Asymptotic reasoning in Coq

Fastrelax meeting, Sophia Antipolis, June 6, 2018

Cyril Cohen *(Inria, France)*

j.w.w. Reynald Affeldt *and* Damien Rouhling

1

What this talk is about

Motivation

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

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

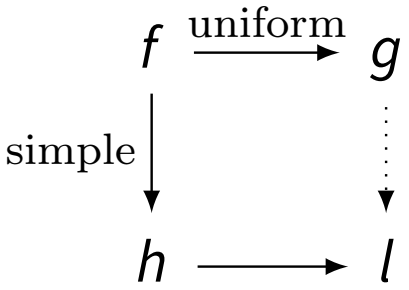
$$\underset{x \to 0}{o}(x^n) + \underset{x \to 0}{o}(x^n) = \underset{x \to 0}{o}(x^n)$$

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

$$\dots$$

Example: double limit theorem

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



Justification:

$$|l - gx_2| \leqslant |l - hx_1| + |hx_1 - fx_1x_2| + |fx_1x_2 - gx_2|$$

COQUELICOT as a 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 as a benchmark (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

1. Reduce the size of the boilerplate
2. Make it robust to change
3. Go straight to the point

2

Framework

Context

Constaints:

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

⇒ Mix COQUELICOT and MATHEMATICAL COMPONENTS

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

⇒ COQUELICOT and Hilbert's epsilon

Context

Constaints:

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

⇒ Mix COQUELICOT and MATHEMATICAL COMPONENTS

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

⇒ 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

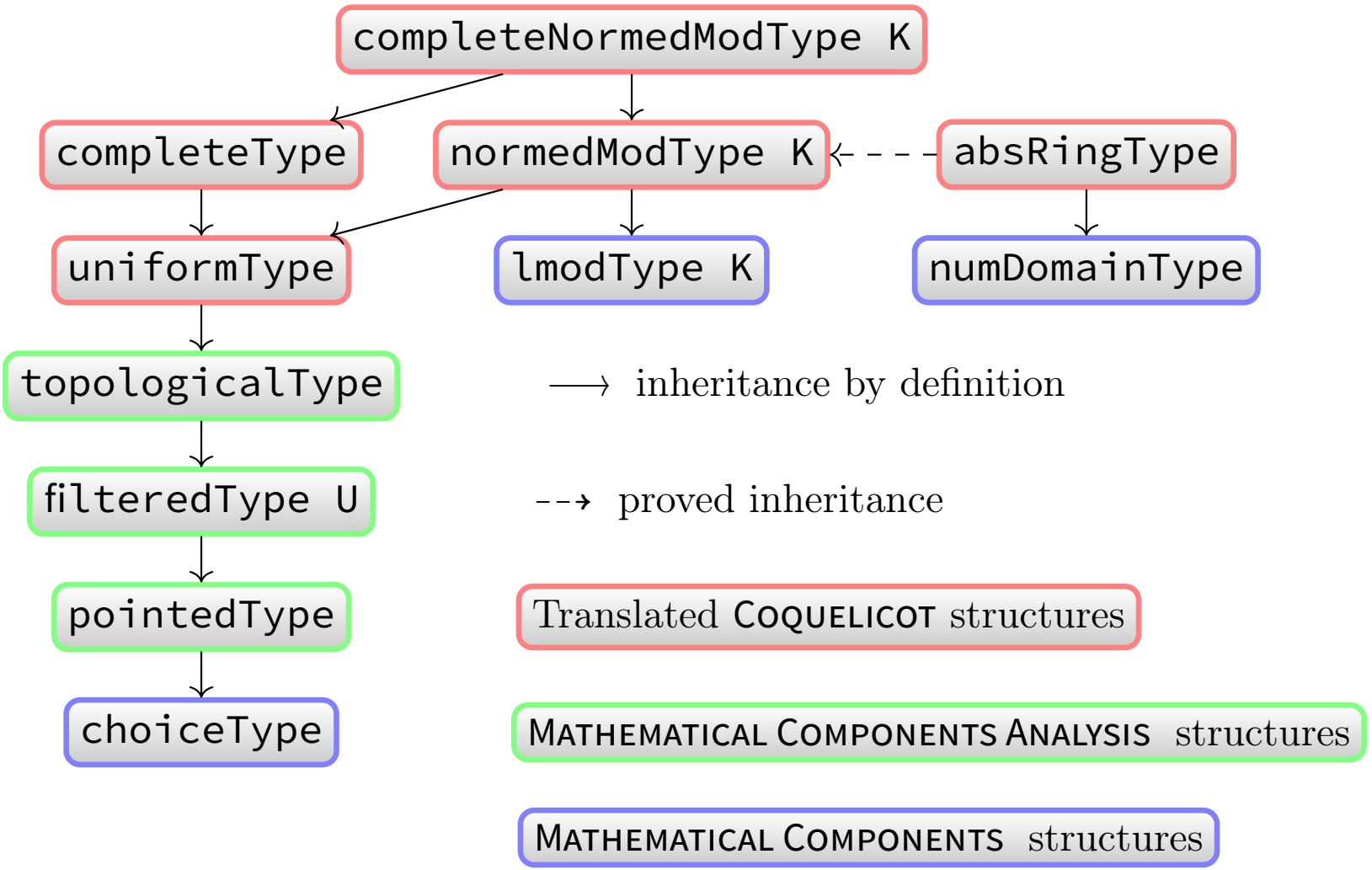


Figure: MATHEMATICAL COMPONENTS ANALYSIS hierarchy

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_ballP => _/posnumP[e]; rewrite !
    near_simpl.
near F1 have 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_ballP => _/posnumP[e];
      rewrite !near_simpl.
near F1 have 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 have x`	takes `x` from `filter_ex` with metavariable `Q`
`near: x`	given a goal $(R_i\ x)$, accumulates R_i in `Q`
`end_near`	leaves accumulated $(F\ R_i)$ 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

DEMO!

```
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_ballP => _ /posnumP[e]; near=> f => [t|].
  near F have 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:

- $$f = o\left(e\right) \quad \text{and} \quad f = \mathcal{O}\left(e\right)$$
- $$f = g + o\left(e\right) \quad \text{and} \quad f = g + \mathcal{O}\left(e\right)$$
- 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:

- $$f = o\left(e\right) \quad \text{and} \quad f = \mathcal{O}\left(e\right)$$
- $$f = g + o\left(e\right) \quad \text{and} \quad f = g + \mathcal{O}\left(e\right)$$
- 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)

DEMO!

The trick

Definition:

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

Lemma:

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

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}) => 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.
```

5

Conclusion and future work

Conclusion

- Tools to reason more like on paper
- Tested in the library
  `http://github.com/math-comp/analysis`

What I did not show:

- Tool for manifest positivity
- Lightweight automatic differentiation (Damien)

Wishes:

- Manuel Eberl's tool looks appealing
- Semi-automated bounding tools
  (ingredients: same as big-$\mathcal{O}$ and manifest positivity)
- Catch up with Isabelle/HOL, Lean and Coquelicot, and go beyond

Thank you for listening,
you may ask questions,
then lunch!