A Progress Report on Formalization of Measure Theory with MathComp-Analysis*

[1] Yoshihiro Ishiguro
Graduate School of Mathematics, Nagoya University

[2] Reynald Affeldt
Digital Architecture Research Center,
National Institute of Advanced Industrial Science and Technology (AIST)

Abstract 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. 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.

Formal Verification of Probabilistic Programs Probabilistic programs provide a way to deal with uncertainty in a rigorous way. They have applications in artificial intelligence, information security, etc. Their semantics relies on probability theory and more generally on measure theory [13].

The Coq proof assistant has arguably been an important tool for the formal verification of programs[1]. 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 [25], 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 [23]. This formalization [6] highlights, among others, the importance of density functions or Radon-Nikodým derivatives. See also [8, Sect. 2.3] that also stresses the importance of Radon-Nikodým derivatives.

*This is a (slightly edited) version of a paper presented at the 25th Workshop on Programming and Programming Languages (PPL2023).

[1] Coq 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.

Formalization of Connections between Derivatives and Integrals 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 [22, Thm 7.18]. It says that for any function f that is *absolutely continuous* [2] 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, which is the matter of another theorem by Lebesgue. See Sect. 7 for more details.

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 first ones for this proof assistant to the best of our knowledge). The Radon-Nikodým theorem establishes the existence of Radon-Nikodým derivatives or density functions. The Lebesgue-Stieltjes measure is a generalization of the Lebesgue measure used in particular in probability theory. Besides these contributions to Coq, this paper also has another objective. We would like 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. We therefore explain in particular what are the reusable parts of our formalization. For example, the Radon-Nikodým uses charges that can be advantageously formalized using Hierarchy-Builder [10], a tool to formalize hierarchies of mathematical structures in Coq.

Outline This paper is organized as follows. Section 2 summarizes mathematical preliminaries and provide 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 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] 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 $\sum_{k=1}^{n} |f(b_k) - f(a_k)| < \epsilon$.

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] 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 *measurable sets* the sets in Σ_T. 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_n 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 countable family of measurable sets F such that $A = \bigcup_i F_i$ and $\mu(F_i) < \infty$ for all i. A measure μ over A is a *finite measure* when $\mu(A) < +\infty$.

A function $f : \Sigma_X \to \Sigma_Y$ is *measurable* when for all measurable sets B, $f^{-1}(B)$ 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

Basic Notations from MathComp We make use of standard MathComp [20] 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]` 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 the scope of rings. Iterated operations 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. In the case of additions, there is a specialized notation `\sum_(k < n | P k) f k`.

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 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`), 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. 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` 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 predicate `x \is a fin_num`. 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[ ``, etc. When an interval `i` is an interval of extended real numbers, we can write `i.1` and `i.2` for its endpoints.

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 $(T, \mathcal{A})$, and `measurable S` when `S` belongs to $\mathcal{A}$. 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`, a measure on `T` is denoted by `{measure set T -> \bar R}` where `R` has type `realType` [3, Sect. 3.5.2]w. Similarly, `{content set T -> \bar R}` is for contents [3, Sect. 3.5.2] and `{sigma_finite_measure set T -> \bar R}` is for σ-finite measures [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 [14] 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 [10] to build hierarchies of mathematical structures.

3.1 Formalization of Charges and Introduction to Hierarchy-Builder

Let $\mathcal{A}$ 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*. 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 `fin_num_fun`):

```
Definition fin_num_fun d (T : semiRingOfSetsType d) (R : numDomainType)
  (mu : set T -> \bar R) := forall U, measurable U -> mu U \is a fin_num.
```

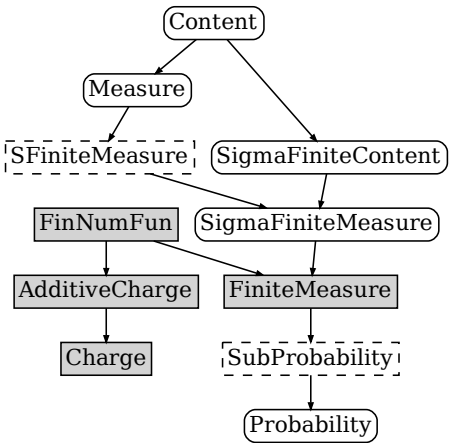

Figure 1. Hierarchy of structures for measures (Round boxes represent structures that predate this paper [3]. Square boxes represent structures introduced for the purpose of this paper. Among them, filled boxes represent structures that are strictly needed to define charges and finite measures. Dashed boxes represent structures needed to accommodate a hierarchy of measure-theoretical kernels from previous work [6]; this paper does not deal directly with s-finite measures and probability measures, they are displayed for the sake of completeness.)

```
HB.mixin Record SigmaFinite_isFinite d (T : semiRingOfSetsType d) (R : numDomainType)
  (mu : set T -> \bar R) := { fin_num_measure : fin_num_fun mu }.
```

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 command `HB.structure` defines structures using a Coq sigma-type composed of a carrier and of the interface(s) it supports.

We also use the interface `SigmaFinite_isFinite` to define the structures of charges and, incidentally, of finite measures, as can be seen in Fig. 1. This figure also explains the naming of this interface: this is the interface used to define the structure of finite measures *from* the structure of σ-finite measures.

For the definition of charges in itself, we follow 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` comes from MathComp-Analysis and is a formal paraphrase of the pencil and paper definition we saw in Sect. 2.1:

```
1  Definition semi_additive := forall F n,
2   (forall k, measurable (F k)) -> trivIset setT F ->
3    measurable (\big[setU/set0]_(k < n) F k) ->
4    mu (\big[setU/set0]_(i < n) F i) = \sum_(i < n) mu (F i).
```

Note that the hypothesis that the iterated union is measurable at line 3 is always true over measurable types.

We use the latter interface 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 isCharge 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 isCharge d T R mu & AdditiveCharge d mu }.
```

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

The mathematical structure of finite measures is defined similarly, by combining the interface `SigmaFinite_isFinite` and the structure of σ-finite measures [3, Sect. 4.3]. 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 measure-theoretical 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.

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 [15]). 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 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, a set N is a negative set when it is measurable and when for all measurable sets $E \subseteq N$, $\nu(E) \leq 0$:

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

The definition of *positive sets* is similar:

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

3.2.2 Proof of the Hahn Decomposition Theorem

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

Lemma 3.1. *Given a set D and a set A such that $0 \leq d(A)$, there exists a measurable set B such that $0 \leq \nu(B)$, $B \subseteq D \setminus A$, and $\min\left(\frac{d(A)}{2}, 1\right) \leq \nu(B)$.*

Proof. This lemma can be proved by using a property of the supremum. □

Lemma 3.2. *Given a measurable set D such that $\nu(D) \leq 0$, there exists a measurable negative set A such that $A \subseteq D$ and $\nu(A) \leq \nu(D)$.*

Proof. When $0 \leq d(A)$, there exists a measurable A_0 such that $0 \leq \nu(A_0)$ and $\min(\frac{d(A)}{2}, 1) \leq \nu(A_0)$ (Lemma 3.1). Since $0 \leq d(\emptyset)$, we can build such a A_0, and then, by induction, we can build three sequences A, d, U such that $U_0 = \emptyset$, $d_{n+1} = d(U_n)$, $A_{n+1} \subseteq D \setminus U_n$, $U_{n+1} = U_n \bigcup A_{n+1}$ (again using Lemma 3.1). The desired set A is $D \setminus \bigcup_n A_n$. □

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

Lemma 3.3. *Given A such that $s(A) \leq 0$, there exists a measurable negative set B such that $\nu(B) \leq 0$, $B \subseteq A^{\complement}$, and $\nu(B) \leq \max\left(\frac{s(A)}{2}, -1\right)$.*

Proof. This lemma can be proved by using Lemma 3.2 and a property of the infimum. □

Theorem 3.1 (Hahn decomposition)**.** *There exist a negative set N and a 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. When $s(A) \leq 0$, there exists a measurable negative set A_0 such that $\nu(A_0) \leq 0$ and $\nu(A_0) \leq \max\left(\frac{s(A)}{2}, -1\right) w$ (using Lemma 3.3). Since $s(\emptyset) \leq 0$, we can build such a A_0, and then, by induction, we can build three sequences A, s, U such that $U_0 = \emptyset$, $s_{n+1} = s(U_n)$, $A_{n+1} \subseteq U_n{}^{\complement}$, $U_{n+1} = U_n \cup A_{n+1}$ (again using Lemma 3.3). The set P is taken to be $N^{\complement}$. We omit 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_decomp d (T : measurableType d) (R : realType)
    (nu : {charge set T -> \bar R}) P N :=
  [/\ positive_set nu P, negative_set nu 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_decomp nu P N.
```

(The `d`, `T`, and `R` parameters do not appear in the expression `hahn_decomp nu P N` because Coq can infer them from the type of `nu`: they are *implicit parameters*.)

We explain how we build the sequences A_i, s_i, U_i of the proof of Theorem 3.1. For this purpose, we use the following lemma[3] that produces a sequence `f` obeying a relation `R` given an initial element `x0` and a proof that an element `x` always has a "successor" `y`:

```
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)}.
```

First, we define a type for the elements of the sequence (A_i, s_i, U_i). An object of this type is such that $s_i \leq 0$ and A_i is a negative set such that $\nu(A_i) \leq \max\left(\frac{s_i}{2}, -1\right)$. These properties are captured by the predicate `elt_prop`; the type of an element (A_i, s_i, U_i) is `elt_type`:

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

Second, we define the relation between two successive elements (A_i, s_i, U_i) and (A_j, s_j, U_j) of the sequence. The definition `ss` corresponds to the function $A \mapsto s(A)$ of Sect. 3.2.2.

```
Let seq_inf i j := [/\ s_ j = ss (U_ i),
  A_ j `<=` setT `\` 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 : ss U <= 0 -> { A | [/\ A `<=` setT `\` U,
  negative_set nu A &
  nu A <= maxe (ss U * 2^-1%R%:E) (- 1%E)] }.
```

Using the above elements, we can explain the proof script to construct N. We defined s_0 as `ss set0` (line 1). From the proof that $s_0 \leq 0$, we build A_0 (line 3). We build the sequence (A_i, s_i, U_i) as `v` at line 6. The inductive construction of the sequence happens between lines 8–12. The set N is defined at line 13. See [17] for details and for the rest of the proof.

```
1  have ss0 : ss set0 <= 0.                          (* build s_0 *)
2    by apply: ereal_inf_lb => /=; exists set0; split => //; rewrite charge0.
3  have [A0 [_ negA0 A0s0]] := next_elt ss0.         (* build A_0 *)
4  have [v [v0 Pv]] : {v |
5      v 0%N = exist _ (A0, ss set0, A0) (And3 ss0 negA0 A0s0) /\
6      forall n, seq_inf (v n) (v n.+1)}.            (* build A_i, s_i, U_i *)
7    apply: dependent_choice_Type => -[[[A d] U] [/= s_le0 negA]].
8    pose s' := ss U.                                (* build s_{i+1} *)
9    have s'_le0 : s' <= 0.
10     by apply: ereal_inf_lb => /=; exists set0; split => //; rewrite charge0.
11   have [A' [? negA' A's'] ] := next_elt s'_le0.   (* build A_{i+1} *)
12   by exists (exist _ (A', s', U `|` A') (And3 s'_le0 negA' A's')).
13 set N := \bigcup_k (A_ (v k)).                    (* build N *)
```

[3] This lemma is part of the standard library of Coq except that until recently it was specialized with `X` of type `Set`. See `https://github.com/coq/coq/pull/16382`.

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.

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 $(T, \mathcal{A})$ and two (signed or unsigned) measures μ_1, μ_2 over T, μ_2 *dominates*[4] μ_1 (written $\mu_1 \ll \mu_2$) when $\mu_2(A) = 0$ implies $\mu_1(A) = 0$ for all measurable sets A of T. This definition translates directly in MathComp-Analysis:

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

Theorem 4.1 (Radon-Nikodým). *Let $(T, \mathcal{A})$ be a σ-algebra. Given a σ-finite measure μ and a charge ν on $\mathcal{A}$ 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 also translates directly in MathComp-Analysis using its formalization of measure and integration theory and the formalization of charges we developed in Sect. 3.1:

```
Theorem Radon_Nikodym d (T : measurableType d) (R : realType)
  (mu : {sigma_finite_measure set T -> \bar R}) (nu : {charge set T -> \bar R}) :
  nu `<< mu ->
  exists2 f : T -> \bar R, mu.-integrable setT f &
    forall E, measurable E -> nu E = \int[mu]_(x in E) f x.
```

The proviso about almost-everywhere equality is a consequence of a generic lemma that can be found in [19].

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.

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 $\int_x g_k(x)(\mathbf{d}\,\mu) > M - \frac{1}{k+1}$. 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 $F_m(x) = g_j(x)$:

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

[4] 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.

The sets $E_{m,j}$ are pairwise disjoint for all j and for each m, and the sets $E_{m,j}$ cover the full set. Using the sets $E_{m,i}$, 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

In this section, we explain the formalization of the main elements of the proof sketched in Sect. 4.2: the set $\mathcal{G}$, the bound M, the functions g_k, F_m, and f, as well as 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 : X -> \bar R | [/\
  forall x, 0 <= g x, mu.-integrable setT 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 of the g functions 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 $\int_x g_k(x)(\mathbf{d}\,\mu) > M - \frac{1}{k+1}$ 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 : (X -> \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 [19] 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 $\int_{x\in A} f(x)(\mathbf{d}\,\mu) < \mu(A)$, 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 that it is σ-additive, i.e., that it satisfies the interface `isCharge`:

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

Since 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 [19, lemma `radon_nikodym_finite_ge0`] for the remaining details of the proof of Radon-Nikodým for finite measures, [19, lemma `radon_nikodym_sigma_finite_ge0`] for the generalization of `mu` to a σ-finite measure, and [19, theorem `Radon_Nikodym`] 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).

5 The Lebesgue-Stieltjes Measure

The last section of this paper documents the construction of the Lebesgue-Stieltjes measure. 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 recall in Sect. 5.1 generic lemmas from previous work [3] that we use in Sect. 5.2 to build the Lebesgue-Stieltjes measure by extension.

5.1 Measure Extension with 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 provides a generic construction whose main elements are a definition `Hahn_ext` (which stands for "Hahn extension") and a lemma `Hahn_ext_sigma_additive`.

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

```
Context d (R : realType) (T : semiRingOfSetsType d).
Variable mu : {content set T -> \bar R}.
Let I := [the measurableType _ of salgebraType (@measurable _ T)].
Definition Hahn_ext : 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`. See [3, Sect. 3.3] for the formalization of generated σ-algebra and [3, Sect. 4.1] for the formalization of outer measures.

The lemma `Hahn_ext_sigma_additive` shows that the measure extension `Hahn_ext` is actually σ-additive. It requires a proof that `mu` is σ-subadditive (hypothesis `mu_sub` below):

```
Hypothesis mu_sub : sigma_sub_additive mu.
Lemma Hahn_ext_sigma_additive : semi_sigma_additive Hahn_ext.
Proof. (* see [2] *) Qed.
```

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 of MathComp-Analysis has already been used to construct the Lebesgue measure in [3, Sect. 5].

5.2 Construction of the Lebesgue-Stieltjes Measure

The construction of the Lebesgue-Stieltjes measure by extension starts by defining the semiring of sets of open-closed intervals. Let `ocitv` by 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 `ocitv_type` identifier is an alias for `R` to be used below.) 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`[5]:

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

Second, we define the length of an interval. Recall that the Lebesgue-Stieltjes measure is parameterized by a non-decreasing *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 EFinf {R : numDomainType} (f : R -> R) : \bar R -> \bar R :=
  fun x => if x is r%:E then (f r)%:E else x.
```

[5]The occurrence of `Pointed.class R` in the declaration of this instance is a technicality that can be ignored by the reader.

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 := EFinf f.
Definition hlength (A : set itvs) : \bar R := let i := Rhull A in g i.2 - g i.1.
```

The function `hlength` is non-negative (proof `hlength_ge0'`) and additive (proof `hlength_semi_additive`). We can therefore instantiate the interface of contents with `hlength`:

```
HB.instance Definition _ (f : cumulative R) :=
  isContent.Build _ R _ (hlength f : set ocitv_type -> _)
    (hlength_ge0' f) (hlength_semi_additive f).
```

(The type `cumulative R` is for non-decreasing real functions.) Since `hlength` is a content, we can use the definition `Hahn_ext` of the previous section (Sect. 5.1) to define the Lebesgue-Stieltjes measure:

```
Definition lebesgue_stieltjes_measure (f : cumulative R) :=
  Hahn_ext [the content _ _ of hlength f : set ocitv_type -> _ ].
```

The function `lebesgue_stieltjes_measure` is σ-additive. Indeed, we can prove that `hlength` is σ-subadditive and use the generic lemma `Hahn_ext_sigma_additive` of Sect. 5.1 to prove that `lebesgue_stieltjes_measure` is σ-additive. The proof that `hlength` is σ-subadditive is actually where the mathematical difficulty lies, see [18, lemma `hlength_sigma_sub_additive`] for its formalization. Given the proofs that `lebesgue_stieltjes_measure` meets the properties of a measure, we can declare its instance:

```
HB.instance Definition _ (f : cumulative R) := isMeasure.Build _ _ _
  (lebesgue_stieltjes_measure f)
  (lebesgue_stieltjes_measure0 f)
  (lebesgue_stieltjes_measure_ge0 f)
  (@lebesgue_stietjes_measure_semi_sigma_additive 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 [18].

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, one can find formal proofs of these results in other proof assistants.

Isabelle/HOL has had an extensive formalization of measure and integration theory since 2011 [16]. With the contents of this paper, MathComp-Analysis now covers the same material. Compared to [16], some aspects of our work are a bit more general thanks to the Lebesgue-Stieltjes measure. As far as we understand, in [16], the Radon-Nikodým theorem seems to be specialized to non-negative measures, [16] does not mention charges or signed measures. Signed measures and the Hahn decomposition theorem have recently been added to the Isabelle Archive of Formal Proofs [11] but as an independent development.

One can also find the Radon-Nikodým theorem in the HOL proof assistant [21]. In [21], the Radon-Nikodým theorem is stated for finite measures, it is not clear from [21] whether it is extended to charges, the source code available online does not seem to indicate so [9]. Mhamdi et al. [21] point at other applications of the Radon-Nikodým theorem such as the formalization of the Kullback-Leibler divergence.

The Lean proof assistant also has an extensive and more recent formalization of measure and integration theory. It contains a formalization of the Radon-Nikodým theorem [24] using Lebesgue's decomposition theorem.

The formalization of the Fundamental Theorem of Calculus that we have been using as a motivating example has already been explored in Coq [12]. This is however a version using the Riemann integral of continuous functions and a constructive proof while we are dealing with the Lebesgue integral.

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. These additions are now available as reusable lemmas in the form of pull requests [17–19] to MathComp-Analysis.

Our next step is to formalize the Fundamental Theorem of Calculus of Sect. 1 using the Radon-Nikodým theorem and the Lebesgue-Stieltjes measure. For this purpose, more formal theories are needed to relate absolutely continuous functions and Lebesgue-Stieltjes measures, as well as to relate other notions as explained below. An important point is to connect the definitions of absolute continuity of functions and the one of domination for measures with the following lemma: a function f is absolutely continuous if and only if the Lebesgue-Stieltjes measure associated with f is absolutely continuous w.r.t. to the Lebesgue measure. We also need to take into account the situation where f is not non-decreasing, in which case Λ_f should intuitively be replaced by a charge. To see that the Radon-Nikodým derivative coincides with the usual derivative, we can use the Lebesgue differentiation theorem. It says that when $f : \mathbb{R} \to \mathbb{R}$ is a function that is integrable in a neighborhood of $x \in \mathbb{R}$, then $\lim_{r\to 0} \frac{1}{\Lambda[x-\frac{r}{2};x+\frac{r}{2}]} \int_{t=x-\frac{r}{2}}^{x+\frac{r}{2}} f(t)(\mathbf{d}\,\Lambda) = f(x)$. From this theorem, we can show that a Radon-Nikodým derivative f^{RN} is the derivative f' as follows:

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

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

Acknowledgments The authors are grateful to Cyril Cohen, Takafumi Saikawa, and Zachary Stone for their guidance. They also thank the anonymous reviewers of PPL2023 for their comments. This work is partially supported by JSPS Kakenhi 22H00520.

References

[1] R. Affeldt. *An Introduction to MathComp-Analysis*. 2022. Intensive course, Graduate School of Mathematics, Nagoya University, `https://www.math.nagoya-u.ac.jp/~garrigue/lecture/2022_affeldt/karate-coq-nagoya2022.pdf`.

[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`, 2022. Since 2017. Version 0.6.0.

[3] R. Affeldt and C. Cohen. Measure construction by extension in dependent type theory with application to integration, 2022. version 2 (2023), `https://arxiv.org/pdf/2209.02345.pdf`.

[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] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] F. Dahlqvist, A. Silva, and D. Kozen. *Semantics of Probabilistic Programming: A Gentle Introduction*, chapter Chapter 1 of [7]. Cambridge University Press, 2021.

[14] 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.

[15] 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.

[16] 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. Online preprint. the Radon-Nikodým theorem in Isabelle/HOL.

[17] Y. Ishiguro and R. Affeldt. Hahn decomposition theorem. https://github.com/math-comp/analysis/pull/777, 2022. PR to [2].

[18] Y. Ishiguro and R. Affeldt. Lebesgue Stieltjes measure. https://github.com/math-comp/analysis/pull/677, 2022. PR to [2].

[19] Y. Ishiguro and R. Affeldt. Radon-Nikodým theorem. `https://github.com/math-comp/analysis/pull/818`, 2022. PR to [2].

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

[21] 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.

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

[23] 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.

[24] 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/`.

[25] 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.