@a \\ \mathrm{locally}(l_g) \subseteq g@a \\ A \in \mathrm{locally}(l_f + l_g) \end{array}\right. ,}{\left\{\; A \in (f+g)@a \right. .}$$

Motivating example (cont.)

To prove

$$f@a \to l_f \;\Rightarrow\; g@a \to l_g \;\Rightarrow\; (f+g)@a \to (l_f + l_g)$$

Filter reasoning:

$$\left\{\begin{array}{ll} \mathrm{locally}(l_f) \subseteq f@a & \\ \mathrm{locally}(l_g) \subseteq g@a & \\ \varepsilon > 0 & \\ \mathrm{ball}_\varepsilon(l_f + l_g) \subseteq A & \quad \text{unfolding} \end{array}\right. ,$$

$$\left\{\begin{array}{l} A \in (f+g)@a \\ (\text{i.e. } (f+g)^{-1}(A) \in \mathrm{locally}(a)) \end{array}\right. .$$

Motivating example (cont.)

To prove

$$f@a \to l_f \;\Rightarrow\; g@a \to l_g \;\Rightarrow\; (f+g)@a \to (l_f + l_g)$$

Filter reasoning:

$$\frac{\begin{cases} \mathrm{locally}(l_f) \subseteq f@a \\ \mathrm{locally}(l_g) \subseteq g@a \\ \varepsilon > 0 \\ \mathrm{ball}_\varepsilon(l_f + l_g) \subseteq A \\ B := (f+g)(f^{-1}(\mathrm{ball}_{\frac{\varepsilon}{2}}(l_f)) \cap g^{-1}(\mathrm{ball}_{\frac{\varepsilon}{2}}(l_g))) \qquad \text{magical guess} \end{cases},}{\text{closure by extension} \qquad \begin{cases} B \in (f+g)@a \\ B \subseteq A \end{cases}.}$$

Motivating example (cont.)

To prove

$$f@a \to l_f \ \Rightarrow\ g@a \to l_g \ \Rightarrow\ (f+g)@a \to (l_f + l_g)$$

Filter reasoning:

$$\left\{\begin{array}{l} \text{locally}(l_f) \subseteq f@a \\ \text{locally}(l_g) \subseteq g@a \\ \varepsilon > 0 \\ \text{ball}_\varepsilon(l_f + l_g) \subseteq A \\ B := (f+g)(f^{-1}(\text{ball}_{\frac{\varepsilon}{2}}(l_f)) \cap g^{-1}(\text{ball}_{\frac{\varepsilon}{2}}(l_g))) \end{array}\right. ,$$

$$\forall C,\ f(f^{-1}(C)) \subseteq C \subseteq f^{-1}(f(C))$$

$$\left\{\begin{array}{l} f^{-1}(\text{ball}_{\frac{\varepsilon}{2}}(l_f)) \cap g^{-1}(\text{ball}_{\frac{\varepsilon}{2}}(l_g)) \in \text{locally}(a) \\ \text{ball}_{\frac{\varepsilon}{2}}(l_f) + \text{ball}_{\frac{\varepsilon}{2}}(l_g) \subseteq \text{ball}_\varepsilon(l_f + l_g) \end{array}\right. .$$

Motivating example (cont.)

To prove

$$f@a \to l_f \;\Rightarrow\; g@a \to l_g \;\Rightarrow\; (f+g)@a \to (l_f + l_g)$$

Filter reasoning:

$$\frac{\left\{\begin{array}{l} \mathrm{locally}(l_f) \subseteq f@a \\ \mathrm{locally}(l_g) \subseteq g@a \\ \varepsilon > 0 \\ \mathrm{ball}_\varepsilon(l_f + l_g) \subseteq A \\ B := (f+g)(f^{-1}(\mathrm{ball}_{\frac{\varepsilon}{2}}(l_f)) \cap g^{-1}(\mathrm{ball}_{\frac{\varepsilon}{2}}(l_g))) \end{array}\right. ,}{\text{closure by intersection} \quad \left\{\begin{array}{l} f^{-1}(\mathrm{ball}_{\frac{\varepsilon}{2}}(l_f)) \in \mathrm{locally}(a) \\ g^{-1}(\mathrm{ball}_{\frac{\varepsilon}{2}}(l_g)) \in \mathrm{locally}(a) \end{array}\right. .}$$

The pros and cons of filter reasoning

The situation is improved:

- The explicit existential witnesses are removed.
- Parts of the arithmetic is hidden thanks to the abstraction.

But:

- There is still a magical guess: we have to know beforehand how we want to split the epsilons.
- We manipulate sets while (I think) it is more intuitive to reason about points.

The pros and cons of filter reasoning

The situation is improved:

- The explicit existential witnesses are removed.
- Parts of the arithmetic is hidden thanks to the abstraction.

But:

- There is still a magical guess: we have to know beforehand how we want to split the epsilons.
- We manipulate sets while (I think) it is more intuitive to reason about points.

$\Rightarrow$ Reintroduce points without breaking the abstraction and use existential variables.

Key ingredients

- A lemma to reintroduce points and use existential variables.

```
Lemma filter_near_of F (P : in_filter F) Q :
  Filter F -> (forall x, prop_of P x -> Q x) -> F Q.
```

- A notation $\forall x \, \mathrm{near} \, F, \; Q \, x$, standing for $Q \in F$, to invite the user to reason about points.
- The fact that filters are closed by intersection, to accumulate properties.

Motivating example (end)

To prove

$$f @ a \to l_f \;\Rightarrow\; g @ a \to l_g \;\Rightarrow\; (f+g) @ a \to (l_f + l_g)$$

Filter reasoning:

$$\frac{\left\{ \begin{array}{l} f @ a \to l_f \\ g @ a \to l_g \end{array} \right. ,}{\left\{ \; (f+g) @ a \to (l_f + l_g) \right. \quad .}$$

Motivating example (end)

To prove

$$f@a \to l_f \;\Rightarrow\; g@a \to l_g \;\Rightarrow\; (f+g)@a \to (l_f + l_g)$$

Improved filter reasoning:

$$\frac{\left\{\begin{array}{l} \forall\varepsilon > 0,\ \forall x\,\mathrm{near}\,a,\ |f(x) - l_f| < \varepsilon \\ \forall\varepsilon > 0,\ \forall x\,\mathrm{near}\,a,\ |g(x) - l_g| < \varepsilon \end{array}\right.}{\left\{\ \forall\varepsilon > 0,\ \forall x\,\mathrm{near}\,a,\ |f(x) + g(x) - (l_f + l_g)| < \varepsilon\right.}$$

Motivating example (end)

To prove

$$f@a \to l_f \;\Rightarrow\; g@a \to l_g \;\Rightarrow\; (f+g)@a \to (l_f + l_g)$$

Improved filter reasoning:

$$\left\{\begin{array}{l} \forall \varepsilon > 0,\ \forall x \,\text{near}\, a,\ |f(x) - l_f| < \varepsilon \\ \forall \varepsilon > 0,\ \forall x \,\text{near}\, a,\ |g(x) - l_g| < \varepsilon \\ \varepsilon > 0 \qquad \text{regular intro} \end{array}\right. ,$$

$$\left\{\ \forall x \,\text{near}\, a,\ |f(x) + g(x) - (l_f + l_g)| < \varepsilon\right. .$$

Motivating example (end)

To prove

$$f@a \to l_f \;\Rightarrow\; g@a \to l_g \;\Rightarrow\; (f+g)@a \to (l_f + l_g)$$

Improved filter reasoning:

$$\left\{\begin{array}{l} \forall \varepsilon > 0,\ \forall x \,\text{near}\, a,\ |f(x) - l_f| < \varepsilon \\ \forall \varepsilon > 0,\ \forall x \,\text{near}\, a,\ |g(x) - l_g| < \varepsilon \\ \varepsilon > 0 \\ x \,\text{near}\, a, \qquad \underline{\text{near}}\ \text{intro} \end{array}\right. ,$$

$$\left\{\ |(f(x) - l_f) + (g(x) - l_g)| < \varepsilon\right. \quad .$$

Motivating example (end)

To prove

$$f@a \to l_f \;\Rightarrow\; g@a \to l_g \;\Rightarrow\; (f+g)@a \to (l_f + l_g)$$

Improved filter reasoning:

$$\frac{\begin{cases} \forall \varepsilon > 0,\ \forall x \,\mathrm{near}\, a,\ |f(x) - l_f| < \varepsilon \\ \forall \varepsilon > 0,\ \forall x \,\mathrm{near}\, a,\ |g(x) - l_g| < \varepsilon \\ \varepsilon > 0 \\ x \,\mathrm{near}\, a, \end{cases},}{\begin{cases} |f(x) - l_f| < \frac{\varepsilon}{2} \\ |g(x) - l_g| < \frac{\varepsilon}{2} \end{cases}.}$$

Motivating example (end)

To prove

$$f@a \to l_f \;\Rightarrow\; g@a \to l_g \;\Rightarrow\; (f+g)@a \to (l_f + l_g)$$

Improved filter reasoning:

$$\dfrac{\left\{\begin{array}{l} \forall \varepsilon > 0,\ \forall x \,\mathrm{near}\, a,\ |f(x) - l_f| < \varepsilon \\ \forall \varepsilon > 0,\ \forall x \,\mathrm{near}\, a,\ |g(x) - l_g| < \varepsilon \\ \varepsilon > 0 \end{array}\right.,}{\underline{\mathrm{near}}\ \mathrm{revert} \quad \left\{\begin{array}{l} \forall x \,\mathrm{near}\, a,\ |f(x) - l_f| < \frac{\varepsilon}{2} \\ \forall x \,\mathrm{near}\, a,\ |g(x) - l_g| < \frac{\varepsilon}{2} \end{array}\right. .}$$

Near tactics

- `near=> x`

  Introduces a near variable quantified by `\forall x \near F`.

- `near: x`

  Reverts a near variable to `\forall x \near F`.

Near tactics

- `near=> x`

  Introduces a near variable quantified by `\forall x \near F`.
  Should be integrated to regular intro patterns (Coq PR #7962).

- `near: x`

  Reverts a near variable to `\forall x \near F`.
  Should be integrated to regular discharge patterns (Coq Issue #8007).

Near tactics

- `near=> x`

  Introduces a near variable quantified by `\forall x \near F`.
  Should be integrated to regular intro patterns (Coq PR #7962).

- `near: x`

  Reverts a near variable to `\forall x \near F`.
  Should be integrated to regular discharge patterns (Coq Issue #8007).

- `near F => x`

  Gives an element near `F`.

- `end_near`

  Does garbage collection of unused evars.
  Should ultimately be transparent for the user (Coq Issue #8006).

Comparison with Procrastination

Both mechanisms are a generalization of `big_enough`.

“Matching” tactics:

Procrastination	Near
`begin defer assuming x`	`near F => x`
`defer`/`deferred`	`near: x`
`end defer`	`end_near`
`exploit deferred tac`	✗
✗	`near=> x`

Main differences:

	Procrastination	Near
Accumulated predicates	Proven to be valid at the end	Proven to belong to the filter when accumulated
Witnesses	Parameter of the predicates Global to the group	What makes the predicate belong to the filter One for each predicate
Mechanism	Lots of `Ltac`	Few `Ltac` + canonical structures

Selected applications

- Intermediate Value Theorem:
  128 loc in Coq → 57 loc.

```
Lemma IVT (f : R -> R) (a b v : R) : a <= b ->
  {in `[a, b], continuous f} -> minr (f a) (f b) <= v <= maxr (f a) (f b) ->
  exists2 c, c \in `[a, b] & f c = v.
```

- Double limit theorems:
  48 loc in Coquelicot → 12 loc.

```
Lemma flim_switch_1 {U : uniformType}
  F1 {FF1 : ProperFilter F1} F2 {FF2 : Filter F2}
  (f : T1 -> T2 -> U) (g : T2 -> U) (h : T1 -> U) (l : U) :
  f @ F1 --> g -> (forall x1, f x1 @ F2 --> h x1) -> h @ F1 --> l ->
  g @ F2 --> l.
```

- Cauchy completeness of function space:
  34 loc in Coquelicot → 10 loc.

```
Lemma fun_complete {T : choiceType} {U : completeType}
  (F : set (set (T -> U))) {FF : ProperFilter F} :
  cauchy F -> cvg F.
```

DEMO

Selected applications

TEASER

Differential chain rule:

76 loc in COQUELICOT, 56 loc in LEAN $\rightarrow$ 11 loc

Asymptotic comparison

Mathematical (ε/δ) definition, specialized at a neighborhood of 0:

f is a little-o of $e \Leftrightarrow$

$$\forall \varepsilon > 0,\ \exists \delta > 0,\ \forall x,\ |x| < \delta \Rightarrow |f(x)| \leq \varepsilon\,|e(x)|\,,$$

f is a big-$\mathcal{O}$ of $e \Leftrightarrow$

$$\exists k > 0,\ \exists \delta > 0,\ \forall x,\ |x| < \delta \Rightarrow |f(x)| \leq k\,|e(x)|\,.$$

Asymptotic comparison

Mathematical (ε/δ) definition, specialized at a neighborhood of 0:

$$f \underline{\text{ is a }} \text{little-}o \text{ of } e \Leftrightarrow \\ \forall \varepsilon > 0,\ \exists \delta > 0,\ \forall x,\ |x| < \delta \Rightarrow |f(x)| \leq \varepsilon\, |e(x)|\,,$$

$$f \underline{\text{ is a }} \text{big-}\mathcal{O} \text{ of } e \Leftrightarrow \\ \exists k > 0,\ \exists \delta > 0,\ \forall x,\ |x| < \delta \Rightarrow |f(x)| \leq k\, |e(x)|\,.$$

Coq definition:

```
Context {T : Type} {K : absRingType} {V W : normedModType K}.

Definition littleo (F : set (set T)) (f : T -> V) (e : T -> W) :=
  forall eps : R, 0 < eps ->
  \forall x \near F, `|[f x]| <= eps * `|[e x]|.

Definition bigO (F : set (set T)) (f : T -> V) (e : T -> W) :=
  \forall k \near +oo, \forall x \near F, `|[f x]| <= k * `|[e x]|.
```

Mathematical practice and Bachmann-Landau notations

In practice, we consider the little-o and big-$\mathcal{O}$ predicates as equalities.

- We want to write:

$$\begin{array}{lcl} f = o(e) & \text{and} & f = \mathcal{O}(e) \\ f(x) = o(e(x)) & \text{and} & f(x) = \mathcal{O}(e(x)) \\ f = g + o(e) & \text{and} & f = g + \mathcal{O}(e) \\ f(x) = g(x) + o(e(x)) & \text{and} & f(x) = g(x) + \mathcal{O}(e(x)) \end{array}$$

- We want to do arithmetic on little-o and big-$\mathcal{O}$:

$$-o(e) = o(e), \quad o(e) + o(e) = o(e), \quad o(e) + \mathcal{O}(e) = \mathcal{O}(e), \quad \ldots$$

- We want to substitute.

The trick

- Definition (little-o with explicit witness):

$$o\,(e)\,[h] := \begin{cases} h, & \text{if } \quad h \quad \text{is a little-}o\text{ of} \quad e \\ 0, & \text{otherwise} \end{cases}$$

- Parsing:

$$f = g + o\,(e) \quad \textbf{is parsed} \quad f = g + o\,(e)\,[f - g]$$

- Change of witness:

$$f = g + o\,(e)\,[f - g] \Leftrightarrow \exists h, f = g + o\,(e)\,[h]$$

- Display:

$$f = g + o\,(e)\,[h] \quad \textbf{is displayed} \quad f = g + o\,(e)$$

Applications

- Equivalence:

```
Notation "f ~_x g" := (f = g +o_x g)
```

- Differential:

```
Definition diff (F : filter_on V) (f : V -> W) :=
  (get (fun (df : {linear V -> W}) =>
  continuous df /\ forall x,
  f x = f (lim F) + df (x - lim F) +o_(x \near F) (x - lim F))).

Lemma diff_locallyxP (x : V) (f : V -> W) :
  differentiable x f <-> continuous ('d_x f) /\
  forall h, f (h + x) = f x + 'd_x f h +o_(h \near 0) h.
```

Differential chain rule

```
Fact dcomp (U V W : normedModType R)
  (f : U -> V) (g : V -> W) x :
  differentiable x f -> differentiable (f x) g ->

  forall h, g (f (h + x)) =
    g (f x) + ('d_(f x) g \o 'd_x f) h +o_(h \near 0) h.

Proof.
move=> df dg; apply: eqaddoEx => h.
rewrite diff_locallyx// -addrA diff_locallyxC// linearD.
rewrite addrA -addrA; congr (_ + _ + _).
rewrite diff_eq0 // ['d_x f : _ -> _]diff_eq0 //.
by rewrite {2}eqo0 add0x comp0o_eqox compo0_eqox addox.
Qed.
```

Conclusion

- Tools for stable local reasoning.
- Ease of use and similarity with pen and paper proofs.
- Tested in the Mathematical Components Analysis library and used on examples in robotics.
- Described in an article accepted for publication in the Journal of Formalized Reasoning[2].

[2] Preprint: `https://hal.inria.fr/hal-01719918`.

Future work

What's next:

- Manuel Eberl's multiseries for automated limits, little-o...
- Experiment with the little-o trick on lower/upper bounds, limits, derivatives, differentials...
- More analysis, more applications.

Improvements:

- Better integration of the near tactics with Coq intro/discharge patterns.
- Why should we split the epsilons?

Axioms

From the standard library of Coq:

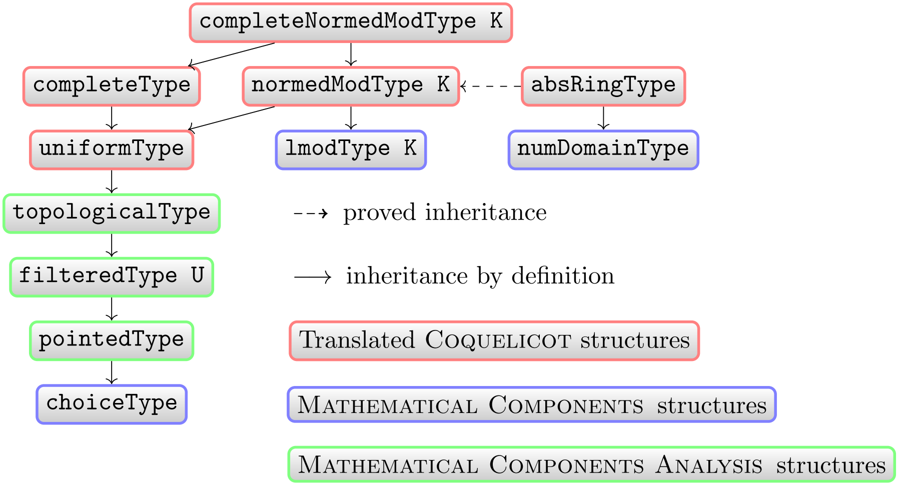
```
propositional_extensionality :
forall (P Q : Prop), P <-> Q -> P = Q.
functional_extensionality_dep :
forall (A : Type) (B : A -> Type) (f g : forall x : A, B x) :
  (forall x : A, f x = g x) -> f = g.
constructive_indefinite_description :
 forall (A : Type) (P : A -> Prop),
   (exists x : A, P x) -> {x : A | P x}.
```

We prove and use only:

```
propext : forall (P Q : Prop), (P <-> Q) -> (P = Q).
funext : forall {T U : Type} (f g : T -> U),
   (forall x, f x = g x) -> f = g.

pselect : forall (P : Prop), {P} + {~P}.
gen_choiceMixin : forall {T : Type}, Choice.mixin_of T.
```

Mathematical Components Analysis hierarchy



Canonical structures for filter inference

Two structures:

- `filteredType U`: interface type for types whose elements represent filters on `U`.
- `Filtered.source Y Z`: structure that records types X such that there is a function mapping functions of type `X -> Y` to filters on `Z`. Allows to infer the canonical filter associated to a function by looking at its source type: in particular useful for filters, sequences and sets in a normed space.

Example: double limit theorem

$$
\begin{aligned}
f &: T_1 \to T_2 \to U \\
g &: T_2 \to U \\
h &: T_1 \to U \\
l &: U
\end{aligned}
$$

$$
\begin{tikzcd}
f \arrow[r, "\text{uniform}"] \arrow[d, "\text{simple}"'] & g \arrow[d, dotted] \\
h \arrow[r] & l
\end{tikzcd}
$$

Justification:

$$\|l - g(x_2)\| \leq \|l - h(x_1)\| + \|h(x_1) - f(x_1, x_2)\| + \|f(x_1, x_2) - g(x_2)\|$$

ISABELLE/HOL's proof

```
lemma swap_uniform_limit:
  assumes f: "∀F n in F. (f n ⟶ g n) (at x within S)"
  assumes g: "(g ⟶ l) F"
  assumes uc: "uniform_limit S f h F"
  assumes "¬trivial_limit F"
  shows "(h ⟶ l) (at x within S)"
proof (rule tendstoI)
  fix e :: real
  define e' where "e' = e/3"
  assume "0 < e"
  then have "0 < e'" by (simp add: e'_def)
  from uniform_limitD[OF uc ‹0 < e'›]
  have "∀F n in F. ∀x∈S. dist (h x) (f n x) < e'"
    by (simp add: dist_commute)
  moreover
  from f
  have "∀F n in F. ∀F x in at x within S. dist (g n) (f n x) < e'"
    by eventually_elim (auto dest!: tendstoD[OF _ ‹0 < e'›] simp: dist_commute)
  moreover
  from tendstoD[OF g ‹0 < e'›] have "∀F x in F. dist l (g x) < e'"
    by (simp add: dist_commute)
  ultimately
  have "∀F _ in F. ∀F x in at x within S. dist (h x) l < e"
  proof eventually_elim
    case (elim n)
    note fh = elim(1)
    note gl = elim(3)
    have "∀F x in at x within S. x ∈ S"
      by (auto simp: eventually_at_filter)
    with elim(2)
    show ?case
    proof eventually_elim
      case (elim x)
      from fh[rule_format, OF ‹x ∈ S›] elim(1)
      have "dist (h x) (g n) < e' + e'"
        by (rule dist_triangle_lt[OF add_strict_mono])
      from dist_triangle_lt[OF add_strict_mono, OF this gl]
      show ?case by (simp add: e'_def)
    qed
  qed
  thus "∀F x in at x within S. dist (h x) l < e"
    using eventually_happens by (metis ‹¬trivial_limit F›)
qed
```

Coquelicot's proof (our benchmark)

```
Lemma filterlim_switch_1 {U : UniformSpace}
  F1 (FF1 : ProperFilter F1) F2 (FF2 : Filter F2) (f : T1 -> T2 -> U) g h (l : U) :
  filterlim f F1 (locally g) ->
  (forall x, filterlim (f x) F2 (locally (h x))) ->
  filterlim h F1 (locally l) -> filterlim g F2 (locally l).

Proof.
  intros Hfg Hfh Hhl P.
  case: FF1 => HF1 FF1.
  apply filterlim_locally.
  move => eps.

  have FF := (filter_prod_filter _ _ F1 F2 FF1 FF2).

  assert (filter_prod F1 F2 (fun x => ball (g (snd x)) (eps / 2 / 2) (f (fst x) (snd x)))).
    apply Filter_prod with (fun x : T1 => ball g (eps / 2 / 2) (f x)) (fun _ => True).
    move: (proj1 (@filterlim_locally _ _ F1 FF1 f g) Hfg (pos_div_2 (pos_div_2 eps))) => {Hfg} /= Hfg.
    by [].
    by apply FF2.
    simpl ; intros.
    apply H.
  move: H => {Hfg} Hfg.

  assert (filter_prod F1 F2 (fun x : T1 * T2 => ball l (eps / 2) (h (fst x)))).
    apply Filter_prod with (fun x : T1 => ball l (eps / 2) (h x)) (fun _ => True).
    move: (proj1 (@filterlim_locally _ _ F1 FF1 h l) Hhl (pos_div_2 eps)) => {Hhl} /= Hhl.
                              (* next page *)
```

Coquelicot' proof (page 2)

```
    by [].
    by apply FF2.
    by [].
  move: H => {Hhl} Hhl.

  case: (@filter_and _ _ FF _ _ Hhl Hfg) => {Hhl Hfg} /= ; intros.

  move: (fun x => proj1 (@filterlim_locally _ _ F2 FF2 (f x) (h x)) (Hfh x) (pos_div_2 (pos_div_2 eps))) => {Hfh}
         /= Hfh.
  case: (HF1 Q f0) => x Hx.
  move: (@filter_and _ _ FF2 _ _ (Hfh x) g0) => {Hfh}.
  apply filter_imp => y Hy.
```

End of boilerplate, and now, the meaningful part.

```
  rewrite (double_var eps).
  apply ball_triangle with (h x).
  apply (p x y).
  by [].
  by apply Hy.
  rewrite (double_var (eps / 2)).
  apply ball_triangle with (f x y).
  by apply Hy.
  apply ball_sym, p.
  by [].
  by apply Hy.
Qed.
```

Our proof

```
Lemma flim_switch_1 {U : uniformType}
  F1 {FF1 : ProperFilter F1} F2 {FF2 : Filter F2}
  (f : T1 -> T2 -> U) (g : T2 -> U) (h : T1 -> U) (l : U) :
  f @ F1 --> g -> (forall x1, f x1 @ F2 --> h x1) -> h @ F1 --> l ->
  g @ F2 --> l.
Proof.
move=> fg fh hl; apply/flim_ballPpos => e.
rewrite near_simpl; near F1 => x1; near=> x2.
apply: (@ball_split _ (h x1)); first by near: x1; apply/hl/locally_ball.
apply: (@ball_splitl _ (f x1 x2)); first by near: x2; apply/fh/locally_ball.
by move: (x2); near: x1; apply/(flim_ball fg).
Grab Existential Variables. all: end_near. Qed.
```

Comparison with Coquelicot

```
Lemma flim_switch_1 {U : uniformType} F1 {FF1 : ProperFilter F1} F2 {FF2 : Filter F2}
  (f : T1 -> T2 -> U) (g : T2 -> U) (h : T1 -> U) (l : U) :

  f @ F1 --> g -> (forall x, f x @ F2 --> h x) -> h @ F1 --> l -> g @ F2 --> l.
```

```
Proof.
  (*...*)
  (*25 lines of boilerplate, then*)

  rewrite (double_var eps).
  apply ball_triangle with (h x).
  apply (p x y).
  by [].
  by apply Hy.
  rewrite (double_var (eps / 2)).
  apply ball_triangle with (f x y).
  by apply Hy.
  apply ball_sym, p.
  by [].
  by apply Hy.
Qed.
```

```
Proof.
move=> fg fh hl; apply/flim_ballPpos => e.
rewrite near_simpl; near F1 => x1; near=> x2.
(* 2 lines of boilerplate, then 3 lines of actual proof *)

apply: (@ball_split _ (h x1)); first by near: x1; apply/hl/
       locally_ball.
apply: (@ball_splitl _ (f x1 x2)); first by near: x2; apply/fh/
       locally_ball.
by move: (x2); near: x1; apply/(flim_ball fg).

(* Finally: 1 line of boilerplate *)
Grab Existential Variables. all: end_near. Qed.
```