特集●「プログラミングおよびプログラミング言語」特集

The Radon-Nikodým Theorem and the Lebesgue-Stieltjes Measure in Coq

Yoshihiro Ishiguro Reynald Affeldt

We are concerned with the formalization of measure theory in the Coq proof assistant. Concretely, we extend MathComp-Analysis, a library for functional analysis built on top of the Mathematical Components library, with standard constructions such as charges and the Lebesgue-Stieltjes measure, and with standard theorems such as the Hahn decomposition theorem and the Radon-Nikodým theorem. These are prerequisites for the formalization of probabilistic programs, of probability theory, and also for other applications such as the formalization of connections between derivatives and integrals.

1 Introduction

In this paper, we are concerned with the formalization of measure theory in the Coq proof assistant [36]. Our goal is to extend MathComp-Analysis [2], an on-going effort to formalize functional analysis in Coq, with theorems useful to deal with probabilistic programs and to formalize mathematics in general, as illustrated by the following two motivating examples.

1.1 Motivation 1: Formal Verification of Probabilistic Programs

Probabilistic programs provide a way to deal with uncertainty in a rigorous way. They have applications in artificial intelligence (e.g., formal verification of machine learning [8]), information security (e.g., formal verification of cryptographic proofs [23]), etc. Their semantics relies on probability theory and more generally on measure theory [18].

The Coq proof assistant has arguably been an important tool for the formal verification of programs. It has been awarded the ACM SIGPLAN Programming Languages Sofware Award and the ACM Software System Award in 2013; the ACM Software System Award of 2021 has also been awarded to a C compiler developed in Coq [29]. Unfortunately, it has been lacking libraries for measure theory and this has been detrimental to the development of formal semantics for probabilistic programs. For example, in the formalization of [40], most of measure and integration theory is added in the form of unproved axioms.

Using MathComp-Analysis (in particular [3]), it has however been possible to formalize without axioms the semantics of a first-order probabilistic programming language proposed by Staton [35]. This formalization [6, 34] highlights, among others, the importance of density functions or Radon-Nikodým derivatives. See also [13, Sect. 2.3] that also stresses the importance of Radon-Nikodým derivatives.

1.2 Motivation 2: Formalization of Connections between Derivatives and In-

Coq による Radon-Nikodým の定理と Lebesgue-Stieltjes Measure の形式化

Yoshihiro Ishiguro, 名古屋大学大学院多元数理科学研究科, Graduate School of Mathematics, Nagoya University.

Reynald Affeldt, 国立研究開発法人産業技術総合研究所, Digital Architecture Research Center, National Institute of Advanced Industrial Science and Technology (AIST) .

コンピュータソフトウェア, Vol.0, No.0 (0), pp.0–0.

[研究論文] 2023 年 06 月 16 日受付.

本論文は第 25 回プログラミングおよびプログラミング言語ワークショップ (PPL2023) の発表論文 [26] を改訂したものである.

tegrals

Derivatives and integrals are the main topic of analysis. The connections between them take the form of fundamental theorems whose formalization relies on measure theory.

This is for example the case of the Fundamental Theorem of Calculus for the Lebesgue integral [33, Thm 7.18]. It says that for any function f that is *absolutely continuous* †1 on $[a, b]$, there exists almost-everywhere a function f' integrable on $[a, b]$ such that

$$\forall x \in [a, b], \quad f(x) - f(a) = \int_{t=a}^{x} f'(t)(\mathbf{d}\,\Lambda)$$

where Λ is the Lebesgue measure. Conversely, for any function f that is integrable on $[a, b]$, there exists an absolutely continuous function F such that the derivative of F at x is equal almost-everywhere to $f(x)$ for x in $[a, b]$.

In a nutshell, the proof for a non-decreasing function f goes as follows. Let Λ_f be the Lebesgue-Stieltjes measure associated with f; f is absolutely continuous on $[a, b]$ if and only if Λ_f is dominated by the Lebesgue measure. By applying the Radon-Nikodým theorem to Λ_f, we obtain the Radon-Nikodým derivative f^{RN} of the Lebesgue-Stieltjes measure associated with f which satisfies

$$\Lambda_f(E) = \int_{t \in E} f^{RN}(t)(\mathbf{d}\,\Lambda)$$

for all measurable sets E. If E is an interval $[a, x]$ for any x in $[a, b]$, then $\Lambda_f[a, x]$ evaluates to $f(x) - f(a)$ by definition. It remains to show that the Radon-Nikodým derivative is the standard derivative (see [9]).

1.3 Our Contribution in This Paper

We use the two motivating examples above to set our objectives for the extension of the formalization of measure theory of MathComp-Analysis. Concretely, we provide formalizations of the Radon-Nikodým theorem and of the Lebesgue-Stieltjes measure in the Coq proof assistant. The Radon-Nikodým theorem establishes the existence of Radon-Nikodým derivatives or density functions; it uses as an intermediate step the Hahn decomposition theorem. The Lebesgue-Stieltjes measure is a generalization of the Lebesgue measure used in particular in probability theory [28, Example 1.58]. To the best of our knowledge, these are the first formalizations of these results in Coq and some aspects also compare favorably with related work (see Sect. 6 for details).

Besides these contributions to Coq, this paper also has another objective: to document the process of formalization of measure theory with MathComp-Analysis. Indeed, measure theory is vast (it includes integration and probability theory), its formalization is therefore not an easy task and ought better be a collaborative effort, which requires documentation. For this purpose, we focus on the main technical parts of the pencil-and-paper proofs and explain how we turn them into proof scripts. We emphasize the reusable components of MathComp-Analysis, such as the charges used in the Radon-Nikodým theorem and the Hahn decomposition theorem, or a generic result from previous work that we use to build the Lebesgue-Stieltjes measure. The key technology to organize charges and measures in a hierarchy of mathematical structures that favors reusability is Hierarchy-Builder [15], a generic tool for the Coq proof assistant.

1.4 Paper Outline

This paper is organized as follows. Section 2 summarizes mathematical preliminaries and provides background information about the formalization of measure theory in MathComp-Analysis. Section 3 explains the formalization of charges and of the Hahn decomposition theorem, an intermediate step to prove the Radon-Nikodým theorem, which is the topic of Sect. 4. Section 5 uses previous work [3] to build the Lebesgue-Stieltjes measure. We review related work in Sect. 6 and conclude in Sect. 7 with more insights about the formalization of the Fundamental Theorem of Calculus.

2 Background about the Formalization of Measure Theory

We recall the basics of measure and integration theory in Sect. 2.1 and we provide an overview of how it is formalized in MathComp-Analysis in Sect. 2.2. More details about MathComp-Analysis can be found in the following papers [3–5]

†1 A function $f : \mathbb{R} \to \mathbb{R}$ is *absolutely continuous* on interval $[a, b]$ when f satisfies the following condition. For all $\epsilon > 0$, there exists $\delta > 0$ such that for all $n \in \mathbb{N}$ and for every collection of pairwise disjoint intervals $]a_k, b_k[\subset [a, b]$ with $\sum_{k=1}^{n} |b_k - a_k| < \delta$, we have $\sum_{k=1}^{n} |f(b_k) - f(a_k)| < \epsilon$.

or in teaching material [1].

2.1 Mathematical Preliminaries

A σ-algebra on a set T is a collection of subsets of T that contains $\emptyset$ and that is closed under complement and countable union. We note Σ_T for such a σ-algebra and call the sets in Σ_T *measurable sets*. The standard σ-algebra on $\mathbb{R}$ (the Borel sets) is the smallest σ-algebra containing the intervals. A *semiring of sets* on a set T is a collection of subsets of T that contains $\emptyset$, that is closed under finite intersection, and that is "closed under finite difference", i.e., such that for all sets A and B in the semiring, there exists a set D of pairwise-disjoint sets in the semiring such that $A \setminus B = \bigcup_{x \in D} x$. A σ-algebra is a semiring of sets.

A *measure* is a non-negative function $\mu : \Sigma_X \to [0, \infty]$ such that $\mu(\emptyset) = 0$ and $\mu(\bigcup_i A_i) = \sum_i \mu(A_i)$ for pairwise-disjoint measurable sets A_i where the sum is countable. The latter property is called *σ-additivity*. When the sum is finite, this property is called *additivity* and the measure is said to be a *content*. A function μ is *σ-subadditive* when for any measurable set A and any countable family of measurable sets F, $A \subseteq \bigcup_k F_k$ implies $\mu(A) \leq \sum_k \mu(F_k)$. A non-negative, monotone, σ-subadditive function μ such that $\mu(\emptyset) = 0$ is called an *outer measure*. The standard measure on $\mathbb{R}$ is the Lebesgue measure, which is such that the length of an interval $]a, b]$ is $b - a$. A measure μ is *finite* when the measure of the full set is not $+\infty$. A measure μ is *σ-finite* over A when there is a countable family of measurable sets F such that $A = \bigcup_i F_i$ and $\mu(F_i) < \infty$ for all i.

A function $f : X \to Y$ is *measurable* when for all measurable sets $B \in \Sigma_Y$, $f^{-1}(B)$ is also measurable. We can integrate a measurable function f w.r.t. a measure μ over D to get an extended real number denoted by $\int_{x \in D} f(x)(\mathbf{d}\,\mu)$. When D is the full set, we write $\int_x f(x)(\mathbf{d}\,\mu)$. When a function f is measurable and $\int_{x \in D} |f(x)|(\mathbf{d}\,\mu) < +\infty$, we say that f is *integrable* over D. Integration satisfies the monotone convergence theorem, i.e., $\int_{x \in D} f\,x(\mathbf{d}\,\mu) = \lim_n \int_{x \in D} f_n\,x(\mathbf{d}\,\mu)$ where f_n is an increasing sequence of non-negative measurable functions converging towards f, a non-negative measurable function.

2.2 Measure and Integration Theory in MathComp-Analysis

As already explained in Sect. 1, MathComp-Analysis is an on-going effort to formalize functional analysis in Coq. It extends the Mathematical Components library [30] (hereafter, MathComp) which is a constructive library consisting of several algebraic theories that made it possible to formalize the Odd Order theorem [22, Sect. 6]. MathComp-Analysis extends MathComp with classical axioms [5, Sect. 5].

2.2.1 Basic Notations from MathComp

We make use of standard MathComp notations. The notation `f ^~ y` is for the function $\lambda x.f\,x\,y$. The notation `\o` is for function composition. The projections of a pair are denoted by `.1` and `.2`. The notation `[/\ P0, P1, ... & Pn]` (the first $n-1$ separators are commas (`,`) while the last one is an ampersand (`&`)) is for the iterated conjunction `P0 /\ P1 /\ ... /\ Pn`. This notation comes with constructors when there is a small number of conjuncts, for example `And3` is used to build a conjunction of three propositions. The notation `n%:R` is for injecting the natural number `n` into a ring type; `%R` is a delimiter for the notation scope of rings. Finitely iterated operators are noted `\big[op/idx]_(k < n | P k) f k` where `f` is for the terms, `P` an optional filtering boolean predicate, `op` a binary operation, and `idx` its neutral [11]. In the case of additions, there is a specialized notation `\sum_(k < n | P k) f k`.

2.2.2 Basic Notations from MathComp-Analysis

The type `set T` is for sets of objects of type `T`; we therefore write `S : set T` when `S` is a subset of `T` seen as a full set. The full set of objects of type `T` is denoted by `[set: T]`; it is a notation for `setT`, which can be used instead when the inference of the type is automatic. Set inclusion is denoted by `` `<=` ``, set union by `` `|` `` (identifier `setU`), set intersection by `` `&` `` (identifier `setI`), set difference by `` `\` `` (identifier `setD`), set complement by `~~` (identifier `setC`), the preimage of the set `A` by `f` is denoted by `f @^-1` A`, the identifier corresponding to the empty set is `set0`. Sets can be defined by comprehension using the notation `[set x | P]`, for the set of objects `x` such that `P` holds, or the notation `[set E | x in A]`, where `E` is an expression containing `x` and `A` is a set. When the sets of a family `F` indexed by `D` are pairwise-disjoint, we write

`trivIset D F`. We can write `A a` (in `Prop`) or `a \in A` (in `bool`) to state that `a` belongs to the set `A`.

The convergence of a sequence `u` towards `l` is denoted by `u --> l`, we can also write `lim u = l`, as explained in [5, Sect. 2.3]. The type of a sequence of objects of type `A` indexed by natural numbers is denoted by `A^nat`. The type `{posnum R}` is for positive numeric types, where `R` is a numeric type among `numDomainType` for integral domains with an order, `numFieldType` for numeric fields, or `realType` for real numbers. Given `e : {posnum R}`, `e%:num` is the projection of type `R`. The type `\bar R` is for extended real numbers. In particular, when `R` is `realType`, `\bar R` corresponds to $\overline{\mathbb{R}} = \mathbb{R} \cup \{+\infty, -\infty\}$. Infinite values are denoted by the notations `-oo` and `+oo` and `r%:E` represents the injection of `r : R` into `\bar R`; the notations about extended real numbers lie in the scope `%E`. An extended real number `x` which is not $+\infty$ or $-\infty$ satisfies the boolean (postfix) predicate `\is a fin_num`, i.e., `x \is a fin_num` means that the extended real number `x` is actually a real number[†2]. The supremum of a set `E` of extended real numbers is `ereal_sup E`. The maximum of two real numbers `x` and `y` is `maxe x y`.

The type of intervals over a numeric type `R` is `interval R`. Closed intervals are denoted by the notation `` `[a, b] ``, open intervals by `` `]a, b[ ``, open-closed intervals by `` `]a, b] ``, etc. When an interval `i` is an interval of extended real numbers, we can write `i.1` and `i.2` for its endpoints.

2.2.3 Measure Theory in MathComp-Analysis

The type of σ-algebra is `measurableType d`. Given `T` of type `measurableType d` and `U` of type `set T`, `measurable U` asserts that `U` belongs to the σ-algebra corresponding to `T`. In other words, we write `T : measurableType d` for a measurable space (T, Σ_T), and `measurable S` when `S` belongs to Σ_T. The parameter `d` controls the display of the `measurable` predicate, so that `measurable U` is printed as `d.-measurable U`. This is useful to disambiguate the local context of a proof in the presence of several σ-algebras but this parameter can be ignored on a first reading; see [3, Sect. 3.4] for more details about this display mechanism. The predicate `semi_setD_closed` is the formalization of the property of being "closed under finite difference" (Sect. 2.1). Semiring of sets are available as the type `semiRingOfSetsType d`.

Given `T` of type `measurableType d`, the type of (non-negative) measures on `T` is denoted by `{measure set T -> \bar R}` where `R` has type `realType` [3, Sect. 3.5.2]. Similarly, the type of contents is denoted by `{content set T -> \bar R}` [3, Sect. 3.5.2] and the type of σ-finite measures is denoted by `{sigma_finite_measure set T -> \bar R}` [3, Sect. 4.3].

We write `measurable_fun D f` for a measurable function `f` with domain `D`. When a function `f` is integrable over `D` w.r.t. `mu`, we write `mu.-integrable D f`. The notation for the integral $\int_{x \in D} f(x)(\mathbf{d}\,\mu)$ is `\int[mu]_(x in D) f x`.

3 The Hahn Decomposition Theorem in Coq

The Hahn decomposition theorem is a standard result of measure theory. It is used for example to prove the Radon-Nikodým theorem and can be found in many lecture notes (we used the following online resource [19] but the same proof can be found elsewhere). Before explaining the Hahn decomposition theorem in Sect. 3.2, we need to define charges in Sect. 3.1. We take this occasion to introduce the Hierarchy-Builder tool [15] to build hierarchies of mathematical structures.

3.1 Formalization of Charges and Introduction to Hierarchy-Builder

Let Σ_T be a σ-algebra of subsets of T. A *charge* (a.k.a. signed measure) is a σ-additive function ν that maps measurable sets to *real numbers* [27, Sect. 7.1.1][†3]. MathComp-Analysis already provides measures and therefore also provides formal definitions for additivity and σ-additivity. Since a measure is potentially infinite, these formal definitions are for extended real numbers. In order to reuse these definitions to define charges, we define an interface for extended real-valued functions whose outputs are finite numbers (definition

†2 For aesthetics, MathComp provides the possibility to add to the notation `\is` (and similarly to the notation `\isn't`) the articles `a` or `an`. This is documented in the file `theory/ssr/ssrbool.v` of the standard Coq distribution.

†3 In the literature, a "charge" can also be understood as finitely additive [12].

```
fin_num_fun):
1  Definition fin_num_fun
2      d (T : semiRingOfSetsType d)
3      (R : numDomainType)
4      (mu : set T -> \bar R)  :=
5    forall U, measurable U -> mu U \is a fin_num.
6  HB.mixin Record SigmaFinite_isFinite
7      d (T : semiRingOfSetsType d)
8      (R : numDomainType)
9      (mu : set T -> \bar R) :=
10   { fin_num_measure : fin_num_fun mu }.
```

The reader can observe that `fin_num_fun` is defined with four parameters (`d`, `T`, `R`, and `mu` at lines 2–4) and that the first ones can be inferred from the type of the last one. Arguments that can be inferred from others are called *implicit parameters* and they are not required by Coq upon application. This explains why `fin_num_fun` has only one argument at line 10[†4].

The interface is defined by the command `HB.mixin` which is provided by Hierarchy-Builder. An interface takes the form of a Coq record. Interfaces are used to define structures. For example, the interface above is used to define the structure `FinNumFun` of functions that respect this interface:

```
HB.structure Definition FinNumFun
    d (T : semiRingOfSetsType d)
    (R : numFieldType) :=
  { mu of SigmaFinite_isFinite _ T R mu }.
```

The notation `{mu of ...mu...}` is intentionally reminiscent of Coq's sigma-types (noted `{x : T & P x}` in the standard library) but what Hierarchy-Builder does under the hood is actually to implement a *packed class* [20]. More precisely, the command

```
HB.structure Definition M :=
  {A of f1 & f2 & ... & fn}.
```

equips `A` with the interfaces `f1`, `f2`, ..., `fn`. Concretely, it creates a Coq `Record` with a parameter corresponding to `A` where each field is one of the interfaces applied to the parameter, a dependent pair whose first field corresponds to `A` and whose second field is an instance of the `Record` mentioned just above, a module that contains the `Record` and the dependent pair, and unification hints [15, Sect. 3.2].

Figure 1 explains the naming of the interface. The primary use of `SigmaFinite_isFinite` is to define the structures of finite measures, i.e., this is

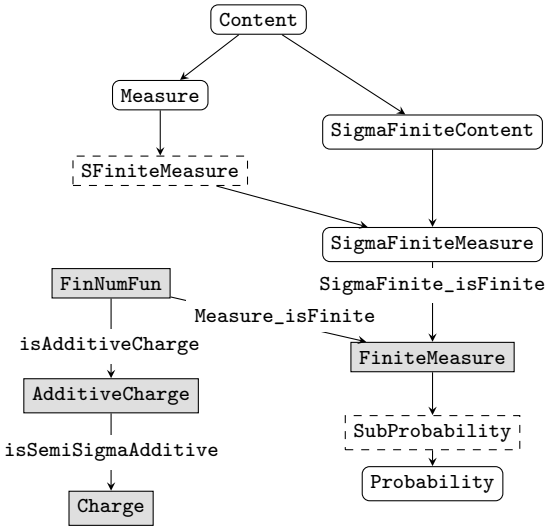

Fig. 1 Hierarchy of structures for measures. Rounded boxes represent structures from previous work [3]. Square boxes represent structures introduced by this paper: filled boxes represent structures that are strictly needed to define charges and finite measures, dashed boxes represent structures needed to accommodate a hierarchy of kernels from previous work [6]. This paper does not deal directly with s-finite measures and probability measures, but they are displayed for the sake of completeness.

the interface used to define the structure of finite measures *from* the structure of σ-finite measures. More precisely, the mathematical structure of finite measures is defined by combining the interface `SigmaFinite_isFinite` and the structure of σ-finite measures [3, Sect. 4.3].

For the definition of charges in itself, we mimick MathComp-Analysis where measures are defined as the structure of σ-additive functions that extends the structure of contents. We therefore introduce an interface for additive charges:

```
HB.mixin Record isAdditiveCharge
    d (T : semiRingOfSetsType d)
    (R : numFieldType)
    (mu : set T -> \bar R) :=
  { charge_semi_additive : semi_additive mu }.
```

The predicate `semi_additive` is coming from MathComp-Analysis and it is a formalization of the notion of additivity that we saw in Sect. 2.1

†4 The user can prefix identifiers with `@` to enforce the explicit use of implicit parameters.

when introducing the definition of measures. It is defined using the library of finitely iterated operators of MathComp (Sect. 2. 2. 1) as follows:

```
1  Definition semi_additive
2    d (R : numFieldType)
3   (T : semiRingOfSetsType d)
4   (mu : set T -> \bar R) := forall F n,
5    (forall k, measurable (F k)) ->
6    trivIset setT F ->
7    measurable (\big[setU/set0]_(k < n) F k) ->
8    mu (\big[setU/set0]_(i < n) F i) =
9      \sum_(i < n) mu (F i).
```

Note that at line 7 there is a condition about the iterated union being measurable. This is why we refer to it as "semi-additivity". When `F` is a family of sets over a measurable type, this condition is always true, which corresponds to the mathematical notion of additivity.

We use the interface `isAdditiveCharge` to define the structure of additive charges:

```
HB.structure Definition AdditiveCharge
    d (T : semiRingOfSetsType d)
    (R : numFieldType) :=
  { mu of isAdditiveCharge d T R mu &
          FinNumFun d mu }.
```

The definition indicates clearly that this structure inherits from the interface `isAdditiveCharge` and the structure `FinNumFun`.

Finally, we define the structure of charges as additive charges that are moreover σ-additive:

```
HB.mixin Record isSemiSigmaAdditive
    d (T : semiRingOfSetsType d)
    (R : numFieldType)
    (mu : set T -> \bar R) :=
  { charge_semi_sigma_additive :
      semi_sigma_additive mu }.

HB.structure Definition Charge
    d (T : semiRingOfSetsType d)
   (R : numFieldType) :=
  { mu of isSemiSigmaAdditive d T R mu &
          AdditiveCharge d mu }.
```

Thanks to Hierarchy-Builder, Coq now considers (σ-additive) charges as a subtype of additive charges. We finally define a notation `{charge set T -> \bar R}` for the type of charges over the measurable type `T`.

Compared with charges, the formalization of finite measures requires additional care because of its use in the definition of s-finite measures. How to deal with this apparent circularity is explained in [6] in the more general case of kernels.

Eventually, the definitions of charges and of finite measures fit in the hierarchy of mathematical structures for measures of MathComp-Analysis seen in Fig. 1.

3. 1. 1 Building Instances of Charge

Looking at the definition of charges as displayed in Fig. 1, at first sight it looks like, to build an instance of charge, one needs beforehand to build an instance of `FinNumFun`, then an instance of `AdditiveCharge`, and finally an instance of `Charge`. This would be cumbersome because, after all, additivity is a consequence of σ-additivity. A textbook would not go through these detours and we now explain how we can indeed avoid them.

For that kind of situations, Hierarchy-Builder introduces the notion of *factory*. A factory is an interface that is used to build instances of several structures at the same time. For example, we provide, using the command `HB.factory`, a factory for charges whose interface consists of (1) the fact that the measure of the empty set is 0, (2) finiteness, and (3) σ-additivity:

```
HB.factory Record isCharge
    d (T : semiRingOfSetsType d)
    (R : realFieldType)
    (mu : set T -> \bar R) := {
  charge0 : mu set0 = 0 ;
  charge_finite : forall x, d.-measurable x ->
    mu x \is a fin_num ;
  charge_sigma_additive :
    semi_sigma_additive mu }.
```

Note that this interface does not feature additivity explicitly. Instead, we prove once and for all that it can be derived from the `isCharge` factory using the commands `HB.builders`/`HB.end`:

```
HB.builders Context
d (T : semiRingOfSetsType d) (R : realFieldType)
mu (_ : isCharge d T R mu).

Let finite : fin_num_fun mu.
Proof. (* omitted *) Qed.
HB.instance Definition _ :=
  SigmaFinite_isFinite.Build d T R mu finite.

Let semi_additive : semi_additive mu.
Proof. (* omitted *) Qed.
HB.instance Definition _ :=
  isAdditiveCharge.Build d T R mu semi_additive.

Let semi_sigma_additive : semi_sigma_additive mu.
Proof. (* omitted *) Qed.
HB.instance Definition _ :=
```

```
  isSemiSigmaAdditive.Build d T R
      mu semi_sigma_additive.
HB.end.
```

In the code above, we assume a function `mu` that satisfies the interface `isCharge` (unnamed hypothesis `isCharge d T R mu`) using the `HB.builders` command. Using this information, we build instances of the structures `FinNumFun`, `AdditiveCharge`, and `Charge`, according to their definitions in Fig. 1. As a consequence of the code above, we can now use the function `isCharge.Build` to build an instance of a charge, the instances for the structures `AdditiveCharge` and `FinNumFun` being generated automatically.

The interface `Measure_isFinite` of Fig. 1 provides another example of factory: it builds a finite measure from the structures of measure and of `FinNumFun`, producing automatically all the structures below in the hierarchy [2, file `measure.v`].

3.1.2 From Charges to Measures

There is a bit of theory to develop about charges. For illustration, we can mention the construction of a measure (which is non-negative by definition) using a charge that happens to be non-negative. First, we provide a definition:

```
Definition measure_of_charge
  d (T : measurableType d) (R : realType)
  (nu : set T -> \bar R)
  (_ : forall E, 0 <= nu E) := nu.
```

In this definition, `nu` is expected to be a charge. The last parameter is an anonymous hypothesis (a.k.a. a phantom type [21]). Given such an hypothesis (say, `nupos`), a charge is obviously a measure, i.e., we can prove that the measure of the empty set is 0 (proof `mu0` below), that it is σ-additive (proof `mu_sigma_additive`) (these facts are directly derived from the definition of charge), and that it is non-negative (proof `mu_ge0` below, established thanks to the `nupos` hypothesis). Using these proofs, we can declare an instance of measure using the `isMeasure.Build` constructor from MathComp-Analysis [3, Sect. 3.5.2] and the command `HB.instance` from Hierarchy-Builder:

```
HB.instance Definition _ :=
  isMeasure.Build d R T
  (measure_of_charge nupos)
  mu0 mu_ge0 mu_sigma_additive.
```

Thanks to this definition, Coq correctly infers the type of `{measure set T -> \bar R}` when given a non-negative charge.

This section has introduced the three main commands of Hierarchy-Builder to create hierarchies of mathematical structures (namely, `HB.mixin`, `HB.structure`, and `HB.instance`). See [3, Sect. 3.1] for another overview of Hierarchy-Builder.

3.2 The Hahn Decomposition Theorem

The Hahn decomposition theorem says that given a charge, a measurable type can be partitioned into a *positive set* and a *negative set*. We first define positive and negative sets in Sect. 3.2.1. We then give an overview of a standard proof of the Hahn decomposition theorem in Sect. 3.2.2 to give an idea of the elements that come into play. In Sect. 3.2.3, we explain the formalization of the Hahn decomposition theorem by focusing in particular on the inductive construction of a particular sequence of sets.

3.2.1 Formal Definition of Negative and Positive Sets

Negative and positive sets are defined using charges (Sect. 3.1). Given a charge ν over a measurable type `T`, a set N is a negative set when it is measurable and when for all measurable sets $A \subseteq N$, $\nu(A) \leq 0$:

```
Definition negative_set
  (nu : {charge set T -> \bar R}) (N : set T) :=
 measurable N /\
 forall A, measurable A -> A `<=` N -> nu A <= 0.
```

To emphasize the charge, we introduce the Coq notation `nu.-negative_set` for `negative_set nu`.

The definition of *positive sets* is similar:

```
Definition positive_set
  (nu : {charge set T -> \bar R}) (P : set T) :=
 measurable P /\
 forall E, measurable E -> E `<=` P -> nu E >= 0.
```

Similarly to negative sets, positive sets come with the Coq notation `nu.-positive_set` for `positive_set nu`.

3.2.2 Proof of the Hahn Decomposition Theorem

We consider a measurable space (T, Σ_T) and a charge ν. Given a set D and a set A, we define the following extended real number $d(A) \stackrel{\text{def}}{=} \sup\{\nu(E) \mid \text{measurable}(E), E \subseteq D \setminus A\}$. Observe that d is non-negative.

Lemma 3.1. *Given a set D and a set A, there exists a measurable set B such that $B \subseteq D \setminus A$, and* $\min(d(A)/2, 1) \leq \nu(B)$.

Proof. When $d(A) = 0$, the proof is by choosing the witness $\emptyset$. Otherwise, the idea is to prove that $\min(d(A)/2, 1) < d(A)$ and to use the following property of the supremum: for any set S and any extended real number x, $x < \sup(S)$ implies that there exists y such that $y \in S$ and $x < y$. □

Lemma 3.2. *Given a measurable set D, there exists a ν-negative set A such that $A \subseteq D$ and $\nu(A) \leq \nu(D)$.*

Proof. By Lemma 3.1, there exists a measurable set A_0 such that $A_0 \subseteq D$ and $\min(d(\emptyset)/2, 1) \leq \nu(A_0)$. Let $g_0 = d(\emptyset)$ and $U_0 = A_0$. By induction, we can build three sequences A_i, g_i, U_i such that $A_{n+1} \subseteq D \setminus U_n$, $g_{n+1} = d(U_n)$, $U_{n+1} = U_n \cup A_{n+1}$ (again using Lemma 3.1). The desired set A is $D \setminus \bigcup_n A_n$. □

Note that in Lemma 3.2, the measure of D is not needed to be non-positive because we can always use the empty set as a subset with a non-positive measure for the base case of the induction.

Given a set A, we consider the extended real number $s(A) \stackrel{\text{def}}{=} \inf\{\nu(x) \mid \text{measurable}(E), E \subseteq A^{\complement}\}$. Observe that s is non-positive.

Lemma 3.3. *Given A, there exists a ν-negative set B such that $B \subseteq A^{\complement}$, and $\nu(B) \leq \max(s(A)/2, -1)$.*

Proof. When $s(A) = 0$, the proof is by choosing the witness $\emptyset$. Otherwise, we first show that $s(A) < \max(s(A)/2, -1)$. By using a property of the infimum, we obtain a measurable set B' such that $B' \subseteq A^{\complement}$ and $\nu(B') < \max(s(A)/2, -1)$. Finally, we use Lemma 3.2 to obtain the desired witness. □

Theorem 3.1 (Hahn decomposition)**.** *Let (T, Σ_T) be a measurable space and ν be a charge over Σ_T. There exist a ν-negative set N and a ν-positive set P such that $T = P \uplus N$.*

Proof. The proof consists in an explicit construction of the set N. The set N is built as $\bigcup_n A_n$ where A_n is a sequence of sets built as follows. There exists a negative set A_0 such that $\nu(A_0) \leq \max(s(\emptyset)/2, -1)$ (using Lemma 3.3). By induction, we can build three sequences A_i, z_i, U_i such that $z_0 = s(\emptyset)$, $U_0 = A_0$, $A_{n+1} \subseteq U_n^{\complement}$, $z_{n+1} = s(U_n)$, $U_{n+1} = U_n \cup A_{n+1}$ (again using Lemma 3.3). The set P is taken to be $N^{\complement}$. We omit the details about the non-trivial proof that P is indeed a positive set. □

3.2.3 Formalization of the Hahn Decomposition Theorem

We define the partition of the Hahn decomposition theorem by the following predicate:

```
Definition hahn_decomposition
    d (T : measurableType d) (R : realType)
    (nu : {charge set T -> \bar R}) P N :=
  [/\ nu.-positive_set P, nu.-negative_set N,
      P `|` N = setT & P `&` N = set0].
```

This makes for a short formal statement of the Hahn decomposition theorem:

```
Context d (T : measurableType d) (R : realType).
Variable nu : {charge set T -> \bar R}.
Theorem Hahn_decomposition :
  exists P N, hahn_decomposition nu P N.
```

The heart of the proof of Theorem 3.1 is the construction of the sequences A_i, z_i, U_i. For this purpose, we use the following lemma that, given a binary relation `R` and a proof that every element `x` has a "successor" `y` (i.e., an element `y` such that `R x y`), produces a sequence `f` obeying a relation `R` given an initial element `x0`:

```
Context X (R : X -> X -> Prop).
Lemma dependent_choice_Type :
  (forall x, {y | R x y}) ->
  forall x0, {f | f 0 = x0 /\
                  forall n, R (f n) (f n.+1)}.
```

This lemma is key to formalize the proof of Theorem 3.1. Surprisingly, its usefulness does not seem to be well-known. As a matter of fact, even though it is part of the standard library of Coq, we could not use it right away because it was specialized with `X` of type `Set` and needed to be generalized to apply here[†5].

First, we define a type for the elements of the sequence (A_i, z_i, U_i). An object of this type is such that $z_i \leq 0$ and A_i is a negative set such that $\nu(A_i) \leq \max(z_i/2, -1)$. These properties are captured by the predicate `elt_prop`:

```
Let elt_prop (x : set T * \bar R) :=
  [/\ x.2 <= 0,
      nu.-negative_set x.1 &
      nu x.1 <= maxe (x.2 * 2^-1%:E) (- 1%E) ].
```

The type of an element (A_i, z_i, U_i) is `elt_type`:

```
Let elt_type := {AzU : set T * \bar R * set T |
```

†5 See the pull request `https://github.com/coq/coq/pull/16382`.

```
  elt_prop AzU.1}.
```

Second, we define the relation between two successive elements (A_i, z_i, U_i) and (A_j, z_j, U_j) of the sequence. The definition `s_` corresponds to the function $A \mapsto s(A)$ of Sect. 3.2.2 (see [2, file `charge.v`] for details).

```
Let elt_rel i j := [/\ z_ j = s_ (U_ i),
  A_ j `<=` ~` U_ i &
  U_ j = U_ i `|` A_ j].
```

Last, we provide a lemma that given a U_i computes A_{i+1}; this is the formalization of Lemma 3.3:

```
Let next_elt U : s_ U <= 0 -> { A |
  [/\ A `<=` ~` U,
      nu.-negative_set A &
      nu A <= maxe (s_ U * 2^-1%R%:E) (- 1%E)] }.
```

Using the above elements, we can explain the proof script to construct N. First, we define z_0 as $s(\emptyset)$ (`s_ set0`). At line 2, we build A_0 from $\emptyset$ to build A_0 (`A0`) together with the proof that it is a negative set (`negA0`) and that $\nu(A_0) \leq \max(s(\emptyset)/2, -1)$ (`A0s0`). We take $z_0 = s(\emptyset)$ and $U_0 = A_0$. We build the sequence (A_i, z_i, U_i) as `v` at line 4. The inductive construction of the sequence happens between lines 9–14. Given U_i (`U`) at line 10, we define z_{i+1} as `s_ U` and build A_{i+1} (`A'`) at line 12 from which we build $U_{i+1} = U_i \cup A_{i+1}$ (`U `|` A'`) at line 13. The set N is defined at line 17.

```
1  (* build A_0 *)
2  have [A0 [_ negA0 A0s0]] := next_elt set0.
3  (* build the sequences A_i, z_i, U_i *)
4  have [v [v0 Pv]] : {v |
5    v 0%N = exist _ (A0, s_ set0, A0)
6      (And3 (s_le0 set0) negA0 A0s0) /\
7    forall n, elt_rel (v n) (v n.+1)}.
8    (* obtain U_i *)
9    apply: dependent_choice_Type =>
10     -[[[A s] U] [/= s_le0' nsA]].
11   (* build A_{i+1} *)
12   have [A' [? nsA' A's'] ] := next_elt U.
13   by exists (exist _ (A', s_ U, U `|` A')
14     (And3 (s_le0 U) nsA' A's')).
15 ...
16 (* build N *)
17 set N := \bigcup_k (A_ (v k)).
```

See [2, file `charge.v`] for details and for the rest of the proof.

4 The Radon-Nikodým Theorem in Coq

Given two measures μ and ν, the Radon-Nikodým theorem establishes the existence of a function f such that $\nu(A) = \int_{x \in A} f(x)(\mathbf{d}\,\mu)$ for all measurable sets A. Here, μ is a σ-finite measure and ν is a charge which is *dominated* by μ. The function f is called the Radon-Nikodým derivative of ν w.r.t. μ; it is indeed unique up-to almost-everywhere equality. It can be written $\mathbf{d}\nu/\mathbf{d}\mu$ in analogy with derivatives of functions.

We explain the formal statement of the Radon-Nikodým theorem in Sect. 4.1. Section 4.2 provides an overview of a standard proof which highlights the main elements whose formalization is the purpose of Sect. 4.3.

4.1 Statement of the Radon-Nikodým Theorem

Given a measurable space (T, Σ_T) and two (signed or unsigned) measures μ_1 and μ_2 over T, μ_2 *dominates*[†6] μ_1 (written $\mu_1 \ll \mu_2$) when $\mu_2(A) = 0$ implies $\mu_1(A) = 0$ for all measurable sets $A \in \Sigma_T$. This definition translates directly in MathComp-Analysis:

```
Context d (T : measurableType d) (R : realType).
Definition measure_dominates m1 m2 :=
 forall A, measurable A -> m2 A = 0 -> m1 A = 0.
Notation "m1 `<< m2" :=
 (measure_dominates m1 m2).
```

Theorem 4.1 (Radon-Nikodým). *Let (T, Σ_T) be a measurable space. Given a σ-finite measure μ and a charge ν on Σ_T such that $\nu \ll \mu$, there exists an extended real-valued function f that is integrable w.r.t. μ and that satisfies $\nu(A) = \int_{x \in A} f(x)(\mathbf{d}\,\mu)$ for all measurable sets A. Moreover, any two functions with this property are equal almost-everywhere on T.*

The statement of Theorem 4.1 translates directly in MathComp-Analysis using its formalization of measure and integration theory and the formalization of charges we developed in Sect. 3.1:

```
1  Context d (T : measurableType d)
2          (R : realType).
3  Variables (nu : {charge set T -> \bar R})
4  (mu : {sigma_finite_measure set T -> \bar R}).
5
6  Local Lemma Radon_Nikodym0 : nu `<< mu ->
7    exists f : T -> \bar R, [/\
8      (forall x, f x \is a fin_num),
```

†6 One can also say that μ_1 is "absolutely continuous" w.r.t. μ_2. However, we use "domination" for measures to avoid confusion with the absolutely continuous functions seen in Sect. 1.

```
9       mu.-integrable [set: T] f &
10      forall A, measurable A ->
11                nu A = \int[mu]_(x in A) f x.
```

First, note that the proviso about almost-everywhere equality of theorem 4.1 is a consequence of a generic lemma that can be found in MathComp-Analysis [2, lemma `integral_ae_eq`]. This proviso gives us the freedom to pick, for the derivative, a function that always returns a real number; this is the meaning of the line 8 above. As indicated by the use of the Coq command modifier `Local`, this is not the statement of the Radon-Nikodým theorem that we expose to the user. Instead, we introduce an identifier and a notation for Radon-Nikodým derivatives. We define `Radon_Nikodym nu mu` to be the witness `f` of `Radon_Nikodym0` when `nu` is dominated by `mu` and the constant function $-\infty$ otherwise:

```
Definition Radon_Nikodym nu mu :=
  match pselect (nu `<< mu) with
  | left nu_mu =>
    sval (cid (Radon_Nikodym0 nu_mu))
  | right _ => cst -oo
  end.
```

The definition `Radon_Nikodym` is an example of a technical idiom. The identifier `pselect` corresponds to a version of the law of excluded middle [5, Sect. 5.2]. It is implemented by a disjunction that can be pattern-matched. When the proposition is true (`left` case), we extract from the lemma `Radon_Nikodym0` a witness. This requires first the use of the axiom of constructive indefinite description `cid` (introduced in Coq to implement Hilbert's epsilon) and the use of the generic function `sval` to extract the first projection of a dependent pair. Constant functions are written with `cst` in MathComp-Analysis.

We then associate the identifier `Radon_Nikodym` with the ASCII notation `'d nu '/d mu`, which of course stands for $\mathbf{d}\nu/\mathbf{d}\mu$:

```
Notation "'d nu '/d mu" := (Radon_Nikodym nu mu).
```

Eventually, we break up `Radon_Nikodym0` into three lemmas that express the properties of `'d nu '/d mu`, i.e., the fact that it is real-valued:

```
Lemma Radon_Nikodym_fin_num x :
    nu `<< mu ->
  ('d nu '/d mu) x \is a fin_num.
```

the fact that it is integrable:

```
Lemma Radon_Nikodym_integrable mu nu :
    nu `<< mu ->
  mu.-integrable [set: T] ('d nu '/d mu).
```

and the property of its integral:

```
Lemma Radon_Nikodym_integral mu nu :
    nu `<< mu ->
  forall A, measurable A ->
    nu A = \int[mu]_(x in A) ('d nu '/d mu) x.
```

4.2 Proof of the Radon-Nikodým Theorem

The first and main step of the proof of the Radon-Nikodým theorem is to prove it for finite measures.

4.2.1 Radon-Nikodým for Finite Measures

The main idea of this proof is to introduce a set $\mathcal{G}$ of non-negative and integrable functions g whose integrals under-approximate $\nu(E)$ for all measurable sets E, i.e., such that $\int_{x\in E} g(x)(\mathbf{d}\,\mu) \leq \nu(E)$. The proof consists in showing that there exists a function $f \in \mathcal{G}$ such that $\int_x f(x)(\mathbf{d}\,\mu) = \sup\{\int_x g(x)(\mathbf{d}\,\mu) \mid g \in \mathcal{G}\} \stackrel{\text{def}}{=} M$ and that this function satisfies $\nu(E) = \int_{x\in E} f(x)(\mathbf{d}\,\mu)$ for all measurable sets E.

From the definition of M, we get a sequence g_k of functions in $\mathcal{G}$ such that $\int_x g_k(x)(\mathbf{d}\,\mu) > M - 1/(k+1)$. We define $F_k(x) \stackrel{\text{def}}{=} \max\{g_i(x) \mid i \leq k\}$ and $f = \lim_{n\to\infty} F_n$.

We then introduce a covering partition of the domain of the functions F_m and g_k in the form of sets $E_{m,j}$ such that $x \in E_{m,j}$ if and only if $j \leq m$ is the smallest natural number such that $F_m(x) = g_j(x)$:

$$E_{m,j} \stackrel{\text{def}}{=} \{x \mid F_m(x) = g_j(x) \wedge \forall k < j,\ g_k(x) < g_j(x)\}.$$

Given m, the sets $E_{m,j}$ are pairwise-disjoint for all j and the sets $E_{m,j}$ for all $j \leq m$ cover the whole set T (recall that (T, Σ_T) is the measurable space on which μ and ν are defined). Using the sets $E_{m,j}$, we can show that $F_m \in \mathcal{G}$ and that $f \in \mathcal{G}$ by the monotone convergence theorem, which leads ultimately to the proof that $\int_x f(x)(\mathbf{d}\,\mu) = M$, which concludes the first part of the proof.

We show that $\int_{x\in E} f(x)(\mathbf{d}\,\mu) = \nu(E)$ for any measurable set E by contradiction. We suppose that there is a measurable set A such that $\int_{x\in A} f(x)(\mathbf{d}\,\mu) < \nu(A)$. Then we can define ε such that $\int_{x\in A}(f(x)+\varepsilon)(\mathbf{d}\,\mu) < \nu(A)$ and define a charge $\sigma(B) \stackrel{\text{def}}{=} \mu(B) - \int_{x\in B}(f(x)+\varepsilon)(\mathbf{d}\,\mu)$.

By the Hahn decomposition theorem (Theorem 3.1), there exist a positive set P and a negative set N for σ. We define:

$$h(x) \stackrel{\text{def}}{=} \begin{cases} f(x)+\varepsilon & x \in A \cap P \\ f(x) & \text{otherwise.} \end{cases}$$

We can show that, on the one hand, for all $S \subseteq A \cap P$, $\int_{x\in S} h(x)(\mathbf{d}\,\mu) \leq \nu(S)$ and, on the other hand, for all $S \subseteq A \cap P^{\complement}$, $\int_{x\in S} h(x)(\mathbf{d}\,\mu) \leq \nu(S)$. This leads to $h \in \mathcal{G}$ which is impossible because $\int_x h(x)(\mathbf{d}\,\mu) > M$.

Once Radon-Nikodým is proved for finite measures, it can be generalized to the case where μ is σ-finite, and then to the case where ν is a charge.

4.3 Formalization of the Proof of the Radon-Nikodým Theorem

The pencil-and-paper proof of the previous section (Sect. 4.2) actually provides a useful hint about how to organize its formalization: it goes through building a specific charge. This indicates that we should proceed by first formalizing the elements that participate to its construction. In this section, we therefore explain the Coq formalization of the set $\mathcal{G}$, the bound M, the functions g_k, F_m, and f, and eventually the charge σ.

Let us first assume two measures `mu` and `nu`. The set $\mathcal{G}$ of functions g can be defined directly as the following definition `approxRN` using the predicates `integrable`, `measurable`, and the definition of the Lebesgue integral:

```
Definition approxRN := [set g : T -> \bar R |
  [/\ forall x, 0 <= g x,
      mu.-integrable [set: T] g &
      forall E, measurable E ->
              \int[mu]_(x in E) g x <= nu E] ].
```

The set $\{\int_x g(x)(\mathbf{d}\,\mu) \mid g \in \mathcal{G}\}$ of integrals and its supremum (M in the pencil-and-paper proof) can be defined using set comprehension and `ereal_sup`:

```
Definition int_approxRN :=
  [set \int[mu]_x g x | g in approxRN].
Definition sup_int_approxRN :=
  ereal_sup int_approxRN.
```

The definition of the functions g_k requires a proof of the existence of such functions as a sequence whose elements belong to $\mathcal{G}$ and such that $\int_x g_k(x)(\mathbf{d}\,\mu) > M - 1/(k+1)$ for each k. Since this proof requires that the measure ν is finite, we assume hereafter that `nu` has type `{finite_measure set X -> \bar R}`:

```
Lemma approxRN_seq_ex :
  { g : (T -> \bar R)^nat |
    forall k, g k \in approxRN /\
      \int[mu]_x g k x > M - k.+1%:R^-1%:E }.
```

Since the proof of existence of the functions g_k takes the form of a sigma-type, we can obtain the sequence g_k by taking the first projection using the generic function `sval`:

```
Definition approxRN_seq : (X -> \bar R)^nat :=
  sval approxRN_seq_ex.
```

To define F_m, it suffices to take the maximum of the functions g_k using the generic iterated operators from MathComp (see Sect. 2.2):

```
Definition max_approxRN_seq m x :=
  \big[maxe/-oo]_(k < m.+1) g_ k x.
```

The function f is the limit of the functions F_m. Given the definitions so far, the start of the proof of the Radon-Nikodým theorem is the matter of the following declarations (where `mu` and `nu` are both finite measures as far as this first step is concerned):

```
Let G := approxRN mu nu.
Let M := sup_int_approxRN mu nu.
Let g := approxRN_seq mu nu.
Let F := max_approxRN_seq mu nu.
Let f := fun x => lim (F ^~ x).
```

It remains to show that f is the function searched for, see [2] for these details.

We now move on to explain the construction of the charge σ. When constructing σ, we are in the second part of the proof sketched in the previous section. This is a proof by contradiction in the context of which we are given a measurable set A such that $\int_{x\in A} f(x)(\mathbf{d}\,\mu) < \nu(A)$, that is, we are in the following local context:

```
Context A (mA : measurable A)
        (h : \int[mu]_(x in A) f x < nu A).
```

In this precise context, we first prove the existence of an $\varepsilon > 0$ such that $\int_{x\in A}(f(x)+\varepsilon)(\mathbf{d}\,\mu) < \nu(A)$:

```
Lemma epsRN_ex : {eps : {posnum R} |
  \int[mu]_(x in A) (f x + eps%:num%:E) < nu A}.
```

Since the conclusion of the lemma `epsRN_ex` is a sigma-type, we can obtain the wanted ε by taking its first projection:

```
Definition epsRN := sval epsRN_ex.
```

We now have enough material to define the function σ (however we do not know yet whether it is a charge):

```
Definition sigmaRN B :=
  nu B - \int[mu]_(x in B) (f x + epsRN%:num%:E).
```

To show that the function σ is actually a charge, we need to show that it satisfies the interface `SigmaFinite_isFinite`, i.e., that `sigmaRN` is not infinite (see Sect. 3.1):

```
Let fin_num_sigmaRN B :
  measurable B -> sigmaRN B \is a fin_num.
```

We also need to show that `sigmaRN` is additive so that it satisfies the interface `isAdditiveCharge`:

```
Let sigmaRN_semi_additive :
  semi_additive sigmaRN.
```

And we also need to show that `sigmaRN` is σ-additive, i.e., that it satisfies the interface `isCharge`:

```
Let sigmaRN_semi_sigma_additive :
  semi_sigma_additive sigmaRN.
```

Now that σ has been shown to be a genuine charge, we can apply the Hahn decomposition theorem of Sect. 3.2.3 to pursue and conclude the proof of the Radon-Nikodým theorem sketched in the previous section. See [2, lemma `radon_nikodym_finite`] for the remaining details of the proof of Radon-Nikodým for finite measures, [2, lemma `radon_nikodym_sigma_finite`] for the generalization of `mu` to a σ-finite measure, and [2, theorem `Radon_Nikodym0`] for the final statement where `nu` is a charge. In particular, this last step makes use of the Jordan decomposition of a charge into a difference of two measures (this is where we use the construction `measure_of_charge` from Sect. 3.1.2).

4.4 Properties of the Radon-Nikodým Derivative

We can go on proving the basic properties of the Radon-Nikodým derivative (formally defined in Sect. 4.1), such as linearity (see lemmas `Radon_Nikodym_cscale` and `Radon_Nikodym_cadd` in [2, file `charge.v`]) or the "chain rule", i.e., $\mathbf{d}\nu/\mathbf{d}\lambda \overset{\text{a.e.}}{=} \mathbf{d}\nu/\mathbf{d}\mu \cdot \mathbf{d}\mu/\mathbf{d}\lambda$ with $\nu \ll \mu \ll \lambda$, whose formalization we now explain.

To prove the chain rule, we first formalize a lemma akin to a change of variables (or to integration by substitution): $\int_E f(\mathbf{d}\,\nu) = \int_E f \cdot \mathbf{d}\nu/\mathbf{d}\mu(\mathbf{d}\,\mu)$ with $\nu \ll \mu$ and f a ν-integrable function:

```
Context d (T : measurableType d) (R : realType).
Variables (nu : {finite_measure set T -> \bar R})
  (mu : {sigma_finite_measure set T -> \bar R}).
Hypothesis numu : nu `<< mu.

Lemma Radon_Nikodym_change_of_variables f E :
  measurable E -> nu.-integrable E f ->
\int[mu]_(x in E) (f x *
  ('d (charge_of_finite_measure nu) '/d mu) x) =
\int[nu]_(x in E) f x.
```

The proof is by using the monotone convergence theorem [3, Sect. 6].

The chain rule can be proved using the above lemma for changes of variables:

```
Context d (T : measurableType d) (R : realType).
Variables (nu : {charge set T -> \bar R})
  (la : {sigma_finite_measure set T -> \bar R})
  (mu : {finite_measure set T -> \bar R}).

Lemma Radon_Nikodym_chain_rule :
  nu `<< mu -> mu `<< la ->
  ae_eq la [set: T] ('d nu '/d la)
    ('d nu '/d mu \*
     'd (charge_of_finite_measure mu) '/d la).
```

Equality holds almost-everywhere, the predicate `ae_eq mu D f g` states almost-everywhere equality between two functions and is a simple wrapper for the more generic notation `{ae mu, forall x, P x}` from MathComp-Analysis [3, Sect. 6.5]. The notation `\*` is for point-wise multiplication of two functions. See [2, file `charge.v`] for more details about the formalization of the chain rule.

5 The Lebesgue-Stieltjes Measure

This section documents the construction of the Lebesgue-Stieltjes measure. As we explained in Sect. 1.2, this is one important construction to prove the Fundamental Theorem of Calculus and in probability theory in general where it corresponds to distribution functions. Given a non-decreasing right-continuous real function f, the Lebesgue-Stieltjes measure Λ_f is the measure such that the length of an interval $]a, b]$ is $f(b) - f(a)$. This generalizes the Lebesgue measure, which is the special case where f is taken to be the identity function.

We start by recalling results from previous work [3]: generic lemmas for measure construction in Sect. 5.1 and the semiring of sets of open-closed intervals in Sect. 5.2. We then define the length of open-closed intervals in Sect. 5.3 and explain the properties of this length function. Among these properties, we explain σ-subadditivity in detail in Sect. 5.4. We conclude by building the Lebesgue-Stieltjes measure by extension in Sect. 5.5.

5.1 Measure Extension in MathComp-Analysis

A standard way to construct measures (over a σ-algebra) is by extension of a measure over a semiring of sets (to produce a measure over the σ-algebra generated by the semiring of sets). To perform such a measure extension, MathComp-Analysis pro-

vides a generic construction whose main element is a definition `measure_extension`.

The definition `measure_extension` is an outer measure (notation `...^*`) from a measure `mu` over a semiring of sets `T`:

```
Context d (T : semiRingOfSetsType d)
        (R : realType).
Variable mu : {measure set T -> \bar R}.
Let I := [the measurableType _ of
  salgebraType (@measurable _ T)].
Definition measure_extension : set I -> \bar R
  := mu^*.
```

The type `I` is the σ-algebra generated from the semiring of sets `T`. The notation `[the aType of b]` is a HIERARCHY-BUILDER notation to infer explicitly the type `aType` from `b` provided that `b` has indeed been shown to be an instance of `aType`. The formalization of generated σ-algebras is explained in [3, Sect. 3.3] and the formalization of outer measures is explained in [3, Sect. 4.1].

To construct a measure over a semiring of sets it suffices to construct a content (Sect. 2.1) over the semiring of sets and to show that it is σ-subadditive. Indeed, MATHCOMP-ANALYSIS provides the proof of a generic lemma that establishes that a σ-subadditive content is actually a σ-additive measure. It is easier to construct a measure by extension rather than by a direct construction in part because it is easier to prove σ-subadditivity rather than σ-additivity directly. The Measure Extension theorem of MATHCOMP-ANALYSIS has already been used to construct the Lebesgue measure in [3, Sect. 5].

5.2 The Semiring of Sets of Open-closed Intervals from MathComp-Analysis

The construction of the Lebesgue-Stieltjes measure by extension starts by defining the semiring of sets of open-closed intervals. This step is shared by the construction of the Lebesgue measure and the contents of this paragraph are therefore borrowed from previous work [3, Sect. 5.1]. Let `ocitv` be the set of such intervals:

```
Variable R : realType.
Definition ocitv_type : Type := R.
Definition ocitv :=
  [set `]x.1, x.2]%classic | x in [set: R * R]].
```

The delimiter `%classic` corresponds to a notation scope where intervals are interpreted as the corresponding set; strictly speaking, `ocitv` is therefore a set of sets. The notation to define sets by comprehension and the notation for intervals have been introduced in Sect. 2.2.2.

This set forms a semiring of sets because it contains the empty set `set0` (proof `ocitv0`), it is closed under finite intersection (proof `ocitvI`), and it is "closed under finite difference" (see Sect. 2) (proof `ocitvD`). We use these proofs to declare a HIERARCHY-BUILDER instance of semiring of sets attached to the identifier `ocitv_type`[†7]:

```
HB.instance Definition _ :=
  @isSemiRingOfSets.Build
  (ocitv_display R) ocitv_type
  (Pointed.class R)
  ocitv ocitv0 ocitvI ocitvD.
```

5.3 The Length of Intervals and its Properties

We define the length of an interval (in the sense of the Lebesgue-Stieltjes measure) and prove its properties. Recall that the Lebesgue-Stieltjes measure is parameterized by a non-decreasing, right-continuous *real* function; since the goal is to produce a measure, and thus an *extended real*-valued function, we need to embed the real function f using:

```
Definition er_map T T' (f : T -> T')
    (x : \bar T) : \bar T' :=
  match x with
  | r%:E => (f r)%:E
  | +oo => +oo
  | -oo => -oo
  end.
```

For the sake of generality, the length of an interval is defined over arbitrary sets for which we take the hull using `Rhull` (see [2, file `normedtype.v`]):

```
Let g : \bar R -> \bar R := er_map f.
Definition wlength (A : set (ocitv_type R))
    : \bar R :=
  let i := Rhull A in g i.2 - g i.1.
```

Let us introduce the type `cumulative R` of non-decreasing, right-continuous real functions. Provided that `f` is non-decreasing, the function `wlength`

†7 The occurrence of `Pointed.class R` in the declaration of this instance is a technicality that can be ignored by the reader. It is necessary with versions of MATHCOMP-ANALYSIS using MATHCOMP version 1; its need disappears with MATHCOMP version 2. A version of MATHCOMP-ANALYSIS compatible with MATHCOMP version 2 is scheduled for release in January 2024.

is non-negative:

```
Lemma wlength_ge0
    (f : cumulative R) (I : set (ocitv_type R)) :
  (0 <= wlength f I)%E.
```

The function `wlength` is also semi-additive (`f` is not required to be cumulative here):

```
Lemma wlength_semi_additive (f : R -> R) :
  semi_additive (wlength f).
```

As a consequence, `wlength f` is a content when `f` is cumulative:

```
HB.instance Definition _ (f : cumulative R) :=
  isContent.Build _ _ R (wlength f)
  (wlength_ge0 f) (wlength_semi_additive f).
```

The function `isContent.Build` is the constructor associated with the structure `Content` (Fig. 1). In addition, `wlength f` is a σ-subadditive when `f` is cumulative:

```
Lemma wlength_sigma_sub_additive
    (f : cumulative R) :
  sigma_sub_additive (wlength f).
```

As a consequence, it is also a measure:

```
HB.instance Definition _ (f : cumulative R) :=
 Content_SubSigmaAdditive_isMeasure.Build _ _ _
  (wlength f) (wlength_sigma_sub_additive f).
```

To build the measure, we use here the function `Content_SubSigmaAdditive_isMeasure.Build` which is associated with an interface that extends a content to a measure with the proof that it is σ-subadditive [3, Sect. 5.2].

Among the properties above, the proof that `wlength` is σ-subadditive is actually where the mathematical difficulty lies, we therefore detail this proof in the next section.

5.4 Proof of the σ-subadditivity of the Length of Intervals

We give an overview of the proof of the σ-subadditivity of the `wlength` function, illustrated with intermediate goals as displayed by Coq when running the proof script from MathComp-Analysis. Our goal in this section is to show that, aesthetics aside, following the proof in Coq is not much different from following the proof on paper. See [2, lemma `wlength_sigma_sub_additive`] for the complete formalization. Before explaining the proof in Sect. 5.4.2, we state an intermediate lemma in Sect. 5.4.1.

5.4.1 Intermediate Lemma

The proof of σ-subadditivity of `wlength` uses the following intermediate lemma.

Lemma 5.1. *Let f be a cumulative function. Let D be a finite set of natural numbers and a, b be two sequences of real numbers such that $a_i \leq b_i$ for any $i \in D$. Then, for any two real numbers a_0, b_0, we have that*

$$]a_0, b_0] \subseteq \bigcup_{i \in D}]a_i, b_i]$$

implies

$$f(b_0) - f(a_0) \leq \sum_{i \in D} f(b_i) - f(a_i).$$

See lemma `wlength_content_sub_fsum` [2, file `lebesgue_stieltjes_measure.v`] for a formal statement and a formal proof.

5.4.2 Proof Overview

We are interested in proving the statement `wlength_sigma_sub_additive` from Sect. 5.3. This amounts to prove that, given a cumulative function `f`, for any open-closed interval I and for any sequence of open-closed intervals A_n, we have that

$$I \subseteq \bigcup_n A_n$$

implies

$$\texttt{wlength}_{\texttt{f}}(I) \leq \sum_n \texttt{wlength}_{\texttt{f}}(A_n),$$

which appears in Coq as:

```
============================
I `<=` \bigcup_n A n ->
(wlength f I <= \sum_(n <oo) wlength f (A n))%E
```

The notation `\sum_(n <oo) h n`, where `h` is an extended real numbers-valued function, combines the iterated operators of MathComp (Sect. 2.2.1) with the notion of limits of MathComp-Analysis (Sect. 2.2.2) to implement enumerable sums (see [3, Sect. 2.1]). The notation `\bigcup_n A n` corresponds to an enumerable union.

Let I be $]a_1, a_2]$. Assuming $a_1 < a_2$ (otherwise this is trivial), we are led to prove:

```
============================
((f a.2 - f a.1)%:E <=
 \sum_(n <oo) wlength f (A n))%E
```

Recall from Sect. 2.2.2 that `%:E` is for injecting real numbers into the set of extended real numbers.

Let A_n by $]b_{n1}, b_{n2}]$ for all n. Without loss of generality (we omit the proof which is technical), we can assume that for all n, $b_{n1} \leq b_{n2}$.

The proof goes on by using the "epsilon trick". First, we introduce a positive ε by using the following equivalence (lemma `lee_addgt0Pr`):

$$\forall x, y \in \overline{\mathbb{R}}.\ x \leq y \leftrightarrow \forall \varepsilon > 0, x \leq y + \varepsilon.$$

This changes the goal to $\forall \varepsilon > 0, f(a_2) - f(a_1) \leq$

$\sum_n \texttt{wlength}_\texttt{f}\,(A_n) + \varepsilon$ which appears in Coq as:

```
e : {posnum R}
============================
((f a.2 - f a.1)%:E <=
  \sum_(n <oo) wlength f (A n) + (e%:num)%:E)%E
```

See Sect. 2.2.2 for the type `{posnum R}`. Second, we perform some arithmetic manipulations and apply a generic lemma (namely, `epsilon_trick`):

$$\forall A_i, \varepsilon \in \mathbb{R}_{\geq 0}.\ \sum_i \left(A_i + \frac{\varepsilon}{2^{i+1}}\right) \leq \sum_i A_i + \varepsilon.$$

This changes the goal to

$$f(a_2) - f(a_1) - \frac{\varepsilon}{2} \leq \sum_i \left(\texttt{wlength}_\texttt{f}\,(A_i) + \frac{\varepsilon/2}{2^{i+1}}\right) \tag{1}$$

which appears in Coq as:

```
e : {posnum R}
============================
((f a.2 - f a.1)%:E + (- (e%:num / 2))%:E <=
 \sum_(i <oo) (wlength f (A i) +
  (e%:num / 2 / (2 ^ i.+1)%:R)%:E))%E
```

As explained in Sect. 2.2.1, the notation `%:R` is to inject natural numbers into the set of real numbers.

Using the right continuity of `f`, we can find c and D_i's such that:

- $f(a_1 + c) \leq f(a_1) + \varepsilon/2$, and
- $\forall i, f(b_{i2} + D_i) \leq f(b_{i2}) + \frac{\varepsilon/2}{2^{i+1}}$.

Moreover, we have

$$[a_1 + \frac{c}{2}, a_2] \subseteq \bigcup_i]b_{i1}, b_{i2} + D_i[.$$

Since the left-hand side of this inclusion is compact and since the right-hand side is an enumerable union of open intervals, we can find a finite number of indices X such that

$$[a_1 + \frac{c}{2}, a_2] \subseteq \bigcup_{i \in X}]b_{i1}, b_{i2} + D_i[.$$

This is the Borel-Lebesgue property that is provided to us by MathComp-Analysis.

We use this last fact to prove the equation (1) by transitivity. Concretely, we use the value B equal to

$$\sum_{i \in X} \left(\texttt{wlength}_\texttt{f}\,(]b_{i1}, b_{i2}]) + f(b_{i2} + D_i) - f(b_{i2})\right).$$

On the one hand, we prove

$$f(a_2) - f(a_1) - \frac{\varepsilon}{2} \leq B,$$

which appears in Coq as

```
============================
((f a.2 - f a.1 - e%:num / 2)%:E <= B)%E
```

The proof is as follows:

$$\begin{aligned} f(a_2) - f(a_1) - \frac{\varepsilon}{2} &\leq f(a_2) - f\left(a_1 + \frac{c}{2}\right) && (2)\\ &\leq \sum_{i \in X} \left(f(b_{i2} + D_i) - f(b_{i1})\right) && (3)\\ &\leq B && (4) \end{aligned}$$

The step (3) uses the Lemma 5.1.

On the other hand, we prove

$$B \leq \sum_i \left(\texttt{wlength}_\texttt{f}\,(A_i) + \frac{\varepsilon/2}{2^{i+1}}\right)$$

which appears in Coq as

```
============================
(B <=
  \sum_(i <oo) (wlength f (A i) +
    (e%:num / 2 / (2 ^ i.+1)%:R)%:E))%E
```

The proof is as follows:

$$\begin{aligned} B &= \sum_{i \in X} \left(\texttt{wlength}_\texttt{f}\,(]b_{i1}, b_{i2}]) + f(b_{i2} + D_i) - f(b_{i2})\right) && (5)\\ &\leq \sum_{i \in X} \left(\texttt{wlength}_\texttt{f}\,(A_i) + \frac{\varepsilon/2}{2^{i+1}}\right) && (6)\\ &\leq \sum_i \left(\texttt{wlength}_\texttt{f}\,(A_i) + \frac{\varepsilon/2}{2^{i+1}}\right) && (7) \end{aligned}$$

This completes the proof of the σ-subadditivity of the function `wlength`.

5.5 Construction of the Lebesgue-Stieltjes Measure

Since `wlength` is a measure on the semiring of sets of open-closed intervals, we can use the definition `measure_extension` from MathComp-Analysis (Sect. 5.1) to define the Lebesgue-Stieltjes measure:

```
Definition lebesgue_stieltjes_measure
    (f : cumulative R) :=
  measure_extension
    [the measure _ _ of wlength f].
```

This construction provides a measure that applies to a σ-algebra generated from open-closed intervals. If we use for the `f` function the identity function, we recover the Lebesgue measure as a special case, see [2, file `lebesgue_measure.v`].

6 Related Work

To the best of our knowledge, the formalizations explained in this paper (charges, Hahn decomposition theorem, Radon-Nikodým theorem, Lebesgue-Stieltjes measure) have never been carried out in the Coq proof assistant. However, similar results

can be found in other proof assistants such as Isabelle/HOL, HOL, and Lean (we do not know of such proofs in Agda, PVS, or Mizar).

Isabelle/HOL has had an extensive formalization of measure and integration theory since 2011 starting with seminal work by Hölzl et al. [24]. With the contents of this paper, MathComp-Analysis now covers the same material as [24] and some aspects of our work are even a bit more general. In particular, to the best of our understanding, the Radon-Nikodým theorem in [24] is specialized to non-negative measures: [24] does not mention charges or signed measures and the Isabelle/HOL scripts available online [25] indicates that Radon-Nikodým derivatives are non-negative extended real-valued functions. However, it should be said that signed measures and the Hahn decomposition theorem have recently been added to the Isabelle Archive of Formal Proofs [16] as an independent development, i.e., it is not used to prove the Radon-Nikodým theorem. Our formalization of the Lebesgue-Stieltjes measure in Coq is also a generalization compared to [24]. Yet, Avigad et al. did formalize the Lebesgue-Stieltjes measure in Isabelle/HOL along their formalization of the Central Limit Theorem [7]. They briefly explain in prose the main difficulty of applying the Carathéodory extension theorem for that purpose [7, Sect. 3.8]. In comparison, we provide more technical and concrete details in the context of a dependently type theory in Sect. 5. 4.

One can also find the Radon-Nikodým theorem in the HOL proof assistant [31]. In [31], the Radon-Nikodým theorem is stated for finite measures, the last version of the scripts available online [14] indicates that it has been extended to σ-finite measures, however neither indicates a generalization to charges. Mhamdi et al. [31] point at other applications of the Radon-Nikodým theorem such as the formalization of the Kullback-Leibler divergence.

The mathlib library [37] of the Lean proof assistant also has an extensive formalization of measure theory in which one can find the Radon-Nikodým theorem (as advertised in this blog entry [39]) and the Lebesgue-Stieltjes measure (which is documented by the proof scripts). Regarding the Lebesgue-Stieltjes measure, to the best of our understanding, it is not constructed directly using a measure extension theorem, contrary to the construction we explain in Sect. 5 (see also [3, Sect. 7.1] which compares the construction of the Lebesgue measure between MathComp-Analysis and mathlib). The formalization of charges we propose in Sect. 3. 1 is also different: our definition blends into a hierarchy of structures (Fig. 1) whereas mathlib defines a signed measure as an instance of a vector measure [37, file `VectorMeasure.lean`]. Finally, we formalize a proof using the Hahn decomposition theorem about charges whereas mathlib's proof uses Lebesgue's decomposition theorem. Since it is possible to prove Lebesgue's decomposition theorem using the Hahn decomposition theorem, no generality seems lost.

The formalization of the Fundamental Theorem of Calculus that we have been using as a motivating example has already been explored in Coq [17]. This is however a version using the Riemann integral of continuous functions and a constructive proof while we are dealing with the Lebesgue integral. Other formalizations of the Fundamental Theorem of Calculus are listed in the "Formalizing 100 Theorems" list [38] but to the best of our understanding none is dealing with absolutely continuous functions.

7 Conclusion

In this paper, we have extended the measure theory of MathComp-Analysis with new formalizations of standard constructions (charges and the Lebesgue-Stieltjes measure) and standard theorems (Hahn decomposition and Radon-Nikodým theorems). We have chosen to formalize these constructions and theorems because they are useful to deal with the semantics of probabilistic programs and to formalize mathematics in general. To the best of our knowledge, this is the first time that they are proved in the Coq proof assistant. We focused our explanations on how we turn pencil-and-paper proofs into proof scripts, emphasizing the main technical choices that let us formalize without departing from the mathematical literature and explaining the reusable parts of our formalization, in particular the hierarchy of charges and measures. These additions are now available as reusable lemmas in MathComp-Analysis [2].

Our next step is to formalize the Fundamental Theorem of Calculus of Sect. 1. 2 using the Radon-Nikodým theorem and the Lebesgue-Stieltjes mea-

sure. Our plan is to follow the approach of [9, Sect. 3]. Note that there are other approaches to prove the Fundamental Theorem of Calculus for the Lebesgue integral that do not rely on the Radon-Nikodým theorem (e.g., [32]—which actually shares similarities with [9]). In relation with this goal, more formal theories are needed about absolute continuity and bounded variation. An important point is to connect the definitions of absolute continuity for functions and the one of domination for measures with the following lemma: a function f is absolutely continuous (footnote †1) if and only if the Lebesgue-Stieltjes measure associated with f is absolutely continuous w.r.t. the Lebesgue measure (footnote †6). Section 1. 2 highlighted the need to show that the Radon-Nikodým derivative coincides with the usual derivative. We can use the Lebesgue differentiation theorem for this purpose. It says that when $f : \mathbb{R} \to \mathbb{R}$ is a function that is integrable in a neighborhood of x, then

$$\lim_{r\to 0} \frac{1}{\Lambda[x - r/2, x + r/2]} \int_{t=x-r/2}^{x+r/2} f(t)(\mathbf{d}\,\Lambda) = f(x).$$

From this theorem, we can show that a Radon-Nikodým derivative f^{RN} is the derivative f' as follows:

$$\begin{aligned} f^{RN}(x) &= \lim_{r\to 0} \frac{1}{r} \int_{t=x-r/2}^{x+r/2} f^{RN}(t)(\mathbf{d}\,\Lambda) \\ &= \lim_{r\to 0} \frac{f(x + r/2) - f(x - r/2)}{r} \\ &= f'(x). \end{aligned}$$

We are currently investigating the formalization of the elements explained above using MathComp-Analysis.

Acknowledgments

The authors are grateful to Zachary Stone and Cyril Cohen for their guidance and code review, to Quentin Vermande for his code review, and to the anonymous reviewers for their comments. This work has also benefited from the technical inputs and general feedback from Takafumi Saikawa and Jacques Garrigue. This work is partially supported by JSPS KAKENHI Grant Number JP22H00520.

References

[1] R. Affeldt. *An Introduction to MathComp-Analysis*. 2023. `https://staff.aist.go.jp/reynald.affeldt/documents/karate-coq.pdf`, since 2022.

[2] R. Affeldt, Y. Bertot, C. Cohen, M. Kerjean, A. Mahboubi, D. Rouhling, P. Roux, K. Sakaguchi, Z. Stone, P.-Y. Strub, and L. Théry. MathComp-Analysis: Mathematical components compliant analysis library. `https://github.com/math-comp/analysis`, 2023. Since 2017. Version 0.6.7.

[3] R. Affeldt and C. Cohen. Measure construction by extension in dependent type theory with application to integration. *J. Autom. Reason.*, 67(3):28, 2023.

[4] R. Affeldt, C. Cohen, M. Kerjean, A. Mahboubi, D. Rouhling, and K. Sakaguchi. Competing inheritance paths in dependent type theory: A case study in functional analysis. In *10th International Joint Conference on Automated Reasoning (IJCAR 2020), Paris, France, July 1–4, 2020*, volume 12167 of *Lecture Notes in Computer Science*, pages 3–20. Springer, 2020. `https://hal.inria.fr/hal-02463336/document`.

[5] R. Affeldt, C. Cohen, and D. Rouhling. Formalization techniques for asymptotic reasoning in classical analysis. *J. Formaliz. Reason.*, 11(1):43–76, 2018. `https://jfr.unibo.it/article/view/8124/8407`.

[6] R. Affeldt, C. Cohen, and A. Saito. Semantics of probabilistic programs using s-finite kernels in Coq. In *ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2023), Boston, Massachusetts, USA, January 16–17, 2023*. ACM Press, Jan 2023. `https://hal.inria.fr/hal-03917948/document`.

[7] J. Avigad, J. Hölzl, and L. Serafin. A formally verified proof of the central limit theorem. *J. Autom. Reason.*, 59(4):389–423, 2017.

[8] A. Bagnall and G. Stewart. Certifying the true error: Machine learning in Coq with verified generalization guarantees. In *33rd AAAI Conference on Artificial Intelligence, 31st Conference on Innovative Applications of Artificial Intelligence, 9th Symposium on Educational Advances in Artificial Intelligence, Honolulu, Hawaii, USA, January 27–February 1, 2019*, pages 2662–2669. AAAI Press, 2019.

[9] D. Bárcenas. Fundamental theorem of calculus for Lebesgue integral. *Divulgaciones Matemáticas*, 8(1):75–85, 2000.

[10] G. Barthe, J.-P. Katoen, and A. Silva, editors. *Foundations of Probabilistic Programming*. Cambridge University Press, 2021. `https://www.cambridge.org/core/books/foundations-of-probabilistic-programming/819623B1B5B33836476618AC0621F0EE`.

[11] Y. Bertot, G. Gonthier, S. O. Biha, and I. Pasca. Canonical big operators. In *21st International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2008), Montreal, Canada, August 18–21, 2008*, volume 5170 of *Lecture Notes in Computer Science*, pages 86–101. Springer, 2008.

[12] K.P.S. Bhaskara Rao and M. Bhaskara Rao. *Theory of Charges: A Study of Finitely Additive*

Measures. Academic Press, 1983

[13] S. Bhat, A. Agarwal, R. W. Vuduc, and A. G. Gray. A type theory for probability density functions. In *39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2012), Philadelphia, Pennsylvania, USA, January 22–28, 2012*, pages 545–556. ACM, 2012.

[14] A. Coble, M. Qasim, and O. Hasan. `https://github.com/HOL-Theorem-Prover/HOL/blob/develop/src/probability/lebesgueScript.sml`, 2023. Source code of the HOL theorem prover library.

[15] C. Cohen, K. Sakaguchi, and E. 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. `https://hal.inria.fr/hal-02478907/document`.

[16] M. Cousin, M. Echenim, and H. Guiol. The Hahn and Jordan decomposition theorems. `https://www.isa-afp.org/entries/Hahn_Jordan_Decomposition.html`, Nov 2021. Isabelle Archive of Formal Proofs.

[17] L. Cruz-Filipe. A constructive formalization of the fundamental theorem of calculus. In *Selected Papers of the 2nd International Workshop on Types for Proofs and Programs (TYPES 2002), Berg en Dal, The Netherlands, April 24–28, 2002*, volume 2646 of *Lecture Notes in Computer Science*, pages 108–126. Springer, 2002.

[18] F. Dahlqvist, A. Silva, and D. Kozen. *Semantics of Probabilistic Programming: A Gentle Introduction*, chapter Chapter 1 of [10]. Cambridge University Press, 2021.

[19] F. de Marçay. Intégration. `https://gcomte.perso.math.cnrs.fr/Math421/Merker%20Francois%20de%20Marcay%20%20Integration.pdf`, 2022. Département de Mathématiques d 'Orsay Université Paris-Sud, France. The year corresponds to the last access, the pdf document is not dated.

[20] F. Garillot, G. Gonthier, A. Mahboubi, and L. Rideau. Packaging mathematical structures. In *22nd International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2009), Munich, Germany, August 17–20, 2009*, volume 5674 of *Lecture Notes in Computer Science*, pages 327–342. Springer, 2009.

[21] G. Gonthier. Type design patterns for computer mathematics. In *7th ACM SIGPLAN Workshop on Types in Language Design and Implementation (TLDI 2011), Austin, TX, USA, January 25, 2011*, page 1–2. ACM Press, 2011.

[22] G. Gonthier, A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, S. L. Roux, A. Mahboubi, R. O'Connor, S. O. Biha, I. Pasca, L. Rideau, A. Solovyev, E. Tassi, and L. Théry. A machine-checked proof of the odd order theorem. 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 163–179. Springer, 2013.

[23] P. G. Haselwarter, E. Rivas, A. Van Muylder, T. Winterhalter, C. Abate, N. Sidorenco, C. Hriţcu, K. Maillard, and B. Spitters. SSProve: A foundational framework for modular cryptographic proofs in Coq. *ACM Trans. Program. Lang. Syst.*, 45(3), jul 2023.

[24] J. Hölzl and A. Heller. Three chapters of measure theory in Isabelle/HOL. In *2nd International Conference on Interactive Theorem Proving (ITP 2011), Berg en Dal, The Netherlands, August 22–25, 2011*, volume 6898 of *Lecture Notes in Computer Science*, pages 135–151. Springer, 2011.

[25] J. Hölzl. Theory radon_nikodym. `https://isabelle.in.tum.de/library/HOL/HOL-Analysis/Radon_Nikodym.html`, 2023. Source code of the Isabelle/HOL library.

[26] Y. Ishiguro and R. Affeldt. A progress report on formalization of measure theory with MathComp-Analysis. In *25th JSSST Workshop on Programming and Programming Languages (PPL 2023), Nagoya, Japan, March 6–8, 2023*, 2023. Reviewed. Informal proceedings.

[27] V. Kadets. *A Course in Functional Analysis and Measure Theory*. Springer, 2018.

[28] A. Klenke. *Probability Theory: A Comprehensive Course*. Springer, 2020. 2rd edition.

[29] X. Leroy. Formal verification of a realistic compiler. *Commun. ACM*, 52(7):107–115, 2009.

[30] A. Mahboubi and E. Tassi. *Mathematical Components*. Zenodo, Jan 2021. `https://zenodo.org/record/7118596`.

[31] T. Mhamdi, O. Hasan, and S. Tahar. Formalization of entropy measures in HOL. In *2nd International Conference on Interactive Theorem Proving (ITP 2011), Berg en Dal, The Netherlands, August 22–25, 2011*, volume 6898 of *Lecture Notes in Computer Science*, pages 233–248. Springer, 2011.

[32] R. L. Pouso. A simple proof of the fundamental theorem of calculus for the Lebesgue integral, 2012.

[33] W. Rudin. *Real and Complex Analysis*. McGraw-Hill, 1987. 3rd edition.

[34] A. Saito and R. Affeldt. Experimenting with an intrinsically-typed probabilistic programming language in Coq. In *21st Asian Symposium on Programming Languages and Systems (APLAS 2023), Taipei, Taiwan, November 26–29, 2023*, volume 14405 of *Lecture Notes in Computer Science*, pages 182–202. Springer, 2023.

[35] S. Staton. Commutative semantics for probabilistic programming. In *26th European Symposium on Programming (ESOP 2017), Uppsala, Sweden, April 22–29, 2017*, volume 10201 of *Lecture Notes in Computer Science*, pages 855–879.

Springer, 2017.

[36] The Coq Development Team. *The Coq Proof Assistant Reference Manual*. Inria, 2023. Available at `https://coq.inria.fr/refman/`. Version 8.18.0.

[37] The mathlib community. Lean mathematical components library. `https://github.com/leanprover-community/mathlib`, 2023. Since 2017.

[38] F. Wiedijk. Formalizing 100 theorems. `http://www.cs.ru.nl/~freek/100`, 2023.

[39] K. Ying. The Radon-Nikodym theorem in Lean. Lean community blog, 09 2021. `https://leanprover-community.github.io/blog/posts/the-radon-nikodym-theorem-in-lean/`.

[40] Y. Zhang and N. Amin. Reasoning about "reasoning about reasoning": semantics and contextual equivalence for probabilistic programs with nested queries and recursion. *Proc. ACM Program. Lang.*, 6(POPL):1–28, 2022.