An overview of MathComp-Analysis and its applications

Reynald Affeldt

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

September 27, 2025

Credits

Joint work with

- Cyril Cohen
- Yoshihiro Imai
- Yoshihiro Ishiguro
- Ayumu Saito
- Zachary Stone

MathComp-Analysis' authors:

- R. A.
- Yves Bertot
- Alessandro Bruni
- C. C.
- Marie Kerjean
- Assia Mahboubi
- Kazuhiko Sakaguchi
- Z. S.
- Pierre-Yves Strub
- Laurent Théry

Several contributors (see github)

Context: The Mathematical Components project

https://github.com/math-comp/math-comp

A set of ROCQ packages about mathematics:

- MATHCOMP provides group theory and algebra
- extensions: finite maps (`finmap`), multinomials, etc.
- major results: the Four-Color theorem [Gonthier, 2008], the odd order theorem [Gonthier et al., 2013a], Abel-Ruffini [Bernard et al., 2021], etc.
- tooling: automation (`algebra-tactics`), DSL to build hierarchies of mathematical structures (HIERARCHY-BUILDER), etc.

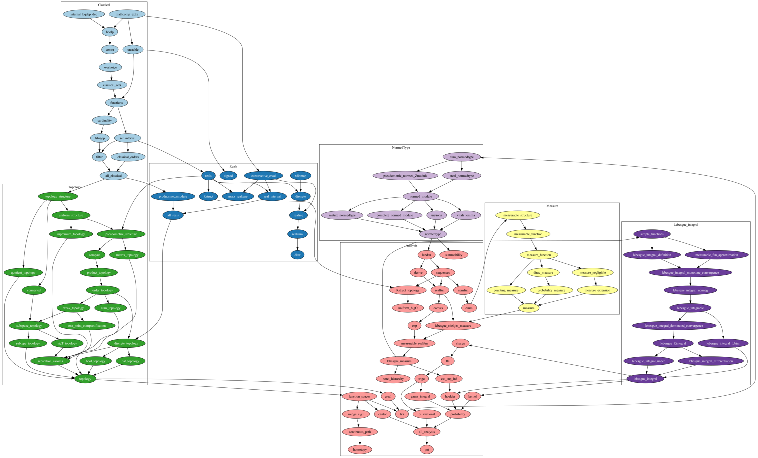
The above libraries are "constructive", in the sense that they do not use classical axioms

Picture taken last month in a high-school in France

The MathComp-Analysis extension

- MathComp-Analysis extends MathComp with classical reasoning
  - However, many "back-ports" are made to MathComp, e.g., automatic discharge of numeric goals

    (see Alessandro Bruni's talk at ITP [Affeldt et al., 2025a])
- MathComp-Analysis is about 100 files for about 90,000 l.o.c.



MathComp-Analysis in terms of Opam packages

1. Classical reasoning: `coq-mathcomp-classical`
2. Real numbers: `coq-mathcomp-reals`
3. Compatibility with Rocq's standard library: `coq-mathcomp-reals-stdlib`, `coq-mathcomp-analysis-stdlib`
   - This enables, e.g., the use of the CoqInterval tactic [Melquiond, 2008]
     (see [Affeldt et al., 2024a] and Alessandro Bruni's talk at ITP [Affeldt et al., 2025a])
4. (Discrete distributions: `coq-mathcomp-experimental-reals`)
5. Analysis per se: `coq-mathcomp-analysis`

Outline

Classical reasoning

`coq-mathcomp-classical` package

Classical axioms can be found in `boolp.v` [Affeldt et al., 2018, Sect. 5]:

- ```
  Axiom functional_extensionality_dep :
    forall (A : Type) (B : A -> Type)
      (f g : forall x : A, B x),
        (forall x : A, f x = g x) -> f = g.
  ```
- ```
  Axiom propositional_extensionality :
    forall P Q : Prop, P <-> Q -> P = Q.
  ```
- ```
  Axiom constructive_indefinite_description :
    forall (A : Type) (P : A -> Prop),
    (exists x : A, P x) -> {x : A | P x}.
  ```
  ```

Naive set theory

`coq-mathcomp-classical` package

Please, consider using `classical_sets.v`[1]:

- complete (many lemmas, 3408 l.o.c.)
- readable statements (notations, consistent naming convention), e.g., using company-coq in emacs:

```
Lemma in_setI (x : T) A B : (x ∈ A ∩ B) = (x ∈ A) && (x ∈ B).
Proof. by apply/idP/andP; rewrite !inE. Qed.
```

```
Lemma setI_bigcupr F P A :
  A ∩ \bigcup_(i in P) F i = \bigcup_(i in P) (A ∩ F i).
Proof.
rewrite predeqE ⇒ t; split ⇒ [[At [k ? ?]]|[k ? [At ?]]];
  by [∃ k |split ⇒ //; ∃ k].
Qed.
```

- well tested (used pervasively in MathComp-Analysis)

More about the `coq-mathcomp-classical` package in Takafumi Saikawa's talk in the next session [Saikawa et al., 2025]

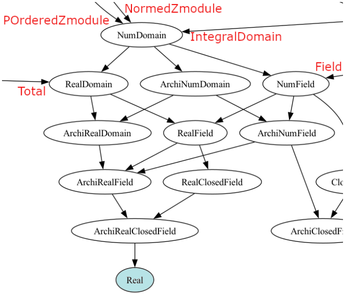
[1] rather than, say, `Ensembles`

Real numbers

coq-mathcomp-reals package

Real numbers are defined in **reals.v** incrementally w.r.t. MathComp.
It is a combination of:

- an archimedean real field (from MathComp)
- a real-closed field (from MathComp)
- with properties of **sup** (from MathComp-Analysis)



Compatibility with the real numbers of Rocq's standard library comes from the fact that they form a model of the resulting interface

Extended real numbers (1/2)

`coq-mathcomp-reals` package

A deceptively simple data structure ($\overline{\mathbb{R}} \stackrel{\text{def}}{=} \mathbb{R} \cup \{-\infty, +\infty\}$) and a surprisingly long theory (`constructive_ereal.v`: 4796 l.o.c)

Extended real numbers (1/2)

`coq-mathcomp-reals` package

A deceptively simple data structure ($\overline{\mathbb{R}} \stackrel{\text{def}}{=} \mathbb{R} \cup \{-\infty, +\infty\}$) and a surprisingly long theory (`constructive_ereal.v`: 4796 l.o.c)

Addition for $\overline{\mathbb{R}}$:

- Unsurprising: $2 + \infty = +\infty$, $2 - \infty = -\infty$, etc.
- But: $\infty - \infty = -\infty$
  - it is not undefined as in the mathematical practice
  - thanks to this, extended real numbers form an additive monoid, enabling the use of MathComp's iterated operators [Bertot et al., 2008]

Extended real numbers (1/2)

`coq-mathcomp-reals` package

A deceptively simple data structure ($\overline{\mathbb{R}} \overset{\text{def}}{=} \mathbb{R} \cup \{-\infty,+\infty\}$) and a surprisingly long theory (`constructive_ereal.v`: 4796 l.o.c)

Addition for $\overline{\mathbb{R}}$:

- Unsurprising: $2 + \infty = +\infty$, $2 - \infty = -\infty$, etc.
- But: $\infty - \infty = -\infty$
  - it is not undefined as in the mathematical practice
  - thanks to this, extended real numbers form an additive monoid, enabling the use of MathComp's iterated operators [Bertot et al., 2008]

Multiplication for $\overline{\mathbb{R}}$:

- Unsurprising:
  - $+\infty \times +\infty = +\infty$, $-\infty \times -\infty = +\infty$
  - $+\infty \times -\infty = -\infty \times +\infty = -\infty$
  - $-2 \times +\infty = -\infty$
- Standard in measure theory: $0 \times \pm\infty = 0$

Extended real numbers (2/2)

`coq-mathcomp-reals` package

Inversion for $\overline{\mathbb{R}}$:

- $\frac{1}{0} = +\infty$, $\frac{1}{+\infty} = 0$
- But: $\frac{1}{-\infty} = -\infty$, thus $\frac{1}{-\infty} \neq -\frac{1}{+\infty}$
  - in fact, $\frac{1}{-x} = -\frac{1}{x}$ only when $x \in \mathbb{R} \setminus \{0\}$

Extended real numbers (2/2)

`coq-mathcomp-reals` package

Inversion for $\overline{\mathbb{R}}$:

- $\frac{1}{0} = +\infty$, $\frac{1}{+\infty} = 0$
- But: $\frac{1}{-\infty} = -\infty$, thus $\frac{1}{-\infty} \neq -\frac{1}{+\infty}$
  - in fact, $\frac{1}{-x} = -\frac{1}{x}$ only when $x \in \mathbb{R} \setminus \{0\}$

Sample application: "average" of a real-valued function f over a set A

$$\frac{1}{\mu(A)} \int_{y \in A} |f(y)| \mathbf{d}\mu$$

Extended real numbers (2/2)

`coq-mathcomp-reals` package

Inversion for $\overline{\mathbb{R}}$:

- $\frac{1}{0} = +\infty$, $\frac{1}{+\infty} = 0$
- But: $\frac{1}{-\infty} = -\infty$, thus $\frac{1}{-\infty} \neq -\frac{1}{+\infty}$
  - in fact, $\frac{1}{-x} = -\frac{1}{x}$ only when $x \in \mathbb{R} \setminus \{0\}$

Sample application: "average" of a real-valued function f over a set A

$$\frac{1}{\mu(A)} \int_{y \in A} |f(y)| \mathbf{d}\mu$$

Formalization using MathComp-Analysis:

```
Definition iavg f A := (mu A)^-1 * \int[mu]_(y in A) `| (f y)%:E |.
```

Extended real numbers (2/2)

`coq-mathcomp-reals` package

Inversion for $\overline{\mathbb{R}}$:

- $\frac{1}{0} = +\infty$, $\frac{1}{+\infty} = 0$
- But: $\frac{1}{-\infty} = -\infty$, thus $\frac{1}{-\infty} \neq -\frac{1}{+\infty}$
  - in fact, $\frac{1}{-x} = -\frac{1}{x}$ only when $x \in \mathbb{R} \setminus \{0\}$

Sample application: "average" of a real-valued function f over a set A

$$\frac{1}{\mu(A)} \int_{y \in A} |f(y)| \mathbf{d}\mu$$

Formalization using MathComp-Analysis:

```
Definition iavg f A := (mu A)^-1 * \int[mu]_(y in A) `| (f y)%:E |.
```

Before version 1.12.0, we were using a cast to reals (`fine`) with an injection back (`%:E`)

```
Definition iavg f A :=
  (fine (mu A))^-1%:E * \int[mu]_(y in A) `| (f y)%:E |.
```

Outline

Measurable structures in MathComp-Analysis

- A $\boxed{\sigma\text{-algebra}}$ Σ_X on a set X is a collection of subsets of X that
  - contains $\emptyset$
  - is closed under complement and
  - countable union

Measurable structures in MathComp-Analysis

- A $\boxed{\sigma\text{-algebra}}$ Σ_X on a set X is a collection of subsets of X that
  - contains $\emptyset$
  - is closed under complement and
  - countable union
- In fact, there are also "poorer" structures of importance:

	semiring of sets	ring of sets	algebra of sets	$\boxed{\sigma\text{-algebra}}$
contains $\emptyset$	✓	✓	✓	✓
closed under $\cap$	✓	✓	✓	✓
closed under "semi-difference"	✓	✓	✓	✓
closed under $\cup$		✓	✓	✓
contains the whole set			✓	✓
closed under countable union				✓

"semi-difference": $\forall A, B \in \mathcal{G}, \exists D \subseteq \mathcal{G}$, D pairwise-disjoint $A \setminus B = \bigcup_{i=1}^{n} D_i$

Measurable structures in MathComp-Analysis

- A $\boxed{\sigma\text{-algebra}}$ Σ_X on a set X is a collection of subsets of X that
  - contains $\emptyset$
  - is closed under complement and
  - countable union
- In fact, there are also "poorer" structures of importance:

	semiring of sets	ring of sets	algebra of sets	$\boxed{\sigma\text{-algebra}}$
contains $\emptyset$	✓	✓	✓	✓
closed under $\cap$	✓	✓	✓	✓
closed under "semi-difference"	✓	✓	✓	✓
closed under $\cup$		✓	✓	✓
contains the whole set			✓	✓
closed under countable union				✓

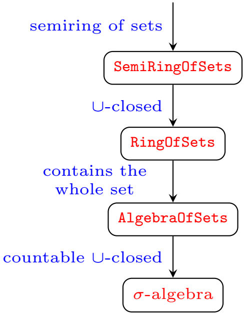
"semi-difference": $\forall A, B \in \mathcal{G}, \exists D \subseteq \mathcal{G}$, D pairwise-disjoint $A \setminus B = \bigcup_{i=1}^{n} D_i$

- These structures intuitively form a hierarchy:

$$\begin{tikzcd}
\boxed{\text{semiring of sets}} \arrow[d, "\cup\text{-closed}"'] \\
\boxed{\text{ring of sets}} \arrow[d, "\text{contains the whole set}"'] \\
\boxed{\text{algebra of sets}} \arrow[d, "\text{countable } \cup\text{-closed}"'] \\
\boxed{\sigma\text{-algebra}}
\end{tikzcd}$$

Measurable structures in MathComp-Analysis

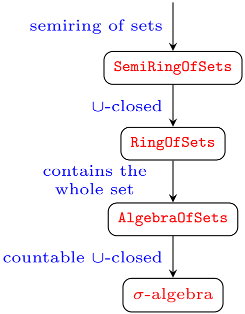
Formalization using Hierarchy-Builder [Cohen et al., 2020] to construct the Lebesgue measure by extension [Affeldt and Cohen, 2023]



Measurable structures in MathComp-Analysis

Formalization using Hierarchy-Builder [Cohen et al., 2020] to construct the Lebesgue measure by extension [Affeldt and Cohen, 2023]

```
HB.mixin Record isSemiRingOfSets (d : measure_display) T := {
  measurable : set (set T) ;
  measurable0 : measurable set0 ;
  measurableI : setI_closed measurable;
  semi_measurableD : semi_setD_closed measurable }.
```

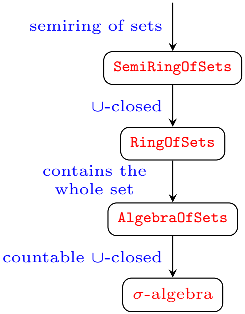


Measurable structures in MathComp-Analysis

Formalization using Hierarchy-Builder [Cohen et al., 2020] to construct the Lebesgue measure by extension [Affeldt and Cohen, 2023]

```
HB.mixin Record isSemiRingOfSets (d : measure_display) T := {
  measurable : set (set T) ;
  measurable0 : measurable set0 ;
  measurableI : setI_closed measurable;
  semi_measurableD : semi_setD_closed measurable }.

HB.structure Definition SemiRingOfSets d :=
  {T of Pointed T & isSemiRingOfSets d T}.
```



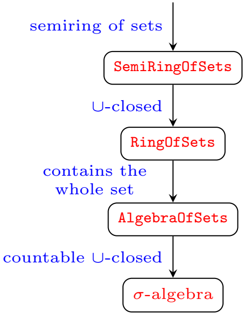
Measurable structures in MathComp-Analysis

Formalization using Hierarchy-Builder [Cohen et al., 2020] to construct the Lebesgue measure by extension [Affeldt and Cohen, 2023]

```
HB.mixin Record isSemiRingOfSets (d : measure_display) T := {
  measurable : set (set T) ;
  measurable0 : measurable set0 ;
  measurableI : setI_closed measurable;
  semi_measurableD : semi_setD_closed measurable }.
```

```
HB.structure Definition SemiRingOfSets d :=
  {T of Pointed T & isSemiRingOfSets d T}.
```

```
HB.mixin Record SemiRingOfSets_isRingOfSets d T of SemiRingOfSets d T :=
  { measurableU : @setU_closed T measurable }.
```



Measurable structures in MathComp-Analysis

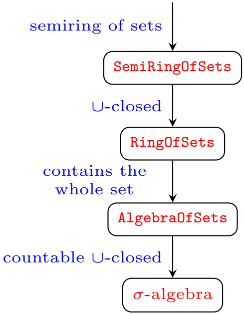
Formalization using Hierarchy-Builder [Cohen et al., 2020] to construct the Lebesgue measure by extension [Affeldt and Cohen, 2023]

```
HB.mixin Record isSemiRingOfSets (d : measure_display) T := {
  measurable : set (set T) ;
  measurable0 : measurable set0 ;
  measurableI : setI_closed measurable;
  semi_measurableD : semi_setD_closed measurable }.

HB.structure Definition SemiRingOfSets d :=
  {T of Pointed T & isSemiRingOfSets d T}.

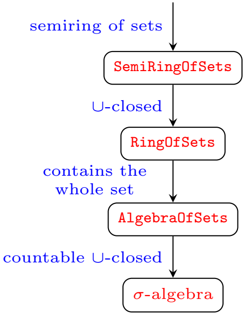
HB.mixin Record SemiRingOfSets_isRingOfSets d T of SemiRingOfSets d T :=
  { measurableU : @setU_closed T measurable }.

HB.structure Definition RingOfSets d :=
  {T of SemiRingOfSets d T & SemiRingOfSets_isRingOfSets d T }.
```



Measurable structures in MathComp-Analysis

Formalization using Hierarchy-Builder [Cohen et al., 2020] to construct the Lebesgue measure by extension [Affeldt and Cohen, 2023]




```
HB.mixin Record isSemiRingOfSets (d : measure_display) T := {
  measurable : set (set T) ;
  measurable0 : measurable set0 ;
  measurableI : setI_closed measurable;
  semi_measurableD : semi_setD_closed measurable }.

HB.structure Definition SemiRingOfSets d :=
  {T of Pointed T & isSemiRingOfSets d T}.

HB.mixin Record SemiRingOfSets_isRingOfSets d T of SemiRingOfSets d T :=
  { measurableU : @setU_closed T measurable }.

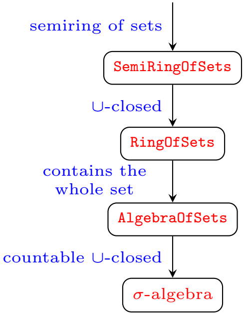
HB.structure Definition RingOfSets d :=
  {T of SemiRingOfSets d T & SemiRingOfSets_isRingOfSets d T }.

HB.mixin Record RingOfSets_isAlgebraOfSets d T of RingOfSets d T :=
  { measurableT : measurable [set: T] }.
```

Measurable structures in MathComp-Analysis

Formalization using Hierarchy-Builder [Cohen et al., 2020] to construct the Lebesgue measure by extension [Affeldt and Cohen, 2023]

```
HB.mixin Record isSemiRingOfSets (d : measure_display) T := {
  measurable : set (set T) ;
  measurable0 : measurable set0 ;
  measurableI : setI_closed measurable;
  semi_measurableD : semi_setD_closed measurable }.
```




```
HB.structure Definition SemiRingOfSets d :=
  {T of Pointed T & isSemiRingOfSets d T}.

HB.mixin Record SemiRingOfSets_isRingOfSets d T of SemiRingOfSets d T :=
  { measurableU : @setU_closed T measurable }.

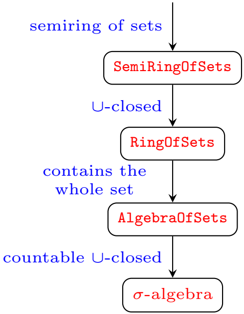
HB.structure Definition RingOfSets d :=
  {T of SemiRingOfSets d T & SemiRingOfSets_isRingOfSets d T }.

HB.mixin Record RingOfSets_isAlgebraOfSets d T of RingOfSets d T :=
  { measurableT : measurable [set: T] }.

HB.structure Definition AlgebraOfSets d :=
  {T of RingOfSets d T & RingOfSets_isAlgebraOfSets d T }.
```

Measurable structures in MathComp-Analysis

Formalization using Hierarchy-Builder [Cohen et al., 2020] to construct the Lebesgue measure by extension [Affeldt and Cohen, 2023]




```
HB.mixin Record isSemiRingOfSets (d : measure_display) T := {
  measurable : set (set T) ;
  measurable0 : measurable set0 ;
  measurableI : setI_closed measurable;
  semi_measurableD : semi_setD_closed measurable }.

HB.structure Definition SemiRingOfSets d :=
  {T of Pointed T & isSemiRingOfSets d T}.

HB.mixin Record SemiRingOfSets_isRingOfSets d T of SemiRingOfSets d T :=
  { measurableU : @setU_closed T measurable }.

HB.structure Definition RingOfSets d :=
  {T of SemiRingOfSets d T & SemiRingOfSets_isRingOfSets d T }.

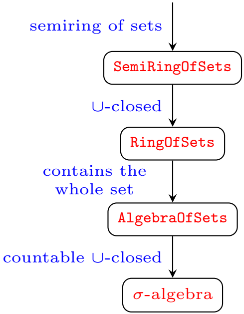
HB.mixin Record RingOfSets_isAlgebraOfSets d T of RingOfSets d T :=
  { measurableT : measurable [set: T] }.

HB.structure Definition AlgebraOfSets d :=
  {T of RingOfSets d T & RingOfSets_isAlgebraOfSets d T }.

HB.mixin Record hasMeasurableCountableUnion d T of SemiRingOfSets d T := {
  bigcupT_measurable : forall F : (set T)^nat,
    (forall i, measurable (F i)) -> measurable (\bigcup_i (F i)) }.
```

Measurable structures in MathComp-Analysis

Formalization using Hierarchy-Builder [Cohen et al., 2020] to construct the Lebesgue measure by extension [Affeldt and Cohen, 2023]




```
HB.mixin Record isSemiRingOfSets (d : measure_display) T := {
  measurable : set (set T) ;
  measurable0 : measurable set0 ;
  measurableI : setI_closed measurable;
  semi_measurableD : semi_setD_closed measurable }.

HB.structure Definition SemiRingOfSets d :=
  {T of Pointed T & isSemiRingOfSets d T}.

HB.mixin Record SemiRingOfSets_isRingOfSets d T of SemiRingOfSets d T :=
  { measurableU : @setU_closed T measurable }.

HB.structure Definition RingOfSets d :=
  {T of SemiRingOfSets d T & SemiRingOfSets_isRingOfSets d T }.

HB.mixin Record RingOfSets_isAlgebraOfSets d T of RingOfSets d T :=
  { measurableT : measurable [set: T] }.

HB.structure Definition AlgebraOfSets d :=
  {T of RingOfSets d T & RingOfSets_isAlgebraOfSets d T }.

HB.mixin Record hasMeasurableCountableUnion d T of SemiRingOfSets d T := {
  bigcupT_measurable : forall F : (set T)^nat,
    (forall i, measurable (F i)) -> measurable (\bigcup_i (F i)) }.

HB.structure Definition Measurable d :=
  {T of AlgebraOfSets d T & hasMeasurableCountableUnion d T }.
```

Extensibility of hierarchies using Hierarchy-Builder

In fact, there is yet another measurable structure:

	semi ring of sets	ring of sets	algebra of sets	$\boxed{\sigma\text{-ring}}$	σ-algebra
contains $\emptyset$	✓	✓	✓	✓	✓
closed under $\cap$	✓	✓	✓	✓	✓
closed under "semi-difference"	✓	✓	✓	✓	✓
closed under $\cup$		✓	✓	✓	✓
contains the whole set			✓		✓
closed under countable union				✓	✓

σ-rings are used in [Halmos, 1974] and enable generalizations

Extensibility of hierarchies using HIERARCHY-BUILDER

In fact, there is yet another measurable structure:

	semi ring of sets	ring of sets	algebra of sets	$\boxed{\sigma\text{-ring}}$	σ-algebra
contains $\emptyset$	✓	✓	✓	✓	✓
closed under $\cap$	✓	✓	✓	✓	✓
closed under "semi-difference"	✓	✓	✓	✓	✓
closed under $\cup$		✓	✓	✓	✓
contains the whole set			✓		✓
closed under countable union				✓	✓

σ-rings are used in [Halmos, 1974] and enable generalizations

With HIERARCHY-BUILDER, adding σ-rings is simple:

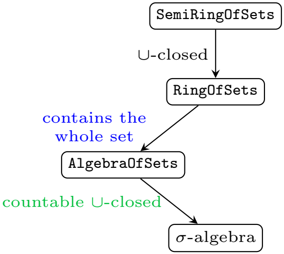
Extensibility of hierarchies using HIERARCHY-BUILDER

In fact, there is yet another measurable structure:

	semi ring of sets	ring of sets	algebra of sets	$\boxed{\sigma\text{-ring}}$	σ-algebra
contains $\emptyset$	✓	✓	✓	✓	✓
closed under $\cap$	✓	✓	✓	✓	✓
closed under "semi-difference"	✓	✓	✓	✓	✓
closed under $\cup$		✓	✓	✓	✓
contains the whole set			✓		✓
closed under countable union				✓	✓

σ-rings are used in [Halmos, 1974] and enable generalizations

With HIERARCHY-BUILDER, adding σ-rings is simple:



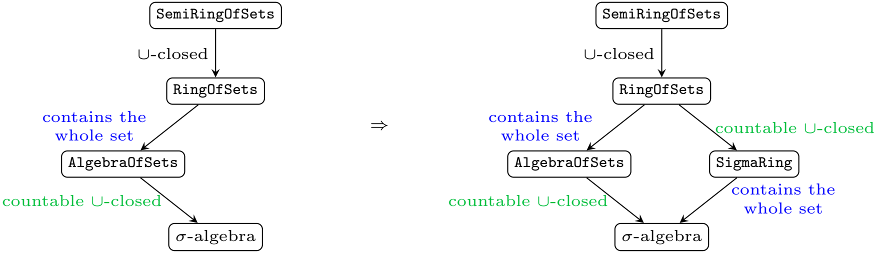
Extensibility of hierarchies using Hierarchy-Builder

In fact, there is yet another measurable structure:

	semi ring of sets	ring of sets	algebra of sets	$\boxed{\sigma\text{-ring}}$	σ-algebra
contains $\emptyset$	✓	✓	✓	✓	✓
closed under $\cap$	✓	✓	✓	✓	✓
closed under "semi-difference"	✓	✓	✓	✓	✓
closed under $\cup$		✓	✓	✓	✓
contains the whole set			✓		✓
closed under countable union				✓	✓

σ-rings are used in [Halmos, 1974] and enable generalizations

With Hierarchy-Builder, adding σ-rings is simple:



Measures in MathComp-Analysis

A (non-negative) *measure* is a function $\mu : \Sigma_X \to [0, \infty]$ such that

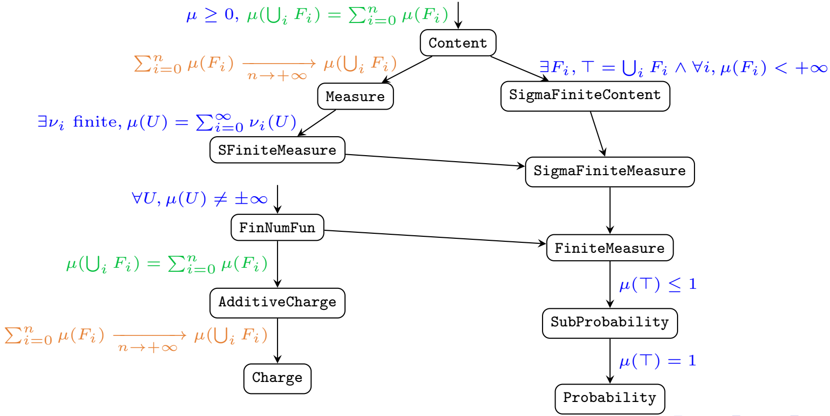
- $\mu(\emptyset) = 0$ and
- $\sum_{i=0}^{n} \mu(F_i) \xrightarrow[n\to+\infty]{} \mu(\bigcup_i F_i)$ for pairwise-disjoint measurable sets F_i

Measures in MathComp-Analysis

A (non-negative) *measure* is a function $\mu : \Sigma_X \to [0, \infty]$ such that

- $\mu(\emptyset) = 0$ and
- $\sum_{i=0}^{n} \mu(F_i) \xrightarrow[n \to +\infty]{} \mu(\bigcup_i F_i)$ for pairwise-disjoint measurable sets F_i

Measure-like functions also form a hierarchy [Affeldt and Cohen, 2023, Ishiguro and Affeldt, 2024]:



Kernels in MathComp-Analysis

A *kernel* $X \rightsquigarrow Y$ is a function $k : X \to \underbrace{\Sigma_Y \to [0, \infty]}_{\text{measure}}$ such that

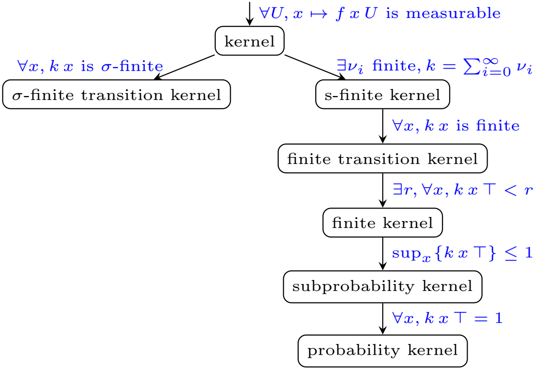
$\forall U \in \Sigma_Y,\ x \mapsto k\,x\,U$ is a *measurable function*.

Kernels in MathComp-Analysis

A *kernel* $X \rightsquigarrow Y$ is a function $k : X \to \underbrace{\Sigma_Y \to [0, \infty]}_{\text{measure}}$ such that

$\forall U \in \Sigma_Y,\ x \mapsto k\,x\,U$ is a *measurable function*.

Kernels also form a hierarchy [Affeldt et al., 2025b]:



The key property of s-finite kernels

Stability by composition [Staton, 2017]:

$$\begin{tikzcd}[column sep=small] l : X \overset{\text{s-fin}}{\rightsquigarrow} Y \arrow[dr] & & k : X \times Y \overset{\text{s-fin}}{\rightsquigarrow} Z \arrow[dl] \\ & l \boxed{;} k : X \overset{\text{s-fin}}{\rightsquigarrow} Z & \end{tikzcd}$$

$$\overset{\text{def}}{=} \lambda x\, U.\, \int_y k\,(x, y)\, U\, \mathbf{d} l\; x$$

The key property of s-finite kernels

Stability by composition [Staton, 2017]:

$$
\begin{tikzcd}
{\color{red}l} : X \overset{\text{s-fin}}{\rightsquigarrow} Y \arrow[dr] & & {\color{blue}k} : X \times Y \overset{\text{s-fin}}{\rightsquigarrow} Z \arrow[dl] \\
& {\color{red}l} \boxed{;} {\color{blue}k} : X \overset{\text{s-fin}}{\rightsquigarrow} Z &
\end{tikzcd}
$$

$$
\overset{\text{def}}{=} \lambda x\, U.\ \int_y {\color{blue}k}\,(x, y)\, U\, \mathbf{d}{\color{red}l}\; x
$$

- Direct definition using the Lebesgue integral in MathComp-Analysis:

```
Definition kcomp l k x U := \int[l x]_y k (x, y) U.
```

The key property of s-finite kernels

Stability by composition [Staton, 2017]:

$$
\begin{tikzcd}
\color{red}{l} : X \overset{\text{s-fin}}{\rightsquigarrow} Y \arrow[dr] & & \color{blue}{k} : X \times Y \overset{\text{s-fin}}{\rightsquigarrow} Z \arrow[dl] \\
& \color{red}{l} \boxed{;} \color{blue}{k} : X \overset{\text{s-fin}}{\rightsquigarrow} Z &
\end{tikzcd}
$$

$$\overset{\text{def}}{=} \lambda x\, U.\ \int_y \color{blue}{k}\,(x, y)\, U\, \mathbf{d}\color{red}{l}\, x$$

- Direct definition using the Lebesgue integral in MathComp-Analysis:

```
Definition kcomp l k x U := \int[l x]_y k (x, y) U.
```

- The difficulty of the stability proof is to establish measurability [Affeldt et al., 2025b, Sect. 5]

Outline

Overview of the formalization of the Lebesgue integral

Overview of the formalization of the Lebesgue integral

- First FTC for Lebesgue integration [Affeldt and Stone, 2024]:
  - For f integrable on $\mathbb{R}$, define $F(x) \stackrel{\text{def}}{=} \int_{-\infty}^{x} f(t)\mathbf{d}t$.
    Then $\quad F$ is differentiable $\quad$ and $\quad F'(x) = f(x) \quad$ (a.e.).

Overview of the formalization of the Lebesgue integral

- First FTC for Lebesgue integration [Affeldt and Stone, 2024]:
  - For f integrable on $\mathbb{R}$, define $F(x) \stackrel{\text{def}}{=} \int_{-\infty}^{x} f(t)\mathbf{d}t$.
    Then F is differentiable and $F'(x) = f(x)$ (a.e.).
- Second FTC for Lebesgue integration for continuous functions [Affeldt et al., 2025c]:
  - For f with antiderivative F in $]a, b[$, f continuous *within* $[a, b]$, F differentiable in $]a, b[$ with $F(x) \xrightarrow[x \to a^+]{} F(a)$ and $F(x) \xrightarrow[x \to b^-]{} F(b)$, we have:

$$\int_{x \in [a,b]} f(x)\mathbf{d}\mu = F(b) - F(a).$$

Overview of the formalization of the Lebesgue integral

- First FTC for Lebesgue integration [Affeldt and Stone, 2024]:
  - For f integrable on $\mathbb{R}$, define $F(x) \stackrel{\text{def}}{=} \int_{-\infty}^{x} f(t)\mathbf{d}t$.
    Then F is differentiable and $F'(x) = f(x)$ (a.e.).
- Second FTC for Lebesgue integration for continuous functions [Affeldt et al., 2025c]:
  - For f with antiderivative F in $]a, b[$, f continuous *within* $[a, b]$, F differentiable in $]a, b[$ with $F(x) \xrightarrow[x \to a^+]{} F(a)$ and $F(x) \xrightarrow[x \to b^-]{} F(b)$, we have:

$$\int_{x \in [a,b]} f(x)\mathbf{d}\mu = F(b) - F(a).$$

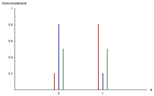
- From the FTC, follow
  - integration by parts
  - integration by substitution (a.k.a. change of variables)
  - continuity/differentiation under the integral sign

  over bounded and unbounded intervals [Affeldt et al., 2025c]

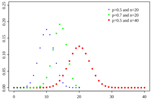
Outline

Available probability distributions

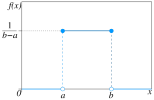
`probability.v`



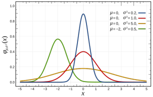
Bernoulli



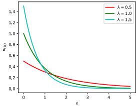
Binomial



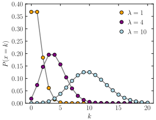
Uniform



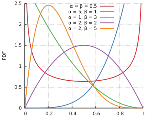
Normal



Exponential



Poisson



Beta

(Pictures are taken from Wikipedia)

Example: the Beta distribution

Example: the Beta distribution

The Beta *probability density function* ($a, b > 0$ and $t \in [0,1]$):

$$\lambda t \mapsto \frac{t^{a-1}(1-t)^{b-1}}{\int_{u\in[0,1]} u^{a-1}(1-u)^{b-1}\mathbf{d}\mu} = \frac{t^{a-1}(1-t)^{b-1}}{\beta(a,b)}$$

Example: the Beta distribution

The Beta *probability density function* ($a, b > 0$ and $t \in [0,1]$):

$$\lambda t \mapsto \frac{t^{a-1}(1-t)^{b-1}}{\int_{u \in [0,1]} u^{a-1}(1-u)^{b-1} \mathbf{d}\mu} = \frac{t^{a-1}(1-t)^{b-1}}{\beta(a,b)}$$

Formalization of the Beta probability density function `beta_pdf`:

- $\lambda t. t^{a-1}(1-t)^{b-1}|_{[0,1]}$ encoded as

  ```
  XMonemX01 a b = (fun t => t ^+ a.-1 * (1 - t) ^+ b.-1 \_ `[0, 1])
  ```
- $\beta(a,b)$ encoded as $\int_x$ XMonemX01 $a\, b\, x\, \mathbf{d}\mu$
- `Definition beta_pdf a b t := XMonemX01 a b t / β a b.`

Example: the Beta distribution

The Beta *probability density function* $(a, b > 0$ and $t \in [0, 1])$:

$$\lambda t \mapsto \frac{t^{a-1}(1-t)^{b-1}}{\int_{u\in[0,1]} u^{a-1}(1-u)^{b-1}\mathbf{d}\mu} = \frac{t^{a-1}(1-t)^{b-1}}{\beta(a,b)}$$

Formalization of the Beta probability density function `beta_pdf`:

- $\lambda t.t^{a-1}(1-t)^{b-1}|_{[0,1]}$ encoded as

  ```
  XMonemX01 a b = (fun t => t ^+ a.-1 * (1 - t) ^+ b.-1 \_ `[0, 1])
  ```

- $\beta(a, b)$ encoded as $\int_x$ `XMonemX01` $a\, b\, x\, \mathbf{d}\mu$
- `Definition beta_pdf a b t := XMonemX01 a b t / β a b.`

Formalization of the probability measure `beta_prob a b`:

$$U \mapsto \int_{t\in U} \texttt{beta\_pdf}\, a\, b\, t\, \mathbf{d}\mu$$

Properties of the Beta distribution and of the β function

- Integration w.r.t. probability measure:

```
Lemma integral_beta_prob a b f U :
      measurable U -> measurable_fun U f ->
      \int[beta_prob a b]_(x in U) `|f x| < +oo ->
  \int[beta_prob a b]_(x in U) f x =
  \int[mu]_(x in U) (f x * (beta_pdf a b x)%:E).
```

(using Radon-Nikodým's change of variables [Ishiguro and Affeldt, 2024])

Properties of the Beta distribution and of the β function

- Integration w.r.t. probability measure:

```
Lemma integral_beta_prob a b f U :
      measurable U -> measurable_fun U f ->
      \int[beta_prob a b]_(x in U) `|f x| < +oo ->
  \int[beta_prob a b]_(x in U) f x =
  \int[mu]_(x in U) (f x * (beta_pdf a b x)%:E).
```

  (using Radon-Nikodým's change of variables [Ishiguro and Affeldt, 2024])

- Symmetry of the β function:

```
Lemma betafun_sym (a b : nat) : β a b = β b a.
```

  (using integration by substitution)

Properties of the Beta distribution and of the β function

- Integration w.r.t. probability measure:

```
Lemma integral_beta_prob a b f U :
      measurable U -> measurable_fun U f ->
      \int[beta_prob a b]_(x in U) `|f x| < +oo ->
  \int[beta_prob a b]_(x in U) f x =
  \int[mu]_(x in U) (f x * (beta_pdf a b x)%:E).
```

  (using Radon-Nikodým's change of variables [Ishiguro and Affeldt, 2024])

- Symmetry of the β function:

```
Lemma betafun_sym (a b : nat) : β a b = β b a.
```

  (using integration by substitution)

- Relation with the factorial $(a, b > 0)$:

$$\beta\, a\, b = \frac{(a-1)!(b-1)!}{((a+b)-1)!}$$

  (by induction, symmetry of β, and integration by parts)

Outline

Motivation: Eddy's table game [Eddy, 2004]

- game with two players (Alice and Bob) in a casino
- the casino rolls a ball to determine p (and hides it)
- Alice has won 5 out of 8 games

- the casino repeatedly rolls balls until a player has 6 points
- Alice bets she will win
- what is her probability to win?

Encoding as a probabilistic program [Shan, 2018b]:

$$
\begin{aligned}
&\text{normalize}(\\
&\text{let } p := \text{sample}\,(\text{uniform}(0,1)) \text{ in}\\
&\text{let } x := \text{sample}\,(\text{binomial}(8,p)) \text{ in}\\
&\text{let } \_ := \text{guard}(x = 5) \text{ in}\\
&\text{let } y := \text{sample}\,(\text{binomial}(3,p)) \text{ in}\\
&\text{return}(1 \leq y))
\end{aligned}
$$

Intuitive semantics

- Intuitively, each instruction is an s-finite kernel:
  - sample (...) is a probability kernel
  - normalize (...) is a probability kernel
  - score (f k) is an s-finite kernel
    - its meaning: we observe k from the distribution with density f
    - for example, we observe $k = 4$ with the density $f_r(k) = \frac{r^k}{k!}e^{-r}$ (probability mass function of the Poisson distribution)
    - guard$(x = n) \stackrel{\text{def}}{=}$ if $x = n$ then $\texttt{tt}$ else score (0).
  - the semantics of let $x := e_1$ in e_2 is kernel composition
    - because it is stable for s-finite kernels (as we saw Slide 18)

Shan's proof of Eddy's table game

Proof represented by a sequence of program transformations [Shan, 2018b, Shan, 2018a]:

0

$$\begin{array}{l} \text{normalize}(\\ \text{let } p := \text{sample}\,(\text{uniform}(0,1)) \text{ in} \\ \text{let } x := \text{sample}\,(\text{binomial}(8,p)) \text{ in} \\ \text{let } \_ := \text{guard}(x = 5) \text{ in} \\ \text{let } y := \text{sample}\,(\text{binomial}(3,p)) \text{ in} \\ \text{return}(1 \leq y)) \end{array}$$

$\rightarrow$

1

$$\begin{array}{l} \text{normalize}(\\ \text{let } p := \text{sample}\,(\text{uniform}(0,1)) \text{ in} \\ \text{let } x := \text{sample}\,(\text{binomial}(8,p)) \text{ in} \\ \text{let } \_ := \text{guard}(x = 5) \text{ in} \\ \text{sample}\left(\text{bernoulli}\left(1-(1-p)^3\right)\right)) \end{array}$$

$\downarrow$

2

$$\begin{array}{l} \text{normalize}(\\ \text{let } p := \text{sample}\,(\text{uniform}(0,1)) \text{ in} \\ \text{let } \_ := \text{score}\left(56p^5(1-p)^3\right) \text{ in} \\ \text{sample}\left(\text{bernoulli}\left(1-(1-p)^3\right)\right)) \end{array}$$

Slide 40 $\leftarrow$

3

$$\begin{array}{l} \text{normalize}(\\ \text{let } \_ := \text{score}\left(\frac{1}{9}\right) \text{ in} \\ \text{let } p := \text{sample}\,(\text{beta}(6,4)) \text{ in} \\ \text{sample}\left(\text{bernoulli}\left(1-(1-p)^3\right)\right)) \end{array}$$

$\downarrow$

4

$$\begin{array}{l} \text{normalize}(\\ \text{let } \_ := \text{score}\left(\frac{1}{9}\right) \text{ in} \\ \text{sample}\left(\text{bernoulli}\left(\frac{10}{11}\right)\right)) \end{array}$$

$\rightarrow$

5

$$\text{normalize}\left(\text{sample}\left(\text{bernoulli}\left(\frac{10}{11}\right)\right)\right)$$

sfPPL: a first-order probabilistic language

[2] Measurability is not always easy to establish, e.g., $m \mapsto$ `normal_prob` $m\ s$

sfPPL: a first-order probabilistic language

- Types:

$$\mathbf{A} \quad ::= \quad \mathbf{U} \mid \mathbf{B} \mid \mathbf{N} \mid \mathbf{R} \mid P(\mathbf{A}) \mid \mathbf{A}_1 \times \mathbf{A}_2$$

[2] Measurability is not always easy to establish, e.g., $m \mapsto$ `normal_prob` $m\ s$

sfPPL: a first-order probabilistic language

- Types:

$$\mathbf{A} \quad ::= \quad \mathbf{U} \mid \mathbf{B} \mid \mathbf{N} \mid \mathbf{R} \mid P(\mathbf{A}) \mid \mathbf{A}_1 \times \mathbf{A}_2$$

- Expressions (f is a measurable function[2]):

$$\begin{array}{rcl} e & ::= & \texttt{tt} \mid b \mid n \mid r \mid f(e_1, \ldots, e_n) \mid (e_1, e_2) \mid \pi_1(e) \mid \pi_2(e) \\ & & \text{if}\, e \,\text{then}\, e_1 \,\text{else}\, e_2 \mid x \mid \text{return}(e) \mid \text{let } x := e_1 \text{ in } e_2 \mid \\ & & \text{sample}\,(e) \mid \text{score}\,(e) \mid \text{normalize}\,(e) \end{array}$$

[2] Measurability is not always easy to establish, e.g., $m \mapsto \texttt{normal\_prob}\, m\, s$

sfPPL: a first-order probabilistic language

- Types:

$$\mathbf{A} \quad ::= \quad \mathbf{U} \mid \mathbf{B} \mid \mathbf{N} \mid \mathbf{R} \mid P(\mathbf{A}) \mid \mathbf{A}_1 \times \mathbf{A}_2$$

- Expressions (f is a measurable function[2]):

$$\begin{aligned} e \quad ::= \quad & \mathtt{tt} \mid b \mid n \mid r \mid f(e_1, \ldots, e_n) \mid (e_1, e_2) \mid \pi_1(e) \mid \pi_2(e) \\ & \text{if}\, e \,\text{then}\, e_1 \,\text{else}\, e_2 \mid x \mid \text{return}(e) \mid \text{let } x := e_1 \text{ in } e_2 \mid \\ & \text{sample}\,(e) \mid \text{score}\,(e) \mid \text{normalize}\,(e) \end{aligned}$$

- Type contexts:

$$\Gamma ::= (x_1 : \mathbf{A}_1; \ldots; x_n : \mathbf{A}_n)$$

[2] Measurability is not always easy to establish, e.g., $m \mapsto \mathtt{normal\_prob}\; m\; s$

sfPPL: a first-order probabilistic language

- Types:

$$\mathbf{A} \quad ::= \quad \mathbf{U} \mid \mathbf{B} \mid \mathbf{N} \mid \mathbf{R} \mid P(\mathbf{A}) \mid \mathbf{A}_1 \times \mathbf{A}_2$$

- Expressions (f is a measurable function[2]):

$$\begin{array}{rcl} e & ::= & \mathtt{tt} \mid b \mid n \mid r \mid f(e_1, \ldots, e_n) \mid (e_1, e_2) \mid \pi_1(e) \mid \pi_2(e) \\ & & \mathrm{if}\, e\, \mathrm{then}\, e_1\, \mathrm{else}\, e_2 \mid x \mid \mathrm{return}(e) \mid \mathrm{let}\ x := e_1\ \mathrm{in}\ e_2 \mid \\ & & \mathsf{sample}\,(e) \mid \mathsf{score}\,(e) \mid \mathsf{normalize}\,(e) \end{array}$$

- Type contexts:

$$\Gamma ::= (x_1 : \mathbf{A}_1; \ldots; x_n : \mathbf{A}_n)$$

- Type judgments:
  - deterministic expressions: $\Gamma \vdash_{\mathsf{D}} e : \mathbf{A}$
  - probabilistic expressions: $\Gamma \vdash_{\mathsf{P}} e : \mathbf{A}$

[2] Measurability is not always easy to establish, e.g., $m \mapsto$ `normal_prob` $m\ s$

sfPPL: a first-order probabilistic language

- Types:

$$\mathbf{A} \quad ::= \quad \mathbf{U} \mid \mathbf{B} \mid \mathbf{N} \mid \mathbf{R} \mid P(\mathbf{A}) \mid \mathbf{A}_1 \times \mathbf{A}_2$$

- Expressions (f is a measurable function[2]):

$$\begin{array}{rcl} e & ::= & \texttt{tt} \mid b \mid n \mid r \mid f(e_1, \ldots, e_n) \mid (e_1, e_2) \mid \pi_1(e) \mid \pi_2(e) \\ & & \text{if}\, e \,\text{then}\, e_1 \,\text{else}\, e_2 \mid x \mid \text{return}(e) \mid \text{let } x := e_1 \text{ in } e_2 \mid \\ & & \text{sample}\,(e) \mid \text{score}\,(e) \mid \text{normalize}\,(e) \end{array}$$

- Type contexts:

$$\Gamma ::= (x_1 : \mathbf{A}_1; \ldots; x_n : \mathbf{A}_n)$$

- Type judgments:
  - deterministic expressions: $\Gamma \vdash_{\mathsf{D}} e : \mathbf{A}$
  - probabilistic expressions: $\Gamma \vdash_{\mathsf{P}} e : \mathbf{A}$

Examples:

$$\frac{\Gamma \vdash_{\mathsf{D}} e : P(\mathbf{A})}{\Gamma \vdash_{\mathsf{P}} \text{sample}\,(e) : \mathbf{A}} \qquad \frac{\Gamma \vdash_{\mathsf{D}} e : \mathbf{R}}{\Gamma \vdash_{\mathsf{P}} \text{score}\,(e) : \mathbf{U}}$$

$$\frac{\Gamma \vdash_{\mathsf{P}} e : \mathbf{A}}{\Gamma \vdash_{\mathsf{D}} \text{normalize}\,(e) : P(\mathbf{A})}$$

[2] Measurability is not always easy to establish, e.g., $m \mapsto$ `normal_prob` $m\ s$

Formal syntax for sfPPL (excerpt)

Intrinsically-typed syntax: dependent inductive type with 3 indices

1. `flag`: deterministic or probabilistic
2. `typ`: the type of the expression
3. `ctx`: association list variable↔type

```
Inductive exp : flag -> ctx -> typ -> Type :=
```

Formal syntax for sfPPL (excerpt)

Intrinsically-typed syntax: dependent inductive type with 3 indices

1. `flag`: deterministic or probabilistic
2. `typ`: the type of the expression
3. `ctx`: association list variable↔type

```
Inductive exp : flag -> ctx -> typ -> Type :=
(* real constants are deterministic *)
| exp_real g : R -> exp D g Real
(* addition of real numbers *)
| exp_add g : exp D g Real -> exp D g Real -> exp D g Real
(* a Bernoulli measure of some real parameter *)
| exp_bernoulli g : exp D g Real -> exp D g (Prob Bool)
(* Poisson pmf *)
| exp_poisson g : nat -> exp D g Real -> exp D g Real
(* the type of a variable depends on the context *)
| exp_var g str t : t = lookup Unit g str -> exp D g t
(* the context is extended inside a let expression *)
| exp_letin g t1 t2 str : exp P g t1 -> exp P ((str, t1) :: g) t2 ->
    exp P g t2
(* sampling from a probability distribution *)
| exp_sample g t : exp D g (Prob t) -> exp P g t
(* normalization *)
| exp_normalize g t : exp P g t -> exp D g (Prob t)
(* scoring *)
| exp_score g : exp D g Real -> exp P g Unit
```

Issue #1: unification order

Let us encode: `let x := 1 in let y := 2 in x + y`
using “concrete strings” for variable identifiers (`"x"`, `"y"`)

```
Fail Example letin_add
    : exp [::] _ :=
  exp_letin "x" (exp_real 1)

  (exp_letin "y" (exp_real 2)

   (exp_add

    (exp_var "x"
      erefl)
    (exp_var "y"
      erefl))).
```

Issue #1: unification order

Let us encode: `let x := 1 in let y := 2 in x + y`
using "concrete strings" for variable identifiers (`"x"`, `"y"`)

```
Fail Example letin_add
    : exp [::] _ :=
  exp_letin "x" (exp_real 1)

  (exp_letin "y" (exp_real 2)

   (exp_add

    (exp_var "x"
      erefl)
    (exp_var "y"
      erefl))).
```

```
"exp_var "x" (erefl (lookup Unit ?g1 "x"))" has type
"exp ?g1 (lookup Unit ?g1 "x")" while it is expected to
have type "exp ?g1 Real".
```

Issue #1: unification order

Let us encode: `let x := 1 in let y := 2 in x + y`
using "concrete strings" for variable identifiers (`"x"`, `"y"`)

```
Fail Example letin_add
    : exp [::] _ :=
  exp_letin "x" (exp_real 1)

  (exp_letin "y" (exp_real 2)

   (exp_add

    (exp_var "x"
      erefl)
    (exp_var "y"
      erefl))).
```

```
(* same program with explicit variables *)

@exp_letin [::] _ "x" (exp_real 1)

(@exp_letin ?g0 _ "y" (exp_real 2)

 (@exp_add ?g1 Real

  (@exp_var ?g1 _ "x"
    @erefl (lookup Unit ?g1 "x"))
  (@exp_var ?g1 _ "y"
    @erefl (lookup Unit ?g1 "y")))).
```

```
"exp_var "x" (erefl (lookup Unit ?g1 "x"))" has type
"exp ?g1 (lookup Unit ?g1 "x")" while it is expected to
have type "exp ?g1 Real".
```

Solution #1: bidirectional hints [The Rocq Development Team, 2025a]

Special annotation & to direct unification:

```
Arguments exp_add {g} &.          (* unify preferentially g first *)
Arguments exp_letin {g} & {A B}.
```

As a result:

1. `g1` unifies to `[:: ("y", Real), ("x", Real)]`
2. `lookup Unit g1 "x"` evaluates to `Real`

`let x := 1 in let y := 2 in x + y` ⇒

```
Example letin_add : exp [::] _ :=
  exp_letin "x" (exp_real 1)
  (exp_letin "y" (exp_real 2)
   (exp_add
    (exp_var "x" erefl)
    (exp_var "y" erefl))).
```

Issue #2: universally quantified strings

Encoding

```
let x := 1 in let y := 2 in x + y
```

with universally quantified strings for variable identifiers fails:

```
Fail Example
letin_add (x y : string) (xy : x != y) (yx : y != x) : exp [::] _ :=
          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  exp_letin x (exp_real 1)
  (exp_letin y (exp_real 2)
   (exp_add
    (exp_var x erefl)
    (exp_var y erefl))).
```

```
cannot unify "lookup Unit [:: (y, Real); (x, Real)] x"
and "Real".
```

Solution # 2: unification using canonical structures

Overview

Direct application of [Gonthier et al., 2013b]
Goal: a proof of

$$\boxed{\mathbb{R} = \texttt{lookup}\ ((y, \mathbb{R}) :: (x, \mathbb{R}) :: [\,])\ x}$$

1. We introduce the following structure:
   $\texttt{Record find}\ x\ t\ \texttt{:=}\ \texttt{Find}\ \Gamma\ \underbrace{(t = \texttt{lookup}\ \Gamma\ x)}_{\texttt{ctx\_prf}}$
2. We look for an instance P_0 : $\texttt{find}\ x\ \mathbb{R}$ such that
   - the 1st projection is $\Gamma(P_0) = (y, \mathbb{R}) :: (x, \mathbb{R}) :: [\,]$,
   - the 2nd projection `ctx_prf` provides the desired proof

Solution # 2: unification using canonical structures

Details

Goal: a proof of $\boxed{\mathbb{R} = \texttt{lookup}\ ((y, \mathbb{R}) :: (x, \mathbb{R}) :: [])\ x}$

Unification using canonical structures:

1. We look for P_0 : $\texttt{find}\ x\ \mathbb{R}$ such that
   $\Gamma(P_0) = (y, \mathbb{R}) :: \Gamma(P_1)$ for some P_1 : $\texttt{find}\ x\ \mathbb{R}$ with $x \neq y$

Solution # 2: unification using canonical structures

Details

Goal: a proof of $\boxed{\mathbb{R} = \texttt{lookup}\ ((y, \mathbb{R}) :: (x, \mathbb{R}) :: [])\ x}$

Unification using canonical structures:

1. We look for P_0 : $\texttt{find}\ x\ \mathbb{R}$ such that $\Gamma(P_0) = (y, \mathbb{R}) :: \Gamma(P_1)$ for some P_1 : $\texttt{find}\ x\ \mathbb{R}$ with $x \neq y$
2. A structure P_1 : $\texttt{find}\ x\ \mathbb{R}$ is simply such that
   - $\Gamma(P_1) = (x, \mathbb{R}) :: []$

Solution # 2: unification using canonical structures

Details

Goal: a proof of $\boxed{\mathbb{R} = \texttt{lookup}\ ((y, \mathbb{R}) :: (x, \mathbb{R}) :: [])\ x}$

Unification using canonical structures:

1. We look for $P_0 : \texttt{find}\ x\ \mathbb{R}$ such that $\Gamma(P_0) = (y, \mathbb{R}) :: \Gamma(P_1)$ for some $P_1 : \texttt{find}\ x\ \mathbb{R}$ with $x \neq y$
2. A structure $P_1 : \texttt{find}\ x\ \mathbb{R}$ is simply such that
   - $\Gamma(P_1) = (x, \mathbb{R}) :: []$
3. There is a <u>canonical</u> way to construct P_1
   - The instance that puts $(x, \mathbb{R})$ at the head (say, "$\texttt{found}\ x\ \mathbb{R}\ []$")

Solution # 2: unification using canonical structures

Details

Goal: a proof of $\boxed{\mathbb{R} = \texttt{lookup}\ ((y, \mathbb{R}) :: (x, \mathbb{R}) :: [])\ x}$

Unification using canonical structures:

1. We look for P_0 : $\texttt{find}\ x\ \mathbb{R}$ such that
   $\Gamma(P_0) = (y, \mathbb{R}) :: \Gamma(P_1)$ for some P_1 : $\texttt{find}\ x\ \mathbb{R}$ with $x \neq y$
2. A structure P_1 : $\texttt{find}\ x\ \mathbb{R}$ is simply such that
   - $\Gamma(P_1) = (x, \mathbb{R}) :: []$
3. There is a canonical way to construct P_1
   - The instance that puts $(x, \mathbb{R})$ at the head
     (say, "$\texttt{found}\ x\ \mathbb{R}\ []$")
4. Given P_1, there is a canonical way to build P_0
   - The instance that puts $(y, \mathbb{R})$ at the head providing $x \neq y$
     (say, "$\texttt{recurse}\ x\ \mathbb{R}\ y\ \mathbb{R}\ \{\mathrm{H} : \texttt{infer}\ (\mathrm{y}\ != \mathrm{x})\}\ (f : \texttt{find}\ x\ t)$")

Solution # 2: unification using canonical structures

Details

Goal: a proof of $\boxed{\mathbb{R} = \texttt{lookup}\ ((y, \mathbb{R}) :: (x, \mathbb{R}) :: [])\ x}$

Unification using canonical structures:

1. We look for P_0 : `find` x $\mathbb{R}$ such that $\Gamma(P_0) = (y, \mathbb{R}) :: \Gamma(P_1)$ for some P_1 : `find` x $\mathbb{R}$ with $x \neq y$
2. A structure P_1 : `find` x $\mathbb{R}$ is simply such that
   - $\Gamma(P_1) = (x, \mathbb{R}) :: []$
3. There is a canonical way to construct P_1
   - The instance that puts $(x, \mathbb{R})$ at the head (say, "`found` x $\mathbb{R}$ []")
4. Given P_1, there is a canonical way to build P_0
   - The instance that puts $(y, \mathbb{R})$ at the head providing $x \neq y$ (say, "`recurse` x $\mathbb{R}$ y $\mathbb{R}$ {H : `infer` (y != x)} (f : `find` x t)")
5. To control the order, there are "tags" that Rocq unfolds when unification fails

Solution # 2: unification using canonical structures

To cover the case of universally quantified strings, we ask Rocq to look for `find` structures using:

```
Definition exp_var' str {t : typ} (f : find str t) :=
```

$$\quad\texttt{@exp\_var}\quad \underbrace{\Gamma(\texttt{f})}_{\text{1st projection}}\quad \texttt{t str}\ \underbrace{(\texttt{ctx\_prf f})}_{\text{2nd projection}}.$$

Solution # 2: unification using canonical structures

To cover the case of universally quantified strings, we ask Rocq to look for `find` structures using:

```
Definition exp_var' str {t : typ} (f : find str t) :=
```

$$\texttt{@exp\_var}\quad \underbrace{\Gamma(\texttt{f})}_{\text{1st projection}}\quad \texttt{t str}\ \underbrace{(\texttt{ctx\_prf f})}_{\text{2nd projection}}.$$

```
let x := 1 in let y := 2 in x + y
```

$$\Downarrow$$

```
Example letin_add (x y : string)
    (xy : infer (x != y)) (yx : infer (y != x)) : exp [::] _ :=
  exp_letin x (exp_real 1)
  (exp_letin y (exp_real 2)
   (exp_add (exp_var' x _) (exp_var' y _))).
```

Formal syntax of sfPPL applied to Eddy's table game

Using Rocq's *custom entries* [The Rocq Development Team, 2025b]:

```
Definition guard {g} str n
  : @exp R P [:: (str, _) ; g] _ :=
[if #{str} == {n}:N then return TT
                    else Score {0}:R].
```

normalize(
let $p :=$ sample (uniform$(0,1)$) in
let $x :=$ sample (binomial$(8,p)$) in
let _ := guard$(x = 5)$ in
let $y :=$ sample (binomial$(3,p)$) in
return$(y \geq 1)$)

```
Definition table0 : @exp R _ [::] _ :=
 [Normalize
  let "p" := Sample Uniform {0} {1} {ltr01} in
  let "x" := Sample Binomial {8} #{"p"} in
  let "_" := {guard "x" 5} in
  let "y" := Sample Binomial {3} #{"p"} in
  return {1}:N <= #{"y"}].
```

Formal semantics of sfPPL [Saito and Affeldt, 2023]

Basic idea of the semantics [Staton, 2017]:

- deterministic expressions compile to measurable functions:

$$[\![\Gamma \vdash_{\mathsf{D}} e : \mathbf{A}]\!] : [\![\Gamma]\!] \to [\![\mathbf{A}]\!]$$

- probabilistic expressions compile to s-finite kernel:

$$[\![\Gamma \vdash_{\mathsf{P}} e : \mathbf{A}]\!] : [\![\Gamma]\!] \overset{\text{s-fin}}{\rightsquigarrow} [\![\mathbf{A}]\!]$$

Formally, we provide two functions `execD` and `execP` so that semantics can be computed by syntax-directed rewrites

- Example: semantics of let expressions is composition of s-finite kernels (see Slide 18)

```
Lemma execP_letin g x t1 t2
    (e1 : exp P g t1) (e2 : exp P ((x, t1) :: g) t2) :
  execP [let x := e1 in e2] =
  kcomp' (execP e1) (execP e2).
```

Reminder: Shan's proof of Eddy's table game

0

normalize(
let p := sample (uniform$(0,1)$) in
let x := sample (binomial$(8,p)$) in
let _ := guard$(x=5)$ in
let y := sample (binomial$(3,p)$) in
return$(1 \le y)$)

$\rightarrow$

1

normalize(
let p := sample (uniform$(0,1)$) in
let x := sample (binomial$(8,p)$) in
let _ := guard$(x=5)$ in
sample $\left(\text{bernoulli}\left(1-(1-p)^3\right)\right)$)

$\downarrow$

2

normalize(
let p := sample (uniform$(0,1)$) in
let _ := score $\left(56p^5(1-p)^3\right)$ in
sample $\left(\text{bernoulli}\left(1-(1-p)^3\right)\right)$)

Slide 40
$\leftarrow$

3

normalize(
let _ := score $\left(\frac{1}{9}\right)$ in
let p := sample (beta$(6,4)$) in
sample $\left(\text{bernoulli}\left(1-(1-p)^3\right)\right)$)

$\downarrow$

4

normalize(
let _ := score $\left(\frac{1}{9}\right)$ in
sample $\left(\text{bernoulli}\left(\frac{10}{11}\right)\right)$)

$\rightarrow$

5

normalize $\left(\text{sample}\left(\text{bernoulli}\left(\frac{10}{11}\right)\right)\right)$

Sample transformation: 3 → 4

$$\boxed{\begin{array}{l}\text{normalize}(\\ \text{let } \_ := \text{score}\left(\frac{1}{9}\right) \text{ in}\\ \text{let } p := \text{sample}\,(\text{beta}(6,4)) \text{ in}\\ \text{sample}\left(\text{bernoulli}\left(1-(1-p)^3\right)\right))\end{array}}$$

$\downarrow$ collapses samplings

$$\boxed{\begin{array}{l}\text{normalize}(\\ \text{let } \_ := \text{score}\left(\frac{1}{9}\right) \text{ in}\\ \text{sample}\left(\text{bernoulli}\left(\frac{10}{11}\right)\right))\end{array}}$$

The heart of transformation $3 \to 4$

$$\boxed{\begin{array}{l}\text{let } p := \text{sample}\,(\text{beta}(6,4)) \ \text{ in} \\ \text{sample}\,\big(\text{bernoulli}\,\big(1-(1-p)^3\big)\big)\end{array}}$$

$$\downarrow$$

$$\boxed{\text{sample}\left(\text{bernoulli}\left(\frac{10}{11}\right)\right)}$$

Semantically, for all U:

$$\int_z \underline{bernoulli}\,\big(1-(1-z)^3\big)\,\mathbf{d}\,\underline{beta\,(6,4)}\;U = \underline{bernoulli}\left(\frac{10}{11}\right)U$$

Proving transformation $\boxed{3} \to \boxed{4}$

Goal:

$$\int_z \underline{bernoulli}\left(1-(1-z)^3\right) \mathbf{d}\,\underline{beta\,(6,4)}\ U = \underline{bernoulli}\left(\frac{10}{11}\right) U$$

This can be derived from:

$$\int_z \underline{bernoulli}\left((1-z)^3\right) \mathbf{d}\,\underline{beta\,(6,4)}\ U = \underline{bernoulli}\left(\frac{1}{11}\right) U$$

This is an instance of (for all a, b, c, d):

$$\int_{x\in[0,1]} \underline{bernoulli}\left(x^c(1-x)^d\right) \mathbf{d}\,\underline{beta\,(a,b)} = \underline{bernoulli}\left(\frac{\beta\,(a+c)\,(b+d)}{\beta\,a\,b}\right)$$

Which is proved using the relation probability measure/density function and the relation between the β function and factorial (see Slide 24)—and the `lra` tactic

Outline

Other applications

- Observing a noisy draw from a normal distribution [Affeldt et al., 2025c]
- Quantum programming [Zhou et al., 2023]
- Study of fuzzy logics by Natalia Slusarz [Affeldt et al., 2024b]
  - Truth values are not boolean values but ranges of real numbers/extended real numbers
  - The semantics of formulas becomes real-valued function whose differentiation properties are a topic of interest (e.g., *shadow-lifting* [Várnai and Dimarogonas, 2020])

The papers used for this talk

To check the related work

About MathComp-Analysis:

- asymptotic reasoning [Affeldt et al., 2018]
- formalization of hierarchies [Affeldt et al., 2020]
- measure theory [Affeldt and Cohen, 2023, Ishiguro and Affeldt, 2024]
- first fundamental theorem of calculus [Affeldt and Stone, 2024]
- probability theory (see Alessandro Bruni's talk at ITP [Affeldt et al., 2025a])

About probabilistic programming using MathComp-Analysis:

- kernels [Affeldt et al., 2023], probabilistic termination [Affeldt et al., 2025b]
- Syntax and semantics [Saito and Affeldt, 2023]
- Lebesgue integration toolbox and probability distribution [Affeldt et al., 2025c]

Summary

We have explained several aspects of MathComp-Analysis:

- basic theories and their relation with MathComp
- measure theory and its pervasive use of hierarchies
- the toolbox of Lebesgue integration and probability distributions
- omitted aspect: topology

We focused in particular on one application:

- sfPPL: a first-order probabilistic programming language
- intrinsically-typed encoding using Rocq features (bidirectional hints, canonical structures, custom entries)
- the mechanization of Shan's proof of Eddy's table game by rewriting

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

[Affeldt et al., 2024a] Affeldt, R., Barrett, C. W., Bruni, A., Daukantas, I., Khan, H., Saikawa, T., and Schürmann, C. (2024a).
Robust mean estimation by all means (short paper).
In *15th International Conference on Interactive Theorem Proving (ITP 2024), September 9–14, 2024, Tbilisi, Georgia*, volume 309 of *LIPIcs*, pages 39:1–39:8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Affeldt et al., 2025a] Affeldt, R., Bruni, A., Cohen, C., Roux, P., and Saikawa, T. (2025a).
Formalizing concentration inequalities in Rocq: infrastructure and automation.
In *16th International Conference on Interactive Theorem Proving (ITP 2025), Reykjavik, Iceland, September 29–October 2, 2025*, volume 352 of *Leibniz International Proceedings in Informatics*, pages 21:1–21:20. Schloss Dagstuhl.

[Affeldt et al., 2024b] Affeldt, R., Bruni, A., Komendantskaya, E., Slusarz, N., and Stark, K. (2024b).
Taming differentiable logics with Coq formalisation.
In *15th International Conference on Interactive Theorem Proving (ITP 2024), September 9–14, 2024, Tbilisi, Georgia*, volume 309 of *LIPIcs*, pages 4:1–4:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Affeldt and Cohen, 2023] Affeldt, R. and Cohen, C. (2023).
Measure construction by extension in dependent type theory with application to integration.
J. Autom. Reason., 67(3):28:1–28:27.

[Affeldt et al., 2020] Affeldt, R., Cohen, C., Kerjean, M., Mahboubi, A., Rouhling, D., and Sakaguchi, K. (2020).
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. Part II.

[Affeldt et al., 2018] Affeldt, R., Cohen, C., and Rouhling, D. (2018).
Formalization techniques for asymptotic reasoning in classical analysis.
J. Formaliz. Reason., 11(1):43–76.

[Affeldt et al., 2023] Affeldt, R., Cohen, C., and Saito, A. (2023).
Semantics of probabilistic programs using s-finite kernels in Coq.
In *12th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2023), Boston, MA, USA, January 16–17, 2023*, pages 3–16. ACM.

[Affeldt et al., 2025b] Affeldt, R., Cohen, C., and Saito, A. (2025b).
Semantics of probabilistic programs using s-finite kernels in dependent type theory.
ACM Transactions on Probabilistic Machine Learning, 1(3):16:1–16:34.

[Affeldt et al., 2025c] Affeldt, R., Ishiguro, Y., and Stone, Z. (2025c).
A formal foundation for equational reasoning on probabilistic programs.
In *23rd Asian Symposium on Programming Languages and Systems (APLAS 2025), October 27–30, 2025, Bengaluru, India*, Lecture Notes in Computer Science. Springer.
In press.

[Affeldt and Stone, 2024] Affeldt, R. and Stone, Z. (2024).
A comprehensive overview of the lebesgue differentiation theorem in Coq.
In *15th International Conference on Interactive Theorem Proving (ITP 2024), September 9–14, 2024, Tbilisi, Georgia*, volume 309 of *LIPIcs*, pages 5:1–5:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Bernard et al., 2021] Bernard, S., Cohen, C., Mahboubi, A., and Strub, P. (2021).
Unsolvability of the quintic formalized in dependent type theory.
In *12th International Conference on Interactive Theorem Proving (ITP 2021), June 29–July 1, 2021, Rome, Italy (Virtual Conference)*, volume 193 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Bertot et al., 2008] Bertot, Y., Gonthier, G., Biha, S. O., and Pasca, I. (2008).
Canonical big operators.
In *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.

[Cohen et al., 2020] Cohen, C., Sakaguchi, K., and Tassi, E. (2020).
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.

[Eddy, 2004] Eddy, S. R. (2004).
What is Bayesian statistics?
Nature Biotechnology, 22(9):1177–1178.

[Gonthier, 2008] Gonthier, G. (2008).
Formal proof—the four-color theorem.
Notices of the AMS, 55(11):1382–1393.

[Gonthier et al., 2013a] Gonthier, G., Asperti, A., Avigad, J., Bertot, Y., Cohen, C., Garillot, F., Roux, S. L., Mahboubi, A., O'Connor, R., Biha, S. O., Pasca, I., Rideau, L., Solovyev, A., Tassi, E., and Théry, L. (2013a).
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.

[Gonthier et al., 2013b] Gonthier, G., Ziliani, B., Nanevski, A., and Dreyer, D. (2013b).
How to make ad hoc proof automation less ad hoc.
J. Funct. Program., 23(4):357–401.

[Halmos, 1974] Halmos, P. R. (1974).
Measure Theory.
Springer.

[Ishiguro and Affeldt, 2024] Ishiguro, Y. and Affeldt, R. (2024).
The Radon-Nikodým theorem and the Lebesgue-Stieltjes measure in Coq.
Computer Software, 41(2).

[Melquiond, 2008] Melquiond, G. (2008).
Proving bounds on real-valued functions with computations.
In *4th International Joint Conference on Automated Reasoning (IJCAR 2008), Sydney, Australia, August 12–15, 2008*, volume 5195 of *Lecture Notes in Computer Science*, pages 2–17. Springer.

[Saikawa et al., 2025] Saikawa, T., Matsuda, K., and Tsuji, Y. (2025).
Formalization of matching numbers with finmap and mathcomp-classical.
In *The Rocqshop 2025, Reykjavik, Iceland, September 27, 2025.*

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

[Shan, 2018a] Shan, C. (2018a).
Calculating distributions.
In *20th International Symposium on Principles and Practice of Declarative Programming (PPDP 2018), Frankfurt am Main, Germany, September 3–5, 2018*, pages 2:1–2:5. ACM.

[Shan, 2018b] Shan, C.-C. (2018b).
Equational reasoning for probabilistic programming.
POPL 2018 TutorialFest.
Available at: `https://homes.luddy.indiana.edu/ccshan/rational/equational-handout.pdf`.

[Staton, 2017] Staton, S. (2017).
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.

[The Rocq Development Team, 2025a] The Rocq Development Team (2025a).
Bidirectionality hints.
Inria.
Chapter Setting properties of a function's arguments of
[The Rocq Development Team, 2025c], direct link.

[The Rocq Development Team, 2025b] The Rocq Development Team (2025b).
Custom entries.
Inria.
Chapter Syntax extensions and notation scopes of [The Rocq Development Team, 2025c], direct link.

[The Rocq Development Team, 2025c] The Rocq Development Team (2025c).
The Rocq Proof Assistant Reference Manual.
Inria.
Available at `https://rocq-prover.org/doc/V9.0.0/refman/index.html`. Version 9.0.0.

[Várnai and Dimarogonas, 2020] Várnai, P. and Dimarogonas, D. V. (2020).
On robustness metrics for learning STL tasks.
In *2020 American Control Conference (ACC 2020), Denver, CO, USA, July 1–3, 2020*, pages 5394–5399. IEEE.

[Zhou et al., 2023] Zhou, L., Barthe, G., Strub, P., Liu, J., and Ying, M. (2023).
CoqQ: Foundational verification of quantum programs.
Proc. ACM Program. Lang., 7(POPL):833–865.