



Formalizing concentration inequalities in Rocq: infrastructure and automation

Reynald Affeldt  

National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, Japan

Alessandro Bruni  

IT University of Copenhagen, Denmark

Cyril Cohen  

Inria, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP, UMR 5668, Lyon, France

Pierre Roux  

ONERA/DTIS, Université de Toulouse, France

Takafumi Saikawa  

Nagoya University, Japan

Abstract

Concentration inequalities are standard lemmas providing upper bounds on deviations of random variables. To formalize concentration inequalities, we have been developing a general library of lemmas for probability theory in the Rocq prover. This effort led us to revisit already established technical aspects of the Mathematical Components libraries. In this paper, we report on improvements of general interest resulting from our formalization. We devise types for numeric values and a lightweight semi-decision procedure, based on interval arithmetic. We also extend the hierarchy of available mathematical structures to formalize Lebesgue spaces. We illustrate our new formalization of probability theory with the complete proof of a concentration inequality for Bernoulli sampling.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Type theory; Mathematics of computing → Probability and statistics

Keywords and phrases Rocq prover, Mathematical Components library, abstraction interpretation, probability theory, concentration inequalities

Digital Object Identifier [10.4230/LIPIcs.ITP.2025.21](https://doi.org/10.4230/LIPIcs.ITP.2025.21)


Supplementary Material This paper is accompanied by a ROCQ development, see details in Fig. 2.

Acknowledgements The authors are grateful to S. Sonoda for his inputs. The authors would like to thank the anonymous referees of ITP 2025 for their careful and informative review. The first and last authors acknowledge support of the JSPS KAKENHI Grant Number 22H00520, the second author acknowledges support of the Carlsberg Foundation, grant CF24-2347.

1 Introduction

Our motivation is the formalization of concentration inequalities in the ROCQ prover [35]. Concentration inequalities are inequalities of the form $\Pr[\mathbf{X} \geq t] \leq b$, which bound the probability of a random variable \mathbf{X} being greater than a given threshold t . The random variable \mathbf{X} is typically a distance between a more primitive random variable X and some integral involving X , thus representing the deviation of X from one of its characteristic values. A prime example is Chebyshev’s inequality $\Pr[|X - \mathbb{E}[X]| \geq t] \leq \frac{\mathbb{V}[X]}{t^2}$. Concentration inequalities are used in a wide variety of contexts [13] with concrete applications, e.g., the verification of worst-case failure probability of real-time systems [25], risk-limiting audits [31], etc.

The theory of concentration inequalities involves technical developments relying on discrete

 © Reynald Affeldt, Alessandro Bruni, Cyril Cohen, Pierre Roux, and Takafumi Saikawa; licensed under Creative Commons License CC-BY 4.0

16th International Conference on Interactive Theorem Proving (ITP 2025).

Editors: Yannick Forster and Chantal Keller; Article No. 21; pp. 21:1–21:21

 [Leibniz International Proceedings in Informatics](https://www.leibniz-proceedings.de/)
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

mathematics, real analysis, and probability theory. We believe that the complexity of the formal development of a full-fledged theory of concentration inequalities requires a solid infrastructure with a fair share of automation. The Mathematical Components library [24] (hereafter, MATHCOMP) is well-known to provide tooling that takes advantage of the type system of the ROCQ prover to efficiently formalize hierarchies of mathematical structures (as implemented by the HIERARCHY-BUILDER tool [16]). In addition, the ROCQ prover provides automation in the form of verified decision procedures for arithmetic [34] and tactics for numerical computations [27, 28]. This automation should come in handy when dealing with the analytical arguments that are inevitably part of the proofs of concentration inequalities. Yet, existing accounts of probability theory in ROCQ do not seem to lend themselves easily to extension: INFOtheo [21] is restricted to probabilities supported by a finite set, FormalML [36] relies on a formalization of real analysis with limited support for measure theory [12], coq-proba [33] is limited to discrete probability spaces. With this respect, the more recent MATHCOMP-ANALYSIS library [1], which extends MATHCOMP and provides a rich formalization of real analysis and measure theory [?, 2, 22], appears like a good candidate to develop a more versatile formalization of probability theory.

In this paper, we report on the development of probability theory on top of MATHCOMP and MATHCOMP-ANALYSIS. We contribute improvements of general interest for concrete applications of formal probability theory and real analysis. Indeed, our new development of probability theory led us to revisit already established technical aspects of MATHCOMP and MATHCOMP-ANALYSIS.

Our first improvement is the extension of the existing infrastructure of MATHCOMP-ANALYSIS to deal automatically with positive and non-negative numbers [3, Appendix A.2.1]. We devise types for numeric values and a lightweight semi-decision procedure, based on interval arithmetic. Among these types, the type for numbers between 0 and 1 appears naturally in probability theory, in particular when formalizing Minkowski’s inequality, which in turn is used in the construction of *Lebesgue spaces*. Lebesgue spaces refer either to spaces of measurable functions whose absolute value raised to the p th power has a finite integral (hereafter, \mathcal{L}^p spaces), or to the quotient of those by equality almost everywhere (hereafter, L^p spaces). They are pervasive in probability theory, in particular to express succinctly integrability conditions. To complete the formalization of \mathcal{L}^p and L^p spaces, we develop the theory of essential supremums/infimums and extend the MATHCOMP hierarchy of mathematical structures to support seminorms, in a conservative manner thanks to the HIERARCHY-BUILDER tool [16]. These two improvements (types for numeric values within a range and the lemmas used to formalize \mathcal{L}^p and L^p spaces) are useful additions beyond probability theory (for example, \mathcal{L}^p and L^p spaces are central in functional analysis).

Finally, we illustrate the resulting library of probability theory with a formalization of a sampling theorem [29, Exercise 4.5], which is an example of a concentration inequality itself relying on standard concentration inequalities. We aim to show with this example that we do provide an infrastructure that makes the formal verification of concentration inequalities practical, for example by leveraging on automation to simplify unavoidable analytical arguments.

Outline

In Sect. 2, we explain library support to equip numeric values with a type that contains range information. In particular, an instance of this type is used to formalize convexity in Sect. 2.4. Sect. 3 is a brief overview of measure theory and its notations. In Sect. 4, we explain how we formalize \mathcal{L}^p spaces. In particular, we explain in Sect. 4.3.2 how we use our

formalization of convexity to prove Minkowski’s inequality, used to establish the properties of \mathcal{L}^p spaces. In Sect. 5, we provide an overview of how we use the Lebesgue integral to formalize the basics of probability theory. Our main use-case is a formalization of a sampling theorem in Sect. 6. We review related work in Sect. 7 and conclude in Sect. 8.

2 Automation for numeric interval inference

When formalizing analysis results, it is common to encounter many side-proofs about the sign of simple expressions. For instance when doing continuity proofs, one often manipulates ε variables that are positive [3, Appendix A.2.1]. Similarly, here, results about convexity involve λ parameters that must remain in the $[0, 1]$ interval. For instance, for any $x \in [0, 1]$, it is obvious that $1 - x^n$ also lies in that $[0, 1]$ interval, for any $n \in \mathbb{N}$. Repeatedly stumbling on this kind of side goals can rapidly become cumbersome when doing formal proofs. That is why we automate these proofs. We do so by using canonical structure inference¹ to perform interval arithmetic [39]. More precisely, we first define a structure and then canonical instances of it for the operators of interest, thus providing interval abstractions for these operators. The convenience of the abstract interpretation [17] framework here is that we only need to prove correctness, not optimality of the abstractions. This means that there is no need to implement upfront a complete set of instances. Very coarse instances can already give worthwhile results, for a light proof effort. This initial set of instances can then be later refined on-demand.

2.1 Interval structure

We went through several implementations of a semi-decision procedure to automatically prove some positivity and non-negativity goals, extending it successively to cover negativity and non-positivity² and then membership to arbitrary integer intervals³, thus covering more use cases for the same price.

We build on top of the `interval` type already provided by MATHCOMP [?, `interval.v`], that is⁴:

```
Variant itv_bound (T : Type) : Type := BSide of bool & T | BInfty of bool.
Variant interval (T : Type) := Interval of itv_bound T & itv_bound T.
```

In the context of definition of intervals, the boolean parameter of `BSide` indicates whether the bound is open or closed, depending on its side, whereas the one of `BInfty` distinguishes $+\infty$ and $-\infty$. For instance, the interval $[0, +\infty[$ is encoded as `Interval (BSide true 0) (BInfty int false)`. We first define a type for our intervals. Since we may want to handle non-real values (for instance complex numbers), we just add a `Top` case to encompass any value, possibly non-real.

```
Module Itv. (* the definition of this module continues next page *)
Variant t := Top | Real of interval int.
```

¹ A restricted form of type classes offered by Coq since almost three decades.

² <https://github.com/math-comp/analysis/pull/511>

³ <https://github.com/math-comp/analysis/pull/1410>

⁴ The use of the ampersand (&) to write inductive types comes from the SSREFLECT extension of ROCq; see “Anonymous arguments” in [?].

Note the use of the type `int` for integers from MATHCOMP, based on the unary ROCQ `nat` type for natural numbers. At first sight, this might look restrictive and potentially yielding performance issues. However, in our work, we only manipulate bounds with small integer values such as 0 or 1, so these issues do not arise. We plan to generalize our approach to rational bounds and introduce more efficient encodings in the future.

The interval type `t` is equipped with the usual inclusion order

```
Definition sub (x y : t) := match x, y with _, Top => true | Top, Real _ => false
| Real xi, Real yi => subitv xi yi end.
```

using `subitv`, defined in MATHCOMP as `[?, interval.v]`:

```
Definition subitv i1 i2 :=
  let: Interval b1l b1r := i1 in let: Interval b2l b2r := i2 in
  (b2l <= b1l) && (b1r <= b2r).
```

The semantics of values of type `T` within a given interval is parameterized. We provide semantics for several types: `numDomainType`⁵, `nat`, and extended real numbers \mathbb{R} .

```
1 Section Itv_def.
2 Context T (sem : interval int -> T -> bool).
3 Definition spec (i : t) (x : T) := if i is Real i then sem i x else true.
4 Structure def (i : t) := Def { r : T; P : spec i r }.
5 End Itv_def.
```

The `def` type (line 4) is the main type for “value `r` within an interval `i`” where the `P` field is the proof of $r \in i$. The specifics of that property depend on the `sem` parameter, for instance, for natural numbers it is

```
Definition nat_sem (i : interval int) (x : nat) : bool := Posz x \in i.
```

where `Posz` is a constructor for integers, which is semantically the canonical injection from natural numbers into integers. The most commonly used semantics will be the one for `numDomainType`, which, for historical reasons, is the basic MATHCOMP type for algebraic structures with a compatible order relation⁶:

```
Definition num_sem (R : numDomainType) (i : interval int) (x : R) : bool :=
  (x \in Num.real) && (x \in map_itv intr i).
```

where `x \in Num.real` means that `x` is comparable to 0 (i.e., either $x \leq 0$ or $0 \leq x$) and `intr` is a notation for the ROCQ function `intmul`, which provides the canonical injection from integers to any `ringType`.

Finally, we use the *phantom* type `phx` to trigger inference of canonical instances for the above `def` structure. As usual with phantom types, `phx` is not used in the body of the definition but is here only to explicitly provide the `r x` value that will trigger the inference of the `x` interval containing it (@ below is a ROCQ notation to disable implicit arguments).

⁵ A MATHCOMP type for integral domains (i.e., commutative rings with an inverse and such that for any x and y , having $xy = 0$ implies $x = 0$ or $y = 0$) with a compatible order.

⁶ As future work, we could generalize this, for instance to additive semigroups with a compatible order, which would enable us to get rid of the current duplication for addition on `nat`, `numDomainType`, and extended real numbers.

```

Definition from {T f i} {x : @def T f i} (phx : phantom T (r x)) := x.
Notation "e %:itv" := (from (Phantom _ e)) : ring_scope.
End Itv. (* module started in the previous page *)

```

Thus, the notation `e%:itv` enables to infer a value of type `Itv.def` whose field `Itv.r` is `e`, that is essentially an interval `i` along with a proof that `e \in i`.

Note that the module `Itv` is closed at this stage and definitions inside can be referred to by using `Itv` as a prefix from now.

2.2 Canonical instances

Most of the work now requires registering canonical instances for each operator one may want intervals to be inferred for. When encountering unknown operators or variables, a default canonical instance will infer a broad interval based on the type of the expression, for instance $[0, +\infty[$ for natural numbers.

As a first example, we declare that 1 lies in the $[1, 1]$ interval:

```

(* in Section NumDomainInstances *)
Context {R : numDomainType}.
Lemma num_spec_one : Itv.spec (@Itv.num_sem _) (Itv.Real `[1, 1]) (1 : R).
Proof. by apply/andP; split; [exact: real0 | rewrite /= in_itv/= lexx]. Qed.
Canonical one_inum := Itv.Def num_spec_one.

```

More elaborate operators can rely on more elaborate programs, as a simple example, let us look at the implementation of the opposite operator (below, `%:num` is the `Itv.r` projection of the `Itv.def` structure, keeping only the value `r` while dropping both the interval `i` and the proof `P` that $r \in i$):

```

(* in Module IntItv *)
Implicit Types (b : itv_bound int) (i : interval int).
Definition opp_bound b := match b with
| BSide b x => BSide (~~ b) (intZmod.oppz x)
| BInfty b => BInfty _ (~~ b) end.
Definition opp i := let: Interval l u := i in Interval (opp_bound u) (opp_bound l).
Lemma num_spec_opp (i : Itv.t) (x : num_def R i) (r := Itv.real1 opp_itv i) :
  Itv.spec (@Itv.num_sem _) r (- x%:num).
Proof. (* a few lines omitted for conciseness *) Qed.
Canonical opp_inum (i : Itv.t) (x : num_def R i) := Itv.Def (num_spec_opp x).

```

Thus, for instance `(- 1)%:itv` will trigger the computation of the opposite interval of $[1, 1]$, that is $[-1, -1]$. Many more instances are provided for usual operators.

Compared to an implementation based on a reification tactic (implemented for instance in `Ltac` or `Ltac2`), this lightweight solution saves us the implementation of a reification tactic (we are simply using ROCQ's elaborator) and enables to handle arbitrary new constructs locally, by simply adding a canonical instance, rather than having to edit a monolithic tactic.

2.3 Hints

To automatically solve subgoals that appear to be “trivially” true by interval arithmetic, we add a few lemmas and hints like:

```

Lemma gt0 e : unify_itv i (Itv.Real `]0%Z, +oo[]) -> 0 < e%:num := R.
Hint Extern 0 (is_true (0 < _)%R) => solve [apply: gt0] : core.
(* and similarly for lt0, ge0, le0, cmp0, neq0, lt1 and le1 *)

```

where `unify_itv` uses some ROCQ type class option to trigger the computation of interval inclusion `Itv.sub` (see Sect. 2.1). Recall that the notation `%:num` has been explained in Sect. 2.2. The notation scopes `%Z` and `%R` are respectively for integers and rings.

In the whole MATHCOMP-ANALYSIS library, these hints are triggered 1604 times, automatically solving 729 subgoals. Here are a few examples of such automatically solved goals:

$0 \leq k\%:R / 2^{n+1}$	$0 \leq \frac{k}{2^n}$
$0 < (2^n)\%:R$	$0 < 2^n$
$0 < 1 + r $	$0 < 1 + r $
$0 < (3 + d.*2).*2.+2\%:R * (3 + d.*2).*2.+1\%:R$	$0 < (2(3 + 2d) + 2)(2(3 + 2d) + 1)$
$0 < 1 / (2^{i+1})\%:R$	$0 < 1/2^{i+1}$
$0 < n.+1\%:R^{-1} / 2$	$0 < (n+1)^{-1}/2$
$0 < (2 / e\%:num)^{n+2}$ with e non negative	$0 < \left(\frac{2}{e}\right)^2$ with $0 \leq e$

The notation `%:R` is a MATHCOMP notation for injection from `nat` to any `ringType` (like `intr` above for `int`).

2.4 Application to convexity

We learn from previous work that dealing with probabilities benefits from having a theory of *convex spaces*, to represent, among others, convex functions [6, Sect. 3.3]. A convex space [32] is a mathematical structure with a ternary operator that we write $a \lt| p \gt| b$. This operator represents the convex combination of two points a and b with parameter p , a real number between 0 and 1. This operator satisfies a few laws [4, Sect. 2]: skewed commutativity, quasi-associativity, etc. The side condition on p is better handled with a dedicated type for the interval $[0, 1]$ that we can define using `Itv.def` (Sect. 2.1):

```

Notation "{ 'i01' R }" := (Itv.def (@Itv.num_sem R) (Itv.Real `[0, 1]%Z)).

```

This leads to the following definition of convex function [8, `convex.v`]:

```

Definition convex_function (R : realType) (D : set R) (f : R -> R) :=
  forall t : {i01 R}, {in D &, forall x y : R, f (x <| t |> y) <= f x <| t |> f y}.

```

Using convex spaces and convex functions from MATHCOMP-ANALYSIS, we have been able to port results such as the convexity of the exponential function and the concavity of the logarithm function from existing work [4, Sect. 6.2].

3 Background: measure theory in MathComp-Analysis

This section provides a brief overview of notions of measure theory as formalized in MATHCOMP-ANALYSIS.

In MATHCOMP-ANALYSIS, a set can be defined by comprehension with the notation `[set x | P x]` where P is a `Prop`-valued predicate. In particular, the whole set of elements of type T is noted `[set: T]`. A σ -algebra is represented by an object of type `measurableType d` (d is a parameter used to control notations [2, Sect. 3.4]) that consists of a set of sets (i.e., a *set system*) that contains the empty set and that is closed under complement and countable union [1, `measure.v`]. The set system of a σ -algebra T is represented by the expression

`measurable T`. The type of a *measurable function* is noted $\{\text{mfun } T \rightarrow T'\}$. Alternatively, a measurable function can also be represented by a standard ROCQ function between two measurable types that satisfies the predicate `measurable_fun`. A *measure* μ is represented by an object of type $\{\text{measure } \text{set } T \rightarrow \bar{\mathbb{R}}\}$ where T is an object of type `measurableType` and that satisfies the following properties: $\mu(\emptyset) = 0$, $0 \leq \mu(A)$ for any subset A of type T , and σ -additivity, i.e., $\mu(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} \mu(A_i)$ for countably many pairwise disjoint A_i 's. Given a type R of real numbers, the notation $\bar{\mathbb{R}}$ corresponds to the type of extended real numbers; semantically, it is $\mathbb{R} \cup \{\pm\infty\}$. The notation $\%:E$ (corresponding to the type constructor `EFin`) denotes the inclusion from R into $\bar{\mathbb{R}}$. A *probability measure* is a measure whose value for the whole set is 1. It is given the type `probability T R` where T is a `measurableType` and R a type for real numbers. The *pushforward measure* of a measure μ by the function f is noted `pushforward mu f`. The Lebesgue integral with measure μ of a function f is noted `\int[mu]_(x in A) f x` [2, Sect. 6.4].

4 Formalization of \mathcal{L}^p spaces

In this section, we explain our formal definition of \mathcal{L}^p and L^p spaces. Informally, an \mathcal{L}^p space is the set of all functions whose p th power has a finite integral. The measure used for the integral induces an almost everywhere equality, which allows us to take a quotient. The resulting quotient set is the corresponding L^p space. \mathcal{L}^p and L^p spaces are used in functional analysis and probability theory to represent integrable functions (\mathcal{L}^1), square-integrable functions (\mathcal{L}^2), etc., i.e., functions whose *pth-norm* is integrable.

The parameter p ranges over real numbers greater than or equal to 1, as well as the positive infinity ∞ . We begin with the infinity case, where the definition relies on the notion of essential supremum.

4.1 The essential supremum, an adjunction

The traditional definition of the essential supremum [18] is as an infimum of preimages of final segments of $\bar{\mathbb{R}}$ of measure zero: $\text{esssup}_\mu f = \inf \{y \mid \mu(f^{-1}([y, +\infty])) = 0\}$. This historical version, which is still in use in many books, hides its order-theoretic (or more generally, category-theoretic) nature. Moreover, it requires that various preimages of f are measurable, unnecessarily forcing f to be measurable.

However, we only ever need its universal property: $\text{esssup}_\mu f \leq y \iff f \stackrel{\text{a.e.}}{\leq}_\mu \Delta(y)$, where $\Delta(y)$ is the constant function valued at y . Regarding the order structures $(\bar{\mathbb{R}}, \leq)$ and $(T \xrightarrow{\mu\text{-meas.}} \bar{\mathbb{R}}, \leq_\mu^{\text{a.e.}})$ as categories, it is the (unique) left adjoint to the constant functor Δ sending elements of $\bar{\mathbb{R}}$ to the space of measurable functions with the almost everywhere order relatively to μ , i.e., $(T \xrightarrow{\mu\text{-meas.}} \bar{\mathbb{R}}) \xrightleftharpoons[\Delta]{\text{esssup}_\mu} \bar{\mathbb{R}}$.

In ROCQ/MATHCOMP, we follow the folklore on adjunctions to define the essential supremum, instead of using the above mentioned historical definition. Indeed, we use a construction which would have been an instance of adjoint functor theorem for preorders, had we formalized it: $\text{esssup}_\mu f = \inf \{y \mid f \stackrel{\text{a.e.}}{\leq}_\mu y\}$, which translates to formal code as

Definition `ess_sup mu f := ereal_inf [set y | \forallall x \ae mu, f x <= y]`.

Now we can state and prove the universal property (the adjunction), without any side condition on the function f , which translates to formal code as follows:

```
Lemma ess_supP f y : reflect (forall x \ae mu, f x <= y) (ess_sup f <= y).
```

From this universal property we derive in a couple of lines every single useful property we require on the essential supremum, such as the following important lemma:

```
Lemma ess_supD f g : ess_sup (f + g) <= ess_sup f + ess_sup g.
```

which is a one-liner using the above mentioned universal property.

A key lemma in proving the universal property is the following [8, `measure.v`]:

```
Lemma ae_foralln (P : nat -> T -> Prop) :
  (forall n, forall x \ae mu, P n x) -> forall x \ae mu, forall n, P n x.
```

It states that the filter “almost everywhere” commutes with countable intersection, which is a specific property of this filter, and relies on the sigma-additivity of the measure.

4.2 \mathcal{L}^p and L^p spaces

The \mathcal{L}^p space ($1 \leq p \leq \infty$) is the set of measurable functions $f : T \rightarrow \mathbb{R}$ whose *Lebesgue seminorm* $N_p[f]$, defined below, is finite. We provide the definition of a seminorm in Sect. 4.3.

We start by defining the function $N_p[\cdot]$. It is essentially the p th root of the integral of the p th power of the absolute value with $p \neq \infty$, and the essential supremum (Sect. 4.1) otherwise:

$$N_p[f] \stackrel{\text{def}}{=} \begin{cases} (\int |f|^p(d\mu))^{\frac{1}{p}} & 0 < p < \infty \\ \text{esssup}_\mu |f| & p = \infty, \mu(T) > 0 \\ 0 & p = \infty, \mu(T) = 0. \end{cases}$$

In the definition above, we take μ to be the Lebesgue measure.

We formalize the norm $N_p[\cdot]$ in Rocq as a total function, i.e., we do not assume in the definition that $0 < p$ or that f is measurable, even though most interesting properties of $N_p[\cdot]$ only hold for measurable functions and $1 \leq p$:

```
Definition Lnorm {d} {T : measurableType d} {R : realType}
  (mu : {measure set T -> \bar R}) (p : \bar R) (f : T -> \bar R) :=
match p with
| p%:E => (\int[mu] x ~|f x| ^^ p) ^^ p^-1
| +oo%E => if mu [set: T] > 0 then ess_sup mu (abse \o f) else 0
| -oo%E => if mu [set: T] > 0 then ess_inf mu (abse \o f) else 0
end.
```

The notation $^^$ is for the power function and $^-1$ is for the inverse. We use the essential infimum in the case $p = -\infty$ instead of 0 for compatibility with work in progress [?] but this is not relevant in this paper. Hereafter, the notation $'N_p[f]$ denotes $N_p[f]$.

We now define the type of functions in the \mathcal{L}^p space. For that purpose, we use HIERARCHY-BUILDER [16], which provides the command `HB.mixin` to declare interfaces. The interface for functions in \mathcal{L}^p extends the structure for measurable functions. This can be observed below at line 3, `MeasurableFun` being the name of the mathematical structure of measurable functions:

```
1 HB.mixin Record isLfunction d (T : measurableType d) (R : realType)
2   (mu : {measure set T -> \bar R}) (p : \bar R) (p1 : 1 <= p) (f : T -> R)
3   of @MeasurableFun d _ T R f :=
4   { Lfunction_finite : 'N[ mu ]_p [ EFin \o f ] < +oo }.
```

Furthermore, we use the interface `isLFunction` above to define the structure of \mathcal{L}^p functions that is given the type `Lfunction mu p1` (details omitted here).

An L^p space consists of the equivalence classes of functions in \mathcal{L}^p modulo the equivalence relation $f \sim g \stackrel{\text{def}}{=} f \stackrel{\text{a.e.}}{=} g$. These classes are defined by, first, introducing a relation between measurable functions

```
Definition ae_eq_op (f g : {mfun T1 >-> T2}) := `[< f = g %[ae mu] >].
```

and, second, using the library support provided by MATHCOMP's generic quotients [15] to build the quotient of functions modulo equality almost everywhere:

```
Definition aeEqMfun mu : Type := {eq_quot ae_eq_op}.
```

with notation `{mfun_ mu, T1 >-> T2}` and, finally, using HIERARCHY-BUILDER to restrict these classes of functions to those which have a finite Lebesgue norm (assuming a measure `mu` of type `{measure set T -> \bar R}`):

```
HB.mixin Record isFinLebesgue mu (p : \bar R) (p1 : 1 <= p)
  (f : {mfun_ mu, T >-> R}) :=
  { Lebesgue_finite : 'N[ mu ]_p [ EFin \o f ] < +oo ) }.

HB.structure Definition LebesgueSpace mu (p : \bar R) (p1 : 1 <= p) :=
  { f of isFinLebesgue _ _ _ mu p p1 f }.
```

4.3 Equipping \mathcal{L}^p spaces with a seminorm

While L^p spaces are normed spaces, \mathcal{L}^p spaces are not, since, $N_p[f] = 0$ does not imply $f = 0$, but only $f = 0$ almost everywhere. Instead, \mathcal{L}^p spaces are equipped with a *seminorm*, i.e., a norm that does not need to be *positive definite*.

In this section we give the formal definition we use (Sect. 4.3.1) and provide Minkowski's inequality (Sect. 4.3.2), which is used to prove the triangle inequality.

4.3.1 Extension of the MathComp hierarchy of mathematical structures

The notion of seminorm is formalized by an interface (that we name `Zmodule_isSemiNormed`) with an operator `norm` (line 3) satisfying the triangle inequality (line 4), homogeneity (line 5), and symmetry (line 6) only:

```
1 HB.mixin Record Zmodule_isSemiNormed (R : POrderedZmodule.type) M
2   of GRing.Zmodule M := {
3     norm : M -> R;
4     ler_normD : forall x y, norm (x + y) <= norm x + norm y;
5     normrMn : forall x n, norm (x ** n) = norm x ** n;
6     normrN : forall x, norm (- x) = norm x }.
```

The notation `**` is for iterated addition. Seminorms are taken over an Abelian group (`GRing.Zmodule`), returning a value in a partially ordered Abelian group (`POrderedZmodule.type`). We can then create a mathematical structure (using the command `HB.structure`) for seminorms by combining the newly introduced `Zmodule_isSemiNormed` with the mathematical structure of Abelian groups:

```
HB.structure Definition SemiNormedZmodule (R : porderZmodType) :=
  { M of Zmodule_isSemiNormed R M & GRing.Zmodule M }.
```

In order to define norms, the condition of positive-definiteness can be provided as an interface `SemiNormedZmodule_isPositiveDefinite` extending a seminorm:

```
HB.mixin Record SemiNormedZmodule_isPositiveDefinite (R : POrderedZmodule.type) M
  of @SemiNormedZmodule R M := {
  normr0_eq0 : forall x : M, norm x = 0 -> x = 0 }.
```

The combination of the above two interfaces corresponds to norms:

```
HB.structure Definition NormedZmodule (R : porderZmodType) :=
  { M of SemiNormedZmodule_isPositiveDefinite R M & SemiNormedZmodule R M }.
```

In fact, norms were available in MATHCOMP before our work in the form of an interface `Zmodule_isNormed`. The introduction of seminorms just consisted in splitting it while preserving the original interface using HIERARCHY-BUILDER’s factories [16, Sect. 2.2]. In other words, this is a conservative extension of the hierarchy of mathematical structures of MATHCOMP.

4.3.2 Minkowski’s inequality

Minkowski’s inequality is the main lemma to show that $N_p[\cdot]$ (Sect. 4.2) is a seminorm. It says that for any two measurable functions f and g and any extended real number $p \geq 1$, we have $N_p[f + g] \leq N_p[f] + N_p[g]$. It is typically proved by first establishing Hölder’s inequality:

```
Lemma hoelder f g p q : measurable_fun [set: T] f -> measurable_fun [set: T] g ->
  0 < p -> 0 < q -> p^-1 + q^-1 = 1 ->
  'N_1[f \* g] <= 'N_p[E[f]] * 'N_q[E[g]].
```

A consequence of Hölder’s inequality which we record in our development is the *inclusion of \mathcal{L}^p -spaces* for finite measures: i.e., $\mathcal{L}^q \subseteq \mathcal{L}^p$ when $p \leq q$. Then, using `hoelder`, one proves the convexity of the p -th power function. This uses the formalization of convexity from Sect. 2.4.

```
Lemma convex_powR p : 1 <= p -> convex_function ` [0, +oo[ (fun x : R => powR x p).
```

Minkowski’s inequality is eventually proved using Hölder’s inequality and the convexity of the power function:

```
Lemma minkowski f g (p : \bar R) :
  measurable_fun [set: T] f -> measurable_fun [set: T] g -> 1 <= p ->
  'N_p[f \+ g] <= 'N_p[f] + 'N_p[g].
```

This establishes the triangle inequality of the norm $N_p[\cdot]$.

5 Probability theory with MathComp-Analysis

5.1 Basic definitions of probability theory

Basic notions of probability theory are direct applications of the Lebesgue integral. For example, we define the *expectation* $\mathbb{E}[X]$ of a random variable X as follows (notation `'E_P[X]`):

```

Definition expectation {d} {T : measurableType d} {R : realType}
  (P : probability T R) (X : T -> R) := \int[P]_w (X w)%:E.

```

Note that the expectation is an extended real number [8, probability.v]. The *covariance* between two random variables X and Y is another integral $\mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]$:

```

Context {d} {T : measurableType d} {R : realType} (P : probability T R).
Definition covariance (X Y : T -> R) :=
  'E_P[(X - cst (fine 'E_P[X])) * (Y - cst (fine 'E_P[Y]))].

```

The function `fine` is a partial inverse of `%:E`, returning `0` for infinite inputs. The *variance* $\mathbb{V}[X]$ (notation `'V_P[X]`) is a special case of the covariance:

```

(* same context as above *)
Definition variance (X : T -> R) := covariance P X X.

```

We define the distribution of the random variable $X : \{\text{mfun } T \rightarrow T'\}$ by the pushforward of the probability measure P by X (Sect. 3):

```

Definition distribution d d' (T : measurableType d) (T' : measurableType d')
  (R : realType) (P : probability T R) (X : {mfun T -> T'}) :=
  pushforward P X.

```

In order to associate explicitly a random variable with its sample space, we introduce a definition where the sample space appears as a phantom type:

```

Definition random_variable d d' (T : measurableType d) (T' : measurableType d')
  (R : realType) (P : probability T R) := {mfun T -> T'}.
Notation "{ 'RV' P -> T' }" := (@random_variable _ _ T' _ P).

```

This way, when we write $\{\text{RV } P \rightarrow T'\}$ for the type of a random variable, we understand that the underlying sample space is the one corresponding to the probability measure P .

5.2 Basic lemmas of probability theory

The first basic lemmas of probability theory such as Markov's and Chebyshev's (concentration) inequalities are proved easily using the definitions from the previous section.

Markov's inequality is a basic lemma in the sense that it is used to prove other standard concentration inequalities such as Chernoff's inequality. To state the latter, we introduce the *moment-generating function* $M_X(t) \stackrel{\text{def}}{=} \mathbb{E}[e^{tX}]$ (notation `'M_X t`):

```

Definition mmt_gen_fun (X : T -> R) t := 'E_P[expR \o t \o* X].

```

In MATHCOMP-ANALYSIS, the exponential function is `expR` and the notation `\o*` is for multiplication of a function by a constant. Chernoff's inequality (i.e., $\Pr[X \geq a] \leq M_X(r)e^{-ra}$ for r positive) is used in Sect. 6 in a concrete application:

```

Lemma chernoff (X : {RV P -> R}) (r a : R) : 0 < r ->
  P [set x | X x >= a] <= 'M_X r * (expR (- (r * a))%:E.

```

The supporting theory for the (co)variance benefits from our formalization of \mathcal{L}^p -spaces (Sect. 4). For example, consider the following lemma about the distributivity of covariance over sums of random variables:

```
Lemma covarianceDl (X Y Z : T -> R) :
  X \in Lfun P 2 -> Y \in Lfun P 2 -> Z \in Lfun P 2 ->
  covariance P (X \+ Y) Z = covariance P X Z + covariance P Y Z.
```

Here, `Lfun P 2` represents the set of functions that satisfy the `isLfun` mixin presented in Sect. 4.2, i.e., \mathcal{L}^2 . This way, we capture crucially several assumptions about the integrability of random variables used in the proofs, without loss of generality: namely, that the squares of X , Y , and Z are integrable, that X , Y , and Z are integrable by \mathcal{L}^p -space inclusion, and that the products $X * Y$, $Y * Z$, and $X * Z$ are also integrable.

Cantelli's inequality is another basic concentration inequality relating the tail of the distribution of X with its variance, and also a consequence of Markov's inequality: for any $\lambda > 0$, $\Pr[X - \mathbb{E}[X] \geq \lambda] \leq \frac{\mathbb{V}[X]}{\mathbb{V}[X] + \lambda^2}$. The formal statement makes use of `Lfun` and \mathcal{L}^p -space inclusion to express succinctly that both X and its square are integrable:

```
Lemma cantelli (X : {RV P >-> R}) (lambda : R) :
  (X : T -> R) \in Lfun P 2 -> 0 < lambda ->
  P [set x | lambda%:E <= (X x)%:E - 'E_P[X]] <= 'V_P[X] / ('V_P[X] + (lambda^2)%:E).
```

Note that the division that occurs in the left-hand side of the inequality is between extended real numbers [1, `constructive_ereal.v`].

5.3 The power measure

We need the notion of the power of a measure to talk about interesting inequalities that contain multiple random variables. Let T be a measurable space and P be a σ -finite measure on T . We define the n th power $\otimes_n P$ of P recursively on n , by iterating the binary product measure (notation $\backslash x^\wedge$ in MATHCOMP-ANALYSIS) of P and $\otimes_{n-1} P$:

```
1 Fixpoint power_measure n : set (n.-tuple T) -> \bar R :=
2   match n with
3   | 0%N => \d_[:] (* trivial Dirac measure on the singleton set {[:]})
4   | m.+1 => fun A => (P \x^\wedge power_measure m) (pair_of_tuple m @` A)
5   end.
```

The function `pair_of_tuple` at line 4 takes as input a tuple of the form $x :: xs$ and returns a pair (x, xs) . The notation $@`$ is for the image of a set. The expression `pair_of_tuple m @` A` is therefore the set $\{(x, xs) \mid x :: xs \in A\}$ of pairs of an element and a tuple of size m . Hereafter, we use the ROCQ notation $\backslash X_n P$ for `power_measure P n`.

We know that the binary product of measures $\backslash x^\wedge$ is a measure if its first component is σ -finite [2, Sect. 6.5], and we can therefore inductively prove that $\otimes_n P$ (i.e., $\backslash X_n P$) is a measure (proof `power_measureP` [8]). This is recorded by ROCQ upon the declaration of the relevant instance using the command `HB.instance` of `HIERARCHY-BUILDER`:

```
HB.instance Definition _ P n := isMeasure.Build _ _ _ (\X_n P)
  (power_measureP P n).1 (power_measureP P n).2.1 (power_measureP P n).2.2.
```

When P is a probability measure, we also prove that the measure of the whole set by $\otimes_n P$ is 1 (proof `power_measureT` [8]), hence $\otimes_n P$ is a probability measure:

```
HB.instance Definition _ P n :=
  Measure_isProbability.Build _ _ _ (\X_n P) (power_measureT P n).
```

6 Application: Formalization of a sampling theorem

The following sampling theorem can be used to estimate the percentage of a large population with a binary characteristic from a small sample with a given confidence and a given accuracy [30, Theorem 3.1] [29, Exercise 4.5].

► **Theorem 1.** *Given independent 0-1 random variables X_i with $\Pr[X_i = 1] = p$, confidence $0 < \delta \leq 1$, and accuracy $0 < \theta < p$,*

$$\text{if } n \geq \frac{3}{\theta^2} \ln\left(\frac{2}{\delta}\right) \text{ then } \Pr\left[\left|\frac{\sum_{i=1}^n X_i}{n} - p\right| \leq \theta\right] \geq 1 - \delta.$$

This theorem states that if we have a sufficiently large number of Bernoulli random variables with success probability p and if we allow a small deviation θ from the expected value p , then the probability that the average of the random variables deviates from the expected value p by more than θ is at most δ . The condition $\frac{3}{\theta^2} \ln\left(\frac{2}{\delta}\right) \leq n$ expresses the relation between the sample size n , the deviation θ , and the confidence level $1 - \delta$.

We start by formalizing the notion of a Bernoulli trial and then present the main steps of the proof, in particular the application of Chernoff's inequality (Sect. 5.2).

6.1 Bernoulli trial

We first define a Bernoulli random variable as a boolean-valued measurable function X with the property `bernoulliP` (line 3 below): the distribution (Sect. 5.1) of X in P is the Bernoulli measure `bernoulli p`, i.e., the probability measure with mass function $b \mapsto \begin{cases} p & \text{if } b \\ 1 - p & \text{otherwise} \end{cases}$:

```
1 HB.mixin Record RV_isBernoulli d (T : measurableType d) (R : realType)
2   (P : probability T R) (p : R) (X : T -> bool) of @isMeasurableFun d _ T bool X
3   := { bernoulliP : distribution P X = bernoulli p }.
```

Random variables satisfying the Bernoulli property are endowed with the type `bernoulliRV` of `BernoulliRV` structures:

```
#[short(type=bernoulliRV)]
HB.structure Definition BernoulliRV d (T : measurableType d) (R : realType)
  (P : probability T R) (p : R) := {X of @RV_isBernoulli _ _ P p X & }.
```

We now define the value of a Bernoulli trial. A Bernoulli trial is a sequence of n independent Bernoulli random variables. We represent the variables as objects of the type `n.-tuple (bernoulliRV P p)`. We first introduce a function to address the i th random variable and apply it to the i th projection of elements from the sample space (which is made of tuples of n elements—see Sect. 5.3):

```
Definition Tnth n (X : n.-tuple {mfun T -> R}) i t := (tnth X i) (tnth t i).
```

In other words, $\text{Tnth } X \ i$ is like $\text{tnth } X \ i$ but in the power sample space corresponding to the power measure $\otimes_n P$. Second, we define the value of a trial as a random variable in the power sample space:

```
Definition trial_value n (X : n.-tuple {RV P >-> R}) : {RV (\X_n P) >-> R} :=
  \sum_(i < n) Tnth X i.
```

This construction encodes the independence condition of the variables in a Bernoulli trial. Finally, we introduce a function `real_of_bool` to convert a sequence of Boolean random variables to a sequence of real random variables, giving us eventually the following definition for the value of a trial of independent Bernoulli random variables:

```
Definition bool_trial_value n := @trial_value n \o @real_of_bool n.
```

6.2 Proof of the sampling theorem

First, we prove the following bound for the value of a Bernoulli trial $X = \sum_i X_i$: $M_X(t) \leq e^{\mathbb{E}_{(\otimes_n P)}[X](e^t - 1)}$ [29, Sect. 4.2.1]. In ROCQ this is encoded by the following lemma (see Sect. 5.2 for the moment generating function):

```
Lemma mmt_gen_fun_expectation n (X_ : n.-tuple (bernoulliRV P p)) (t : R) :
  0 <= t ->
  let X := bool_trial_value X_ in
  'M_(\X_n P) X t <= expR ('E_(\X_n P) [X] * (expR t - 1)%:E).
```

This lemma relies on the independence condition encoded with the power measure $\backslash X_n P$, which allows us to express the moment generating function of the sum of the random variables as the product of the moment generating functions of the individual random variables:

$$M_X(t) = \mathbb{E}_{(\otimes_n P)}[e^{tX}] = \mathbb{E}_{(\otimes_n P)}\left[e^{t(\sum_{i=1}^n X_i)}\right] = \mathbb{E}_{(\otimes_n P)}\left[\prod_{i=1}^n e^{tX_i}\right] = \prod_{i=1}^n \mathbb{E}_P[e^{tX_i}] = \prod_{i=1}^n M_{X_i}(t).$$

Hence we prove the following lemma for the expectation of the product of random variables defined in the power measure $\backslash X_n P$ (Sect. 5.3):

```
1 Lemma expectation_power_measure_prod n (X : n.-tuple {RV P >-> R}) :
2   [set` X] `<=>` Lfun P 1 ->
3   'E_(\X_n P) [ \prod_(i < n) Tnth X i ] = \prod_(i < n) 'E_P[ tnth X i ].
```

The hypothesis at line 2 says that X regarded as a set of random variables is included in \mathcal{L}^1 .

We then have several inequalities for Bernoulli trials. The first one is [29, Theorem 4.4]:

$$P[X \geq (1 + \delta) \mathbb{E}[X]] \leq \left(\frac{e^\delta}{(1 + \delta)^{1+\delta}} \right)^{\mathbb{E}[X]}. \quad (1)$$

Put formally:

```
Theorem sampling_ineq1 n (X_ : n.-tuple (bernoulliRV P p)) delta :
  0 < delta ->
  let m := 'E_(\X_n P) [bool_trial_value X] in
  (\X_n P) [set i | X i >= (1 + delta) * fine m] <=
  ((expR delta / ((1 + delta) ^ (1 + delta))) ^ (fine m))%:E.
```

The proof uses Chernoff's bound (Sect. 5.2) and the previous lemma `mnt_gen_fun_expectation` to bound the probability of the sum of the random variables deviating from the expected value by at least $(1 + \delta) \mathbb{E}[X]$.

In a similar manner we prove, for $0 < \delta < 1$, three more inequalities that are stated informally as follows:

$$P[X \geq (1 + \delta) \mathbb{E}[X]] \leq e^{-\frac{\mathbb{E}[X] \delta^2}{3}} \quad [29, \text{Theorem 4.4}] \quad (2)$$

$$P[X \leq (1 - \delta) \mathbb{E}[X]] \leq e^{-\frac{\mathbb{E}[X] \delta^2}{2}} \quad [29, \text{Theorem 4.5}] \quad (3)$$

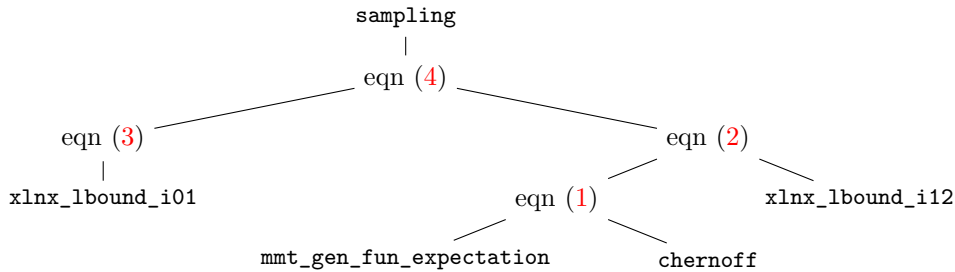
$$P[|X - \mathbb{E}[X]| \geq \delta \mathbb{E}[X]] \leq 2e^{-\frac{\mathbb{E}[X] \delta^2}{3}} \quad [29, \text{Corollary 4.6}] \quad (4)$$

See in the accompanying development `sampling_ineq2` for equation (2), `sampling_ineq3` for equation (3), `sampling_ineq4` for equation (4).

We finally state and prove the sampling theorem in ROCQ:

```
Theorem sampling n (X_ : n.-tuple (bernoulliRV P p)) theta delta :
  let X x := (bool_trial_value X x) / n%:R in
  0 < delta <= 1 -> 0 < theta < p ->
  3 / theta ^+ 2 * ln (2 / delta) <= n%:R ->
  (X_n P) [set i | `| X' i - p | <= theta] >= 1 - delta%:E.
```

Figure 1 displays the proof organization. It consists in applying the inequality (4) modulo some symbolic manipulation of the arithmetic expressions and easy measurability arguments, and then applying the other inequalities (3), (2), and (1). In the next section, we provide more insights about the analytical arguments that occur as subgoals of the intermediate inequalities.



■ **Figure 1** Proof organization of the sampling theorem

6.2.1 Proving analytical arguments using MathComp-Analysis and CoqInterval

One important aspect of the proof of the sampling theorem is the presence of analytical arguments such as the following one, which provides a lower bound of the function $x \ln(x)$:

► **Lemma 2** (`xlnx_lbound_i12`). $\forall x \in]0, 1[, x + \frac{x^2}{3} \leq (1 + x) \ln(1 + x)$

To proceed with the proof of such a concrete analysis, one has to combine several tools, such as symbolic computation of derivatives, their relation to monotonicity, properties of elementary functions such as convexity/concavity, and some automation.

The way to go about Lemma 2 is to define a function $x \mapsto (1 + x) \ln(1 + x) - (x + \frac{x^2}{3})$, compute its derivative (namely, $x \mapsto -\frac{2x}{3} + \ln(1 + x)$), study its sign (it is non-negative in

$]0, 1[$), and conclude based on monotonicity. The computation of the derivative is a matter of systematically applying standard lemmas and the conclusion is a matter of applying a lemma about monotonicity. However, studying the sign of the derivative requires a bit more work. The idea is to use the concavity of \ln to reduce the problem to the inequality $\frac{2}{3} \leq \ln(2)$ or equivalently $e^2 \leq 8$ with $e^2 = 7.389 \dots$.

To prove this inequality, one would have to expand and evaluate the defining series of e to some degree. Instead of manually doing this, we resort to the `interval` tactic provided by `CoqInterval` [27, 28]. The catch is that it requires expressions to be written using the real numbers as defined in the ROCQ standard library⁷ (`Rdefinitions.R`). In other parts of the code than here, we enjoy the MATHCOMP-ANALYSIS design of axiomatic real numbers, which allows one to work with any instance of the structure of real numbers (`Rdefinitions.R` being one) in terms of generic functions and properties. While the inequality we want to prove is of the type `Rdefinitions.R`, it is written with those generic functions. To let `CoqInterval` recognize the inequality, we have to translate them into the ROCQ standard library functions. For that purpose, MATHCOMP-ANALYSIS provides a compatibility module (in the form of the file `Rstruct.v`). As a result, the goal can be proved in a few lines:

```
1  Lemma exp2_le8 : exp 2 <= 8. Proof. interval. Qed.
2
3  Lemma exp2_le8_conversion : reflect (exp 2 <= 8) (expR 2 <= 8).
4  Proof.
5  rewrite RexpE (_ : 8%R = 8); last first.
6    by rewrite !mulrS -!RplusE Rplus_0_r !RplusA !IZRposE/=.
7  by apply: (iffP idP) => /RleP.
8  Qed.
```

At line 1, we prove the desired inequality with the ROCQ standard library automatically using `interval`. The lemma `exp2_le8_conversion` shows that it is the same as the corresponding inequality in MATHCOMP-ANALYSIS using `expR`. The lemma `RexpE` (line 5) equates the different definitions of the exponential function from the ROCQ standard library and MATHCOMP-ANALYSIS by observing their defining power series are equal modulo the names of ring operations. Line 6 proves that the notations `8` in both libraries are the same. This is done using, among others, the lemma `RplusE` [1, `Rstruct.v`] to convert between the additions in both libraries.

The last step of the proof of inequality (3) features another lower bound argument on $x \ln(x)$:

► **Lemma 3** (`xlnx_lbound_i01`). $\forall x \in]0, 1[, x^2 - 1 < 2x \ln(x)$

The proof proceeds similarly to the previous Lemma 2. We first apply the monotonicity lemma for functions with positive derivative to reduce the goal to an evaluation of the derivative, then simplify it to a basic property of the exponential function: the inequality $1 + x \leq e^x$ that is used pervasively in mathematical analysis.

⁷ For historical reasons, ROCQ's standard library offers a theory of real numbers based on a monolithic type, whereas MATHCOMP-ANALYSIS bases it on a full algebraic hierarchy, hence the two different formalizations of real numbers. The `CoqInterval` library had been developped using the former type.

7 Related work

Verification of static-analysis tools based on abstract interpretation with a proof assistant is not new [?], nor is verified interval arithmetic [12, 28]. The novelty of the work offered in Sect. 2 rather lies in a lightweight semi-decision procedure that can be used smoothly during interactive development of proofs, to address small subgoals. Another difference with CoqInterval [28] is that the latter performs floating-point computations, coming with both higher accuracy and rounding errors, making the distinction between open and closed intervals essentially useless, contrary to our integer intervals. Both tools are complementary, as seen in Sect. 6.2.1.

We already mentioned in Sect. 1 that there are already several accounts of probability theory in the ROCQ proof assistant. INFOtheo [21] is a library for probabilities supported by a finite set that has been applied to information theory, error-correcting codes and robust statistics [?, ?, 5, 7]. The coq-proba library [33] provides discrete probability spaces with various standard theorems (including Chernoff’s inequality) and also applications to program verification (e.g., [20]). The FormalML library [36] contains advanced theorems on probability theory with applications to program synthesis [38]. It is a priori not limited to discrete probability spaces and contains a formalization of L^p spaces [?, Sect. 3.4] but restricted to finite p and probability spaces. Our work is not duplicating efforts in the sense that we provide more general definitions. Contrary to FormalML, we can rely on a formalization of the Lebesgue integral [2]. Also, for example, we can show that probability distributions in INFOtheo are instances of MATHCOMP-ANALYSIS’s definitions, opening the door to sharing of results.

Since we have just started formalizing probability theory, we do not provide as many definitions and lemmas as Isabelle/HOL and its Archive of Formal Proofs⁸. Indeed, Isabelle/HOL has long been providing advanced material about probability theory: the theory of \mathcal{L}^p spaces [19], the formalization of the Central Limit Theorem [9], various concentration inequalities (including Cantelli’s) [?], etc. There are however a few aspects on which our work improves. In [19], spaces of functions are not represented by a type but as a subset of functions. This is partly because L^p spaces are difficult to formalize in the absence of dependent types since such a type is naturally parameterized by p and a measure space [19, Sect. 2]. We define conjugate exponents using a notion of inverse for extended real numbers (already mentioned in Sect. 5.2) that seems to lend itself well to the formalization of more properties of conjugate exponents (see [19, Sect. 3] and [8, hoelder.v]). The inclusion of \mathcal{L}^p spaces is proved for probability spaces in [19, Sect. 5] whereas we prove it for finite measures. Yet, Isabelle/HOL provides \mathcal{L}^p spaces for $0 \leq p < 1$ [19, Sections 5.2 and 5.4], Minkowski’s inequality for $0 < p \leq 1$ [19, Sect. 4], quasi-norms and topological results about them [19, Sect. 2.7], and more lemmas (e.g., about the multiplication of function in \mathcal{L}^p spaces [19, Sect. 5.8]).

Mathlib [37] also provides L^p spaces [26] as a subtype of the space `AEEqFun` of equivalence classes of almost everywhere (strongly) measurable functions. According to Sonoda et al. [?, Sect. 7.2], several concentration inequalities (Markov, Chebyshev, Azuma-Hoeffding) can be found in mathlib but Isabelle/HOL still offers more lemmas. Regarding the theory of the essential supremum in mathlib, it is derived as an instance of a more general theory of countable intersection filters on first countable topologies. It provides the same properties as we do, and like us, the essential supremum is not directly stated as an adjunction.

Rather than advancing the plethora of lemmas about \mathcal{L}^p spaces, our work shows that

⁸ <https://www.isa-afp.org/>

Contents	Relevant file (and reference)	Section in this paper
interval inference	<code>interval_inference.v</code> [?]	Section 2
essential supremum/infimum	<code>ess_sup_inf.v</code> [1]	Section 4.1
\mathcal{L}^p , L^p spaces	<code>hoelder.v</code> [1]	Sections 4.2 and 4.3
seminorms	<code>numdomain.v</code> [?]	Section 4.3.1
basic probability theory	<code>probability.v</code> [1]	Sections 5.1 and 5.2
power measure, sampling theorem	<code>sampling.v</code> [8]	Sections 5.3 and 6
compatibility between reals	<code>Rstruct.v</code> , <code>Rstruct_topology.v</code> [1]	Section 6.2.1

■ **Figure 2** Overview of the formalization explained in this paper

ROCQ’s original features can be put in good use: ROCQ’s canonical structures are used in Sect. 2 and ROCQ libraries providing automatic tactics for arithmetic and numeric evaluation are used in Sect. 6. All of these fundamentally rely on the efficient computation capabilities of ROCQ, enabling implementation of reflexive tactics. The use of HIERARCHY-BUILDER to formalize L^p spaces in Sect. 4 also pertains to the more general topic of formalization of hierarchies of mathematical structures [10, 16].

8 Conclusions

In this paper, we proposed a new formalization of probability theory developed so as to take advantage of ROCQ and MATHCOMP specific features. In particular, we benefited from ROCQ’s type system to automate range inference of values and we used HIERARCHY-BUILDER to make the hierarchy of mathematical structures of MATHCOMP evolve to cover \mathcal{L}^p spaces. We also saw that the availability of a theory for \mathcal{L}^p allows for succinct statements of basic lemmas of probability theory such as standard concentration inequalities. Using our development, we formalized a sampling theorem which is itself a concentration inequality and observed in particular that the formalization of analytical arguments benefited from the automation available in ROCQ. Figure 2 provides an overview of the formalization.

Future work

Now that we have laid down the foundations of a new formalization of probability theory, we are planning to port existing results from related ROCQ libraries, e.g., conical spaces [4, Sect. 4].

The sampling theorem we proved is an example of concentration inequality for statistical procedures used to effectively reduce the sample size to achieve a certain precision in a statistical estimation. We aim at further generalizing our results towards the formalization of complex procedures such as risk-limiting audits [31], i.e., statistical procedures used after elections to ascertain that a potential error in counting the ballots does not alter the final election result. We also plan to apply our formalization of \mathcal{L}^p spaces to give semantics to logics used in machine learning as devised by Capucci [?, 14].

References

- 1 Reynald Affeldt, Clark W. Barrett, Alessandro Bruni, Ieva Daukantas, Harun Khan, Takafumi Saikawa, and Carsten Schürmann. Robust mean estimation by all means (short paper). In *15th International Conference on Interactive Theorem Proving (ITP 2024), September 9–14, 2024, Tbilisi, Georgia*, volume 309 of *LIPICs*, pages 39:1–39:8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICs.ITP.2024.39.

- 2 Reynald Affeldt, Yves Bertot, Alessandro Bruni, Cyril Cohen, Marie Kerjean, Assia Mahboubi, Damien Rouhling, Pierre Roux, Kazuhiko Sakaguchi, Zachary Stone, Pierre-Yves Strub, and Laurent Théry. MathComp-Analysis: Mathematical components compliant analysis library. <https://github.com/math-comp/analysis>, Jul 2025. Since 2017. Latest stable version: 1.12.0.
- 3 Reynald Affeldt, Alessandro Bruni, Cyril Cohen, Pierre Roux, and Takafumi Saikawa. Formalizing concentration inequalities in Rocq: infrastructure and automation. <https://github.com/math-comp/analysis/pull/1645>, Jul 2025. Development corresponding to Sections 5.3 and 6. Pull request to [1].
- 4 Reynald Affeldt and Cyril Cohen. Measure construction by extension in dependent type theory with application to integration. *J. Autom. Reason.*, 67(3):28, 2023. doi:10.1007/s10817-023-09671-5.
- 5 Reynald Affeldt, Cyril Cohen, and Damien Rouhling. Formalization techniques for asymptotic reasoning in classical analysis. *J. Formaliz. Reason.*, 11(1):43–76, 2018. doi:10.6092/issn.1972-5787/8124.
- 6 Reynald Affeldt, Jacques Garrigue, and Takafumi Saikawa. Formal adventures in convex and conical spaces. In *13th Conference on Intelligent Computer Mathematics (CICM 2020), Bertinoro, Forlì, Italy, July 26–31, 2020*, volume 12236 of *Lecture Notes in Artificial Intelligence*, pages 23–38. Springer, Jul 2020. doi:10.1007/978-3-030-53518-6_2.
- 7 Reynald Affeldt, Jacques Garrigue, and Takafumi Saikawa. A library for formalization of linear error-correcting codes. *J. Autom. Reason.*, 64(6):1123–1164, 2020. doi:10.1007/s10817-019-09538-8.
- 8 Reynald Affeldt, Jacques Garrigue, and Takafumi Saikawa. Reasoning with conditional probabilities and joint distributions in Coq. *Computer Software*, 37(3):79–95, 2020. Japan Society for Software Science and Technology. doi:10.11309/jssst.37.3_79.
- 9 Reynald Affeldt, Manabu Hagiwara, and Jonas Sénizergues. Formalization of Shannon’s theorems. *J. Autom. Reason.*, 53(1):63–103, 2014.
- 10 Reynald Affeldt and Zachary Stone. A comprehensive overview of the Lebesgue differentiation theorem in Coq. In *15th International Conference on Interactive Theorem Proving (ITP 2024), September 9–14, 2024, Tbilisi, Georgia*, volume 309 of *LIPICs*, pages 5:1–5:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICs.ITP.2024.5.
- 11 Jeremy Avigad, Johannes Hölzl, and Luke Serafin. A formally verified proof of the central limit theorem. *J. Autom. Reason.*, 59(4):389–423, 2017. doi:10.1007/s10817-017-9404-x.
- 12 Anne Baanen. Use and abuse of instance parameters in the Lean mathematical library. In *13th International Conference on Interactive Theorem Proving (ITP 2022), August 7–10, 2022, Haifa, Israel*, volume 237 of *LIPICs*, pages 4:1–4:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ITP.2022.4.
- 13 Frédéric Besson, David Cachera, Thomas P. Jensen, and David Pichardie. Certified static analysis by abstract interpretation. In *Foundations of Security Analysis and Design V (FOSAD 2007/2008/2009), Tutorial Lectures*, volume 5705 of *Lecture Notes in Computer Science*, pages 223–257. Springer, 2009. doi:10.1007/978-3-642-03829-7_8.
- 14 Sylvie Boldo, Catherine Lelay, and Guillaume Melquiond. Coquelicot: A user-friendly library of real analysis for Coq. *Math. Comput. Sci.*, 9(1):41–62, 2015. doi:10.1007/S11786-014-0181-1.
- 15 Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration Inequalities: A Nonasymptotic Theory of Independence*. Oxford University Press, 2013.
- 16 Matteo Capucci. On quantifiers for quantitative reasoning, 2025. URL: <https://arxiv.org/abs/2406.04936>, arXiv:2406.04936.

- 17 Cyril Cohen. Pragmatic quotient types in Coq. In *4th International Conference on Interactive Theorem Proving (ITP 2013), Rennes, France, July 22–26, 2013*, volume 7998 of *Lecture Notes in Computer Science*, pages 213–228. Springer, 2013. doi:[10.1007/978-3-642-39634-2_17](https://doi.org/10.1007/978-3-642-39634-2_17).
- 18 Cyril Cohen, Kazuhiko Sakaguchi, and Enrico Tassi. Hierarchy builder: Algebraic hierarchies made easy in Coq with Elpi (system description). In *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020), June 29–July 6, 2020, Paris, France (Virtual Conference)*, volume 167 of *LIPICs*, pages 34:1–34:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:[10.4230/LIPICs.FSCD.2020.34](https://doi.org/10.4230/LIPICs.FSCD.2020.34).
- 19 Patrick Cousot and Radhia Cousot. Abstract interpretation frameworks. *Journal of logic and computation*, 2(4):511–547, 1992.
- 20 Ieva Daukantas, Alessandro Bruni, and Carsten Schürmann. Trimming data sets: a verified algorithm for robust mean estimation. In *23rd International Symposium on Principles and Practice of Declarative Programming (PPDP 2021), Tallinn, Estonia, September 6–8, 2021*, pages 17:1–17:9. ACM, 2021. doi:[10.1145/3479394.3479412](https://doi.org/10.1145/3479394.3479412).
- 21 The MathComp development team. Mathematical components. <https://github.com/math-comp/math-comp>, 2005. Last stable version: 2.4.0 (2025).
- 22 Jean Dieudonné. *Treatise On Analysis*, volume II. Academic Press, 1976.
- 23 Sebastien Gouezel. Lp spaces. *Archive of Formal Proofs*, October 2016. <https://isa-afp.org/entries/Lp.html>, Formal proof development.
- 24 Simon Oddershede Gregersen, Alejandro Aguirre, Philipp G. Haselwarter, Joseph Tassarotti, and Lars Birkedal. Asynchronous probabilistic couplings in higher-order separation logic. *Proc. ACM Program. Lang.*, 8(POPL):753–784, 2024. doi:[10.1145/3632868](https://doi.org/10.1145/3632868).
- 25 InfoTheo. InfoTheo: A Coq formalization of information theory and linear error-correcting codes. <https://github.com/affeldt-aist/infotheo>, 2025. Authors: Reynald Affeldt, Manabu Hagiwara, Jonas Sénizergues, Jacques Garrigue, Kazuhiko Sakaguchi, Taku Asai, Takafumi Saikawa, Naruomi Obata, and Alessandro Bruni. Last stable release: 0.9.3 (2025).
- 26 Yoshihiro Ishiguro and Reynald Affeldt. The Radon-Nikodým theorem and the Lebesgue-Stieltjes measure in Coq. *Computer Software*, 41(2):41–59, 2024. Japan Society for Software Science and Technology. doi:[10.11309/jssst.41.2_41](https://doi.org/10.11309/jssst.41.2_41).
- 27 Emin Karayel and Yong Kiam Tan. Concentration inequalities. *Archive of Formal Proofs*, November 2023. https://isa-afp.org/entries/Concentration_Inequalities.html, Formal proof development.
- 28 Assia Mahboubi and Enrico Tassi. *Mathematical Components*. Zenodo, Jan 2021. doi:[10.5281/zenodo.4457887](https://doi.org/10.5281/zenodo.4457887).
- 29 Filip Markovic, Pierre Roux, Sergey Bozhko, Alessandro V. Papadopoulos, and Björn B. Brandenburg. CTA: A correlation-tolerant analysis of the deadline-failure probability of dependent tasks. In *IEEE Real-Time Systems Symposium (RTSS 2023), Taipei, Taiwan, December 5–8, 2023*, pages 317–330. IEEE, 2023. doi:[10.1109/RTSS59052.2023.00035](https://doi.org/10.1109/RTSS59052.2023.00035).
- 30 Jairo Miguel Marulanda-Giraldo, Ekaterina Komendantskaya, Alessandro Bruni, Reynald Affeldt, Matteo Capucci, and Enrico Marchioni. Quantifiers for differentiable logics in Rocq (extended abstract). In *8th International Symposium on AI Verification (SAIV 2025), Zagreb, Croatia, July 21–22, 2025*, Jul 2025.
- 31 Mathlib 4. File `MeasureTheory/Function/LpSpace/Basic.lean`. [url](https://mathlib.org/), Jun 2025.
- 32 Guillaume Melquiond. Proving bounds on real-valued functions with computations. In *4th International Joint Conference on Automated Reasoning (IJCAR 2008), Sydney, Australia, August 12–15, 2008*, volume 5195 of *Lecture Notes in Computer Science*, pages 2–17. Springer, 2008. doi:[10.1007/978-3-540-71070-7_2](https://doi.org/10.1007/978-3-540-71070-7_2).

- 33 Guillaume Melquiond, Érik Martin-Dorel, Pierre Roux, and Thomas Sibut-Pinote. CoqInterval. <https://coqinterval.gitlabpages.inria.fr/>, 2025. Since 2008. Version 4.11.1.
- 34 Michael Mitzenmacher and Eli Upfal. *Probability and Computing—Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.
- 35 Samir Rajani. Applications of Chernoff bounds. <http://math.uchicago.edu/~may/REU2019/REUPapers/Rajani.pdf>, 2019. The University of Chicago Mathematics REU 2019.
- 36 Sho Sonoda, Kazumi Kasaura, Yuma Mizuno, Kei Tsukamoto, and Naoto Onda. Lean formalization of generalization error bound by Rademacher complexity, 2025. URL: <https://arxiv.org/abs/2503.19605>, [arXiv:2503.19605](https://arxiv.org/abs/2503.19605).
- 37 Philip B. Stark. Risk-limiting postelection audits: conservative P-values from common probability inequalities. *IEEE Trans. Inf. Forensics Secur.*, 4(4):1005–1014, 2009. doi:[10.1109/TIFS.2009.2034190](https://doi.org/10.1109/TIFS.2009.2034190).
- 38 M. H. Stone. Postulates for the barycentric calculus. *Annali di Matematica Pura ed Applicata*, 29:25–30, 1949.
- 39 Joseph Tassarotti. A probability theory library for the Coq theorem prover. <https://github.com/jtassarotti/coq-proba>, 2023. Since 2020.
- 40 The Coq Development Team. Automatic solvers and programmable tactics, 2024. Chapter in [35].
- 41 The Coq Development Team. The SSReflect proof language, 2024. Chapter in [35].
- 42 The Coq Development Team. The Coq reference manual. <https://rocq-prover.org/doc/V8.20.1/refman/index.html>, 2024. Release 8.20.1.
- 43 The FormalML development team. FormalML: Formalization of machine learning theory with applications to program synthesis. <https://github.com/IBM/FormalML>, 2025. Since 2019.
- 44 The mathlib Community. The lean mathematical library. In *9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020), New Orleans, LA, USA, January 20–21, 2020*, pages 367–381. ACM, 2020. doi:[10.1145/3372885.3373824](https://doi.org/10.1145/3372885.3373824).
- 45 Koundinya Vajjha, Avraham Shinnar, Barry M. Trager, Vasily Pestun, and Nathan Fulton. CertRL: formalizing convergence proofs for value and policy iteration in Coq. In *10th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2021), Virtual Event, Denmark, January 17–19, 2021*, pages 18–31. ACM, 2021. doi:[10.1145/3437992.3439927](https://doi.org/10.1145/3437992.3439927).
- 46 Koundinya Vajjha, Barry M. Trager, Avraham Shinnar, and Vasily Pestun. Formalization of a stochastic approximation theorem. In *13th International Conference on Interactive Theorem Proving (ITP 2022), August 7–10, 2022, Haifa, Israel*, volume 237 of *LIPICs*, pages 31:1–31:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:[10.4230/LIPICS.ITP.2022.31](https://doi.org/10.4230/LIPICS.ITP.2022.31).
- 47 Rosalind Cecily Young. The algebra of many-valued quantities. *Mathematische Annalen*, 104(1):260–290, 1931.