Classical analysis with Coq

Reynald Affeldt, Cyril Cohen *(Inria)*, Assia Mahboubi,
Damien Rouhling *and* Pierre-Yves Strub

Coq Workshop 2018, Oxford, UK

July 8, 2018

1

Context

Analysis in Proof Assistants

Formalizations of real analysis at a glance
[Boldo et al. 2016, Table 1]

	ACL2(r)	Coq	C-CoRN	HOL4	PPHol	Isabelle	HOLL	Mizar	PVS
Logic	FOL	deptypes	deptypes + constr	HOL	HOL	HOL	HOL	ZF	TCC
Partial functions	n.a.	total func + deptypes	deptypes	..... total functions, Hilbert-ε				function graphs	TCC
Definition of reals	non-std analysis	axiomatic	Cauchy sequences	Dedekind cuts	Dedekind cuts	Cauchy + UF	Cauchy sequences	Dedekind cuts	axiomatic
Integrals	n.a.	Riemann	Riemann Stieltjes	Gauge Lebesgue	Gauge	Gauge Lebesgue	Gauge Lebesgue	Riemann Lebesgue	Riemann Lebesgue
Multivariate analysis	1D	1D	2D	1D	1D	nD+	nD	nD+	2D
Dedicated automation	+	++	+	++	+	++	++		++

Analysis in Proof Assistants

Formalizations of real analysis at a glance
[Boldo et al. 2016, Table 1]

	ACL2(r)	Coq	C-CoRN	HOL4	PPHol	Isabelle	HOLL	Mizar	PVS
Logic	FOL	deptypes	deptypes + constr	HOL	HOL	HOL	HOL	ZF	TCC
Partial functions	n.a.	total func + deptypes	deptypes	..... total functions, Hilbert-ε				function graphs	TCC
Definition of reals	non-std analysis	axiomatic	Cauchy sequences	Dedekind cuts	Dedekind cuts	Cauchy + UF	Cauchy sequences	Dedekind cuts	axiomatic
Integrals	n.a.	Riemann	Riemann Stieltjes	Gauge Lebesgue	Gauge	Gauge Lebesgue	Gauge Lebesgue	Riemann Lebesgue	Riemann Lebesgue
Multivariate analysis	1D	1D	2D	1D	1D	nD+	nD	nD+	2D
Dedicated automation	+	++	+	++	+	++	++		++

Analysis in Coq and HOL languages

Let us add in the picture

- Coquelicot by [Boldo et al. 2015]
- Our work Mathematical Components Analysis
- New *criteria* (proof style, abstraction for asymptotic)

	Coq + Coquelicot	C-CoRN	HOL4	PPHol	Isabelle	HOLLight	Our library: Mathematical Components Analysis
Logic	deptypes	deptypes + constr	HOL	HOL	HOL	HOL	deptypes + classical axioms
Partial functions	total func + deptypes	deptypes	.....	total functions, Hilbert-ε			total functions, Hilbert-ε
Definition of reals	axiomatic	Cauchy sequences	Dedekind cuts	Dedekind cuts	Cauchy + UF	Cauchy sequences	dedicated structure + model(s) to be implemented
Integrals	Riemann Gauge	Riemann Stieltjes	Gauge Lebesgue	Gauge	Gauge Lebesgue	Gauge Lebesgue	to be determined
Multivariate analysis	2D	2D	1D	1D	nD+	nD	nD+
Dedicated automation	++	+	++	+	++	++	?
Proof script style	Imperative	Imperative	?	?	Declarative	?	Imperative + Small Scale
Abstraction Asymptotic	Filters	None	?	?	Filters	Nets	Filters

Analysis in Coq and HOL languages

Let us add in the picture

- Coquelicot by [Boldo et al. 2015]
- Our work Mathematical Components Analysis
- New *criteria* (proof style, abstraction for asymptotic)

	Coq + Coquelicot	C-CoRN	HOL4	PPHol	Isabelle	HOLLight	Our library: Mathematical Components Analysis
Logic	deptypes	deptypes + constr	HOL	HOL	HOL	HOL	deptypes + classical axioms
Partial functions	total func + deptypes + no funext	deptypes + no funext	..... extensional total functions, Hilbert-ε				extensional total functions, Hilbert-ε
Definition of reals	axiomatic	Cauchy sequences	Dedekind cuts	Dedekind cuts	Cauchy + UF	Cauchy sequences	dedicated structure + model(s) to be implemented
Integrals	Riemann Gauge	Riemann Stieltjes	Gauge Lebesgue	Gauge	Gauge Lebesgue	Gauge Lebesgue	to be determined
Multivariate analysis	2D	2D	1D	1D	nD+	nD	nD+
Dedicated automation	++	+	++	+	++	++	?
Proof script style	Imperative	Imperative	?	?	Declarative	?	Imperative + Small Scale
Abstraction Asymptotic	Filters	None	?	?	Filters	Nets	Filters

Mathematical Components ∪ Coquelicot

- Mathematical Components:
  - discrete mathematics (sequences, finite sets, graphs, etc)
  - algebra (group theory, ring, fields, linear algebra, Galois theory, character theory, etc)
  - Design principle: fewer definition, more combiners and lemmas, small scale reflection
- Coquelicot:
  - real analysis (reals and topology from std lib, sequences, series, Gauge integral),
  - no classical axioms other than the real axiomatic,
  - defines its own algebraic structures

Conclusion:

- Rewrite of Coquelicot with **stronger axioms**
- Fully compatible with Mathematical Components

`http://github.com/math-comp/analysis`

2

Ground for the development

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.
```

Structure for real numbers

Additional axioms compared to an Archimedean field:

```
Record mixin_of (R : archiFieldType) : Type := Mixin {

  sup : pred R -> R;

   _  : forall E : pred R, has_sup E -> sup E \in ub E;

   _  : forall (E : pred R) (eps : R), has_sup E ->
        0 < eps -> exists2 e : R, E e & (sup E - eps) < e;

   _  : forall E : pred R, ~ has_sup E -> sup E = 0
}.
```

Hierarchy

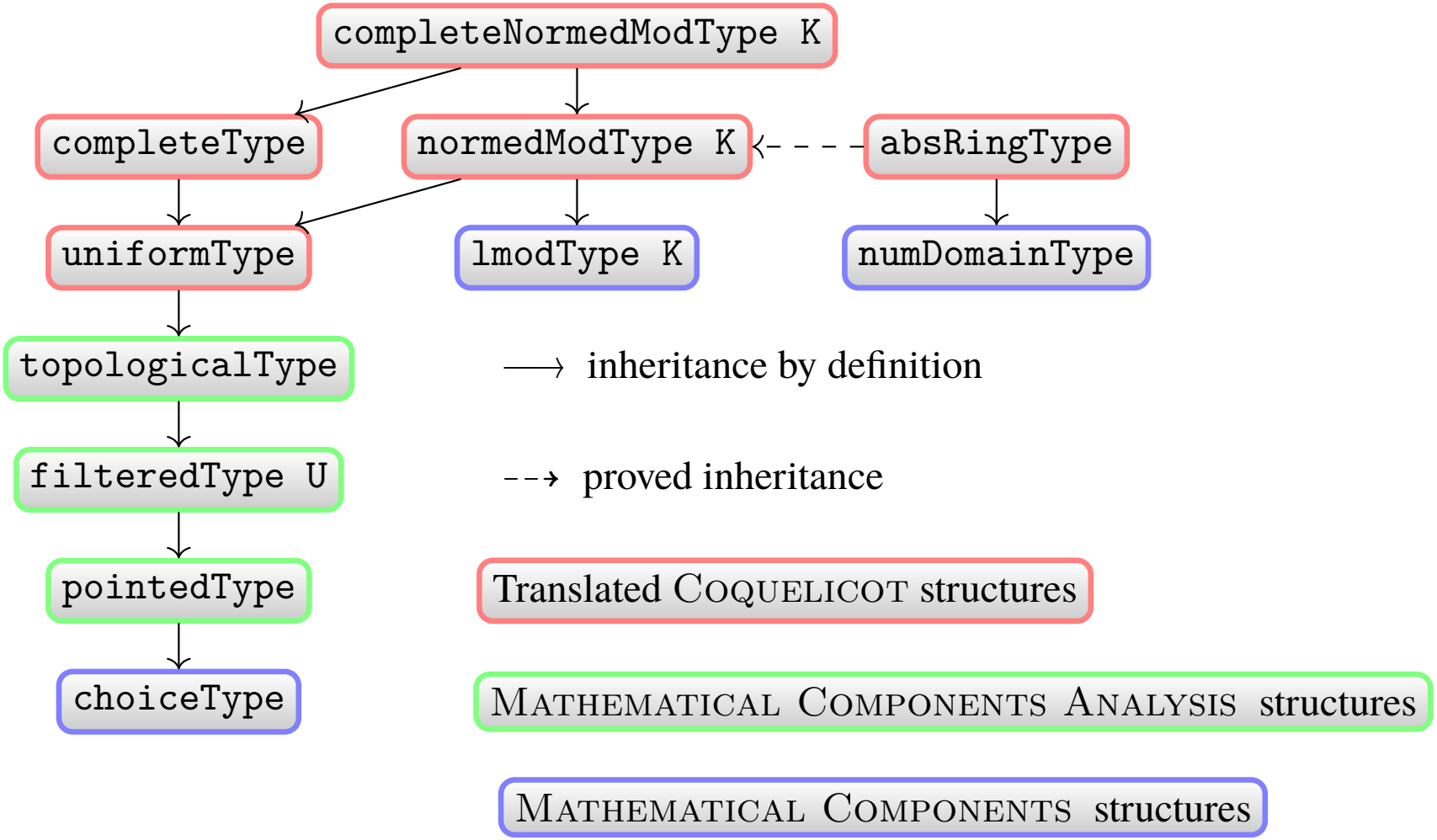


Figure: Mathematical Components Analysis hierarchy

3

A matter of style

Style for writing proofs

Declarative proof style:

- Lots of intermediate statements.
- Relying heavily on powerful automation.
- Scripts 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**.

Near tactics

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:

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

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

A one-slide introduction to filters

Definition (same as Coquelicot)

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

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)

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

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:

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

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`	$\mathrm{locally}(x_0)(P)$

Near tactics

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:

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

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

Near tactics

To prove

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

Typical ε/δ-reasoning:

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

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

Near tactics

To prove

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

Not so Typical ε/δ-reasoning:

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

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

Near tactics

To prove

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

Not so Typical ε/δ-reasoning:

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

$$\{\ \forall \varepsilon > 0,\ \forall x \,\text{near}\, a,\ |f(x) + g(x) - (l_f + l_g)| < \varepsilon.$$

Near tactics

To prove

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

Not so Typical ε/δ-reasoning:

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

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

Near tactics

To prove

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

Not so Typical ε/δ-reasoning:

$$\begin{cases} \forall \varepsilon > 0,\ \forall x \text{ near } a,\ |f(x) - l_f| < \varepsilon \\ \forall \varepsilon > 0,\ \forall x \text{ near } a,\ |g(x) - l_g| < \varepsilon \\ \varepsilon > 0, \\ x \text{ near } a, \qquad \textit{near} \text{ intro} \end{cases},$$

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

Near tactics

To prove

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

Not so Typical ε/δ-reasoning:

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

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

Near tactics

To prove

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

Not so Typical ε/δ-reasoning:

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

$$\begin{cases} |f(x) - l_f| < \frac{\varepsilon}{2} \\ |g(x) - l_g| < \frac{\varepsilon}{2}. \end{cases}$$

Near tactics

To prove

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

Not so Typical ε/δ-reasoning:

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

---

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

Near tactics

To prove

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

Not so Typical ε/δ-reasoning:

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

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

Near tactics

To prove

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

Not so Typical ε/δ-reasoning:

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

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

`Qed.`

Near tactics (continued)

Strict generalization of `big_enough`, complementary to `procrastination`.

Tactics	
`near=> x` `near_intro x`	introduces a *near* variable quantified by `\forall x \near F`
`near: x` `near_revert x`	reverts a *near* variable to `\forall x \near F`
`near F => x` `near F assert x`	gives an element near `F`

Near tactics (continued)

Strict generalization of `big_enough`, complementary to `procrastination`.

Tactics		
`near=> x` `near_intro x`	introduces a *near* variable quantified by `\forall x \near F`	should be integrated to regular intro patterns (Coq PR #7962)
`near: x` `near_revert x`	reverts a *near* variable to `\forall x \near F`	should be integrated to regular discharge patterns (Coq Issue #8007)
`near F => x` `near F assert x`	gives an element near `F`	

Near tactics (continued)

Strict generalization of `big_enough`, complementary to `procrastination`.

Tactics		
`near=> x` `near_intro x`	introduces a *near* variable quantified by `\forall x \near F`	should be integrated to regular intro patterns (Coq PR #7962)
`near: x` `near_revert x`	reverts a *near* variable to `\forall x \near F`	should be integrated to regular discharge patterns (Coq Issue #8007)
`near F => x` `near F assert x`	gives an element near `F`	
`end_near`	garbage collection of unused evar	should ultimately be transparent for the user (Coq Issue #8006)

4

Math Results

Topology and basic real analysis library I

- Tychonoff's Theorem using Zorn's Lemma (inspired by [Schepler ba]),

```
Lemma tychonoff (I : eqType) (T : I -> topologicalType)
  (A : forall i, set (T i)) : (forall i, compact (A i)) ->
  compact [set f : forall i, T i | forall i, A i (f i)].
```

- Heine-Borel's Theorem (only for 1D in Coq)

```
Lemma bounded_closed_compact n (A : set 'rV[R]_n.+1) :
  bounded A -> closed A -> compact A.
```

- the 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.
```

Topology and basic real analysis library II

- 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.
```

Topology and basic real analysis library III

- Differential chain rule (not in Coquelicot, 56 loc in Lean → 11 loc)

```
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.
```

- Summability theory (of infinite sequences)
- Discrete distribution theory.

A key tool: Bachmann-Landau notations

We 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)

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:

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

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 ('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 five lines 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 => 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.
```

5

Conclusion and future work

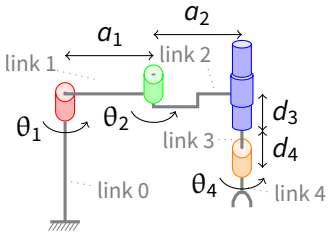
Applications

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

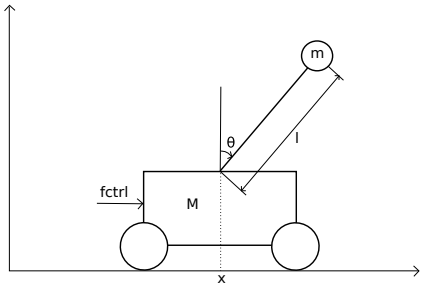
`https://github.com/affeldt-aist/coq-robot`

Applications

The inverted pendulum

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

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



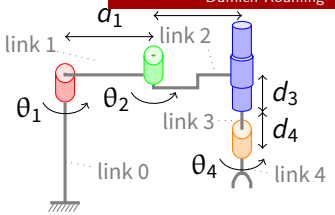
- Goal: stabilize the pendulum on its unstable equilibrium thanks to the control function fctrl.

What is a Robot Manipulator?

- E.g., SCARA (Selective Compliance Assembly Rc

Mitsubishi RH-S series

Sche



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

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

Applications

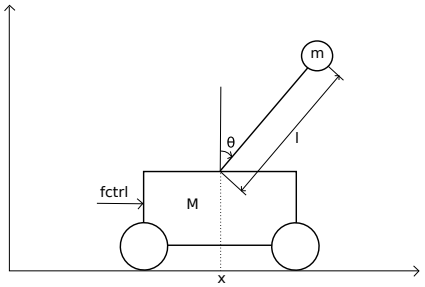
Probabilistic relational logic for

`https://github.com/strub/xhl`

The inverted pendulum

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

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



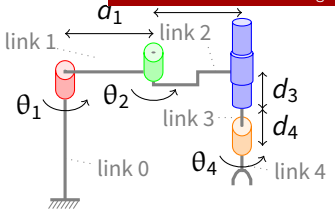
- Goal: stabilize the pendulum on its unstable equilibrium thanks to the control function fctrl.

What is a Robot Manipulator?

- E.g., SCARA (Selective Compliance Assembly Rc

Mitsubishi RH-S series

Sche



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

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

Improvements

Tools:

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

Results:

- uniformize the library (integration of discrete probabilities and sequences)
- reincorporate Perron-Frobenius theorem [Cano 2014]
- add Lebesgue(?) integration and power series

Question: switch from *Axiom of choice* to *Unique choice*?

Thank you for your attention.

References I

Boldo, S., Lelay, C., and Melquiond, G. (2015). Coquelicot: A user-friendly library of real analysis for Coq. *Mathematics in Computer Science*, 9(1):41–62.

Boldo, S., Lelay, C., and Melquiond, G. (2016). Formalization of real analysis: a survey of proof assistants and libraries. *Mathematical Structures in Computer Science*, 26(7):1196–1233.

Cano, G. (2014). *Interaction between linear algebra and analysis in formal mathematics*. Theses, Université Nice Sophia Antipolis.

Schepler, D. Topology library. `https://github.com/coq-contribs/topology`.

Schepler, D. Zorn's Lemma. `https://github.com/coq-contribs/zorns-lemma`.

7

Suplementary material

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:

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

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}
```

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.
```

Double Limit Theorem

```
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.
```

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; 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.
```

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 => /=.
apply: (@ball_splitl _ (g t)); first by near: g; exact/Ft_cvg/locally_ball.
by move: (t); near: g; near: f; apply: nearP_dep; apply: filterS (Fc _ _).
Grab Existential Variables. all: end_near. Qed.
```

Improvements wishes

Improve the workflow of near tactics

Go from this

```
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).
```

Improvements wishes

Improve the workflow of near tactics

Go from this

```
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).
```

---

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 move: x1 (x2); apply/(flim_ball fg).
```
