Asymptotic reasoning in Coq

Cyril Cohen *(Inria, France)*
j.w.w. Reynald Affeldt *and* Damien Rouhling

Applications of Formal Methods to Control Theory and Dynamical Systems, Pittsburgh

June 23, 2018

1

What this talk is about

Motivation

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

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

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

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

$$\ldots$$

Style for writing proofs

Declarative proof style:

- Lots of intermediate statements.
- Relying heavily on powerful automation.
- Script are human readable.
- Robustness guaranteed by appropriate choice of intermediate statements and automation.

Imperative proof style:

- Minimal amount of intermediate statements.
- Limited automation needed.
- Script contains orders you give to the system.
- Robustness guaranteed by the determinism in obeying orders.
  $\Rightarrow$ Most likely to happen with **small scale orders**.

Example: double limit theorem

$$f : T_1 \to T_2 \to U$$
$$g : T_2 \to U$$
$$h : T_1 \to U$$
$$l : U$$

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

Justification:

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

Analysis Libraries

In Coq

- C−CoRN
- Coq standard library + Coquelicot

Analysis Libraries

In Coq

- C−CoRN $\longrightarrow$ **Constructive analysis**
- Coq standard library + Coquelicot $\longrightarrow$ **Constructive** + $\mathbb{R}$ axioms

Analysis Libraries

In Coq

- C−CoRN $\longrightarrow$ **Constructive analysis**
- Coq standard library + Coquelicot $\longrightarrow$ **Constructive** + $\mathbb{R}$ axioms

Classical analysis in

- HOL Light
- Isabelle/HOL
- Lean
- ...

Analysis Libraries

In Coq

- C−CoRN $\longrightarrow$ **Constructive analysis**
- Coq standard library + Coquelicot $\longrightarrow$ **Constructive** + $\mathbb{R}$ axioms

Classical analysis in

- HOL Light
- Isabelle/HOL
- Lean
- ...

Analysis Libraries

In Coq

- C−CoRN $\longrightarrow$ **Constructive analysis**
- Coq standard library + Coquelicot $\longrightarrow$ **Constructive** + $\mathbb{R}$ axioms

Classical analysis in

- HOL Light
- Isabelle/HOL
- Lean
- ...

Disclaimer: the proofs I am going to show have clearly not been reworked to their best shape.

ISABELLE/HOL **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.
```

Achievement of our work

Design techniques to:

1. **Do imperative style small scale proofs**
2. Reduce the size of the boilerplate
3. Make it robust to change
4. Go straight to the point

2

Framework

Context

Constaints:

1. Robotics: kinematic chains as composition of MATHEMATICAL COMPONENTS matrices

$\Rightarrow$ Mix COQUELICOT and MATHEMATICAL COMPONENTS

2. Undergraduate classic textbook analysis
3. Catch up with ISABELLE/HOL and LEAN

$\Rightarrow$ COQUELICOT and Hilbert's epsilon

Context

Constaints:

1. Robotics: kinematic chains as composition of MATHEMATICAL COMPONENTS matrices

$\Rightarrow$ Mix COQUELICOT and MATHEMATICAL COMPONENTS

2. Undergraduate classic textbook analysis
3. Catch up with ISABELLE/HOL and LEAN

$\Rightarrow$ COQUELICOT and Hilbert's epsilon

Conclusion:

- rewrite COQUELICOT, on top of MATHEMATICAL COMPONENTS
- using stronger axioms:
  - Hilbert's epsilon (`constructive_indefinite_description`)
  - Propositional and functional extensionality

Hierarchy

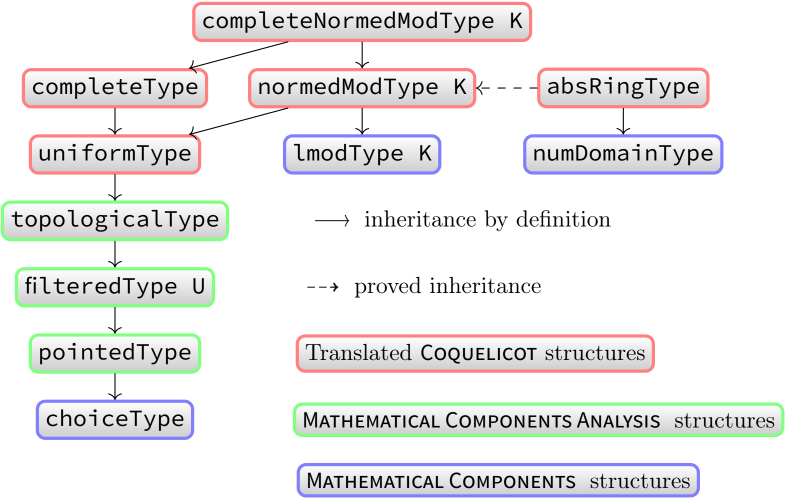

Figure: MATHEMATICAL COMPONENTS ANALYSIS hierarchy

3
Filters

A one-slide introduction to filters

Definition (same as Coquelicot in Coq)

$$\top \in F, \quad \forall A, B \in F.\, A \cap B \in F \quad \text{and} \quad \forall A, B.\, A \subseteq B \Rightarrow A \in F \Rightarrow B \in F.$$

Filter of neighborhoods:

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

A one-slide introduction to filters

Definition (same as COQUELICOT in COQ)

$$\top \in F, \quad \forall A, B \in F.\, A \cap B \in F \quad \text{and} \quad \forall A, B.\, A \subseteq B \Rightarrow A \in F \Rightarrow B \in F.$$

Filter of neighborhoods:

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

Filter application:

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

Limit:

$$f@F \to G := G \subseteq f@F.$$

Filter Notations

Definitions/notations:

`set A`	`A -> Prop`
``A `<=` B``	set inclusion
`[set a | P a]`	the set of elements `a` that satisfy `P`
`F --> G`	reverse set inclusion for filters $F \supseteq G$
`f @ F`	filter $f(F)$
`+oo`	$+\infty$
`\forall x \near x_0, P x`	$\mathrm{locally}(x_0)(P)$

Double Limit Theorem

DEMO!

```
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; first near=> x2.
- apply : ( @ball_split _ ( h x1 ) ) ; first by near: x1 .
  by apply: (@ball_splitl _ (f x1 x2)); [near: x2| move: (x2);
      near: x1].
- by end_near; apply/fh/locally_ball .
- by end_near; [exact/hl/locally_ball|exact/(flim_ball fg)].
Qed.
```

Double limit, 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; first near=> x2.
(* 2 lines of boilerplate, then *)

- apply: (@ball_split _ (h x1)); first by near: x1.
  by apply: (@ball_splitl _ (f x1 x2)); [near: x2|
       move: (x2); near: x1].

(* Two lines of boilerplate: *)
- by end_near; apply/fh/locally_ball.
- by end_near; [exact/hl/locally_ball|exact/(
     flim_ball fg)].
Qed.
```

Filter tactics

The lemmas that make it all work

```
Lemma filterP T (F : set (set T)) {FF : Filter F} (P : set T) :
  (exists2 Q : set T, F Q & forall x : T, Q x -> P x) <-> F P.

Lemma filter_ex T (F : set (set T)) `{ProperFilter F} :
  forall (Q : set T), F Q -> exists x : T, Q x.
```

Filter tactics

The lemmas that make it all work

```
Lemma filterP T (F : set (set T)) {FF : Filter F} (P : set T) :
  (exists2 Q : set T, F Q & forall x : T, Q x -> P x) <-> F P.

Lemma filter_ex T (F : set (set T)) `{ProperFilter F} :
  forall (Q : set T), F Q -> exists x : T, Q x.
```

Tactics	
`near=> x`	applies `filterP` with metavariable `Q`
`near F => x`	takes `x` from `filter_ex` with metavariable `Q`
`near: x`	given a goal ($\mathtt{R}_i$ `x`), accumulates $\mathtt{R}_i$ in `Q`
`end_near`	leaves accumulated (`F` $\mathtt{R}_i$) to be proven

Cauchy completeness

```
Definition cauchy_ex {T : uniformType} (F : set (set T)) :=
  forall eps : R, 0 < eps -> exists x, F (ball x eps).
```

or

```
Definition cauchy {T : uniformType} (F : set (set T)) :=
  forall e, e > 0 -> \forall x & y \near F, ball x e y.
```

equivalently

```
Definition cauchy_entourage T (F : set (set T)) :=
   (F, F) --> entourages.
```

Function space is complete

```
Lemma fun_complete (F : set (set (T -> U))) {FF: ProperFilter F} :
  cauchy F -> cvg F.
Proof.
move=> Fc; have /(_ _) /complete_cauchy Ft_cvg : cauchy (@^~_ @ F).
  by move=> t e ?; rewrite near_simpl; apply: filterS (Fc _ _).
apply/cvg_ex; exists (fun t => lim (@^~t @ F)).
apply/flim_ballPpos => e; near=> f => [t|].
  near F => g => /=.
    by apply: (@ball_splitl _ (g t)); last move: (t); near: g.
  by end_near; [exact/Ft_cvg/locally_ball|near: f].
by end_near; apply: nearP_dep; apply: filterS (Fc _ _).
Qed.
```

4

little-o and big-$\mathcal{O}$

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]|.
```

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]|.
```

But these are not predicates in the mathematical practice!

Use cases

We want to write:

- $$f = o\left(e\right) \quad \text{and} \quad f = \mathcal{O}\left(e\right)$$
  $$f(x) = o\left(e(x)\right) \quad \text{and} \quad f(x) = \mathcal{O}\left(e(x)\right)$$
- $$f = g + o\left(e\right) \quad \text{and} \quad f = g + \mathcal{O}\left(e\right)$$
  $$f(x) = g(x) + o\left(e(x)\right) \quad \text{and} \quad f(x) = g(x) + \mathcal{O}\left(e(x)\right)$$
- Do arithmetic on little-o and big-$\mathcal{O}$:
  $$-o\left(e\right) = o\left(e\right), \quad o\left(e\right) + o\left(e\right) = o\left(e\right), \quad o\left(e\right) + \mathcal{O}\left(e\right) = \mathcal{O}\left(e\right), \quad \ldots$$
- Substitute! (these are equalities)

Use cases

We want to write:

- $$f = o\,(e) \quad \text{and} \quad f = \mathcal{O}\,(e)$$
  $$f(x) = o\,(e(x)) \quad \text{and} \quad f(x) = \mathcal{O}\,(e(x))$$
- $$f = g + o\,(e) \quad \text{and} \quad f = g + \mathcal{O}\,(e)$$
  $$f(x) = g(x) + o\,(e(x)) \quad \text{and} \quad f(x) = g(x) + \mathcal{O}\,(e(x))$$
- Do arithmetic on little-o and big-$\mathcal{O}$:
  $$-o\,(e) = o\,(e)\,, \quad o\,(e) + o\,(e) = o\,(e)\,, \quad o\,(e) + \mathcal{O}\,(e) = \mathcal{O}\,(e)\,, \quad \ldots$$
- Substitute! (these are equalities)

DEMO!

The trick

Definition (little-o with explicit witness):

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

Parsing:

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

Change of witness:

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

The trick

Definition (little-o with explicit witness):

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

Parsing:

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

Change of witness:

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

Display:

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

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 ('d_F f) /\  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.
```

A short proof of the 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 => y.
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.
```

5

Conclusion and future work

Conclusion

- Toolset to give high-level orders, preserving determinism.
- Tested in the library
  `http://github.com/math-comp/analysis`

What I did not show:

- Tool for manifest positivity
- Lightweight automatic differentiation (Damien Rouhling, CPP 2018)

Incoming improvements

Improve the workflow of near tactics

Go from this

```
move=> fg fh hl; apply/flim_ballPpos => e; rewrite near_simpl.
near F1 => x1; first near=> x2.
- apply : (@ball_split _ (h x1)); first by near: x1.
  by apply: (@ball_splitl _ (f x1 x2)); [near: x2|move: (x2); near: x1].
- by end_near; apply/fh/locally_ball.
- by end_near; [exact/hl/locally_ball|exact/(flim_ball fg)].
```

Incoming improvements

Improve the workflow of near tactics

Go from this

```
move=> fg fh hl; apply/flim_ballPpos => e; rewrite near_simpl.
near F1 => x1; first near=> x2.
- apply : (@ball_split _ (h x1)); first by near: x1.
  by apply: (@ball_splitl _ (f x1 x2)); [near: x2|move: (x2); near: x1].
- by end_near; apply/fh/locally_ball.
- by end_near; [exact/hl/locally_ball|exact/(flim_ball fg)].
```

to this

```
move=> fg fh hl; apply/flim_ballPpos => e ; rewrite near_simpl; near F1 => x1 x2.
apply: (@ball_split _ (h x1)); first by near: x1; apply/fh/locally_ball.
apply: (@ball_splitl _ (f x1 x2)); first by near: x2; apply/hl/locally_ball.
by near: x1 (x2); apply/(flim_ball fg).
```

Other possible improvements

- Manuel Eberl's multiseries for automated limits, little-o, etc
- Semi-automated bounding tools
  (ingredients: same as big-$\mathcal{O}$ and manifest positivity)
- add Lebesgue integration and power series
- find limits, derivatives, differentials, integrals and converging sums in a semi-automated automated way.

Thank you for your attention.