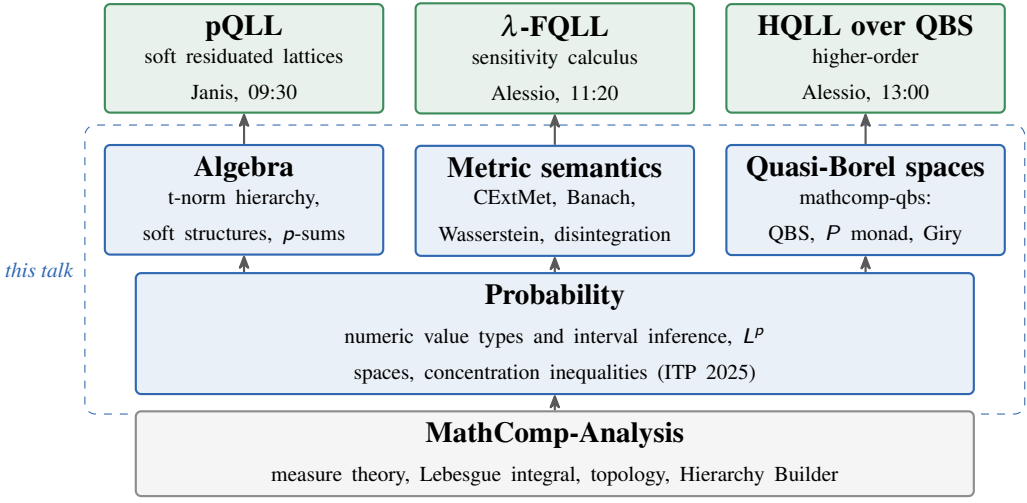
Rocq Infrastructure for Quantitative Logics in MathComp-Analysis

Alessandro Bruni
IT University of Copenhagen

QTT 2026, Copenhagen, 24 November 2026

ONE STACK, SEVERAL TALKS TODAY



Three constructions to watch for

- **L^p norms and Hölder's inequality.**
  Moment bounds in probability; p-sums in the soft logics; the cost of a transport plan.
- **Convex combinations and barycentric algebras.**
  The power function behind Minkowski; soft connectives monotone in p; Wasserstein spaces.
- **Monads built from kernels.**
  The Giry monad; the semantics of probabilistic programs; transport plans as kernels.

Each is one piece of MathComp-Analysis, used three times in three roles. That is the case for one library rather than three.

ITP 2025: ONE THEOREM, THREE TOOLS

Affeldt, Bruni, Cohen, Roux, Saikawa. *Formalizing concentration inequalities in Rocq: infrastructure and automation.* ITP 2025.

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

Markov, Chebyshev, Chernoff, Hölder, Minkowski, one numerical lemma. What survived as a library:

1. **types for numeric values** and a light interval-inference procedure
2. L^p **spaces** added to the structure hierarchy, conservatively
3. a bridge to **CoqInterval** for numerical side-goals

INTERVAL INFERENCE: SIGN GOALS DISAPPEAR

Every proof about probabilities has side-goals of this shape:

`0 <= k%:R / 2 ^+ n`	$0 \le k/2^n$
`0 < 1 + ‘|r|`	$0 < 1 + \lvert r \rvert$
`0 < (2 / e%:num) ^+ 2` with `e` nonnegative	$0 < (2/e)^2$

A type of values in an interval, the interval *inferred* by canonical structures:

```
Check n.+1%:itv.                  : def nat ‘[1, +oo[
Check (1 / 2 ^ n.+1)%:itv.        : def R ‘]0, +oo[
Check (x%:num + 1)%:itv.          : {posnum R} -> def R ‘]1, +oo[
```

- one instance per operator, no reification; over-approximation, refined on demand
- in MathComp: called 1604 times in MathComp-Analysis, closes 729 subgoals
- QLLib extends it to $[0, \infty]$: inversion, powers, `expeR` (`interval_einference.v`)

L^p SPACES: A p-NORM IS NOT A NORM

$\|f\|_p = \left(\int |f|^p\right)^{1/p}$ is *not* positive definite when there is a nonempty null set.

- split `Zmodule_isNormed` into **seminorm** + positive definiteness
- old instances unchanged: *factories*
- $\mathscr{L}^p$: a structure on measurable functions; L^p: quotient by a.e. equality

```
HB.mixin Record isLfunction
    (p : \bar R) (p1 : 1 <= p) (f : T -> R)
    of MeasurableFun d _ T R f :=
  { Lfunction_finite :
      'N[P]_p [EFin \o f] < +oo }.

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

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

Same $\|\cdot\|_p$ over the *counting* measure gives the p-sums of the soft logics.

In progress with Arthur and Reynald Affeldt: the Bochner integral in MathComp-Analysis, i.e. L^p for Banach-space-valued functions: Borel structure on normed modules, simple functions into Banach spaces.

HIERARCHY BUILDER

Cohen, Sakaguchi, Tassi. *Hierarchy Builder: algebraic hierarchies made easy in Coq.* FSCD 2020.

Mixin: operations and axioms added to a carrier, assuming weaker structure.

Structure: a carrier with a *set* of mixins. The lattice of structures, coercions and inheritance are *computed* by HB.

Factory: another way to build mixins; *builders* say how. Old interfaces survive as factories.

Instance: a carrier plus a mixin or factory; HB registers every structure it now satisfies.[1]

```
HB.mixin Record isMonoid T := {
  one : T; mul : T -> T -> T;
  mulA : associative mul;
  mul1 : left_id one mul; mulr1 : right_id one mul }.
HB.structure Definition Monoid := { T of isMonoid T }.

HB.mixin Record Monoid_isGroup T of Monoid T := {
  inv : T -> T; mulV : forall x, mul (inv x) x = one }.
HB.structure Definition Group :=
  { T of Monoid T & Monoid_isGroup T }.

HB.factory Record isGroup T := { (* all at once *) }.
HB.builders Context T of isGroup T.
  HB.instance Definition _ := isMonoid.Build T ...
  HB.instance Definition _ := Monoid_isGroup.Build T ...
HB.end.

HB.instance Definition _ := isGroup.Build int ...
```

[1] Inheritance is resolved by unification against the canonical record, not by instance search, so inheritance paths agree by construction and a refinement leaves instances unchanged.

BACKWARDS-COMPATIBLE HIERARCHY REFINEMENT

Normed Zmodule:

- $\|\cdot\| : T \to \mathbb{R}$
- $\|x+y\| \leq \|x\| + \|y\|$
- $\|n \cdot x\| = n \cdot \|x\|$
- $\|-x\| = \|x\|$
- $\|x\| = 0 \to x = 0$

```
HB.mixin Record Zmodule_isNormed
  (R : POrderedZmodule.type)
  M of GRing.Zmodule M := {
 norm : M -> R;
 ler_normD :
  forall x y, norm (x + y) <= norm x + norm y;
 normrMn : forall x n, norm (x *+ n) = norm x *+ n;
 normrN : forall x, norm (- x) = norm x;
 normr0_eq0 : forall x, norm x = 0 -> x = 0 }.
```

BACKWARDS-COMPATIBLE HIERARCHY REFINEMENT

Seminormed Zmodule:

- $\|\cdot\| : T \to \mathbb{R}$
- $\|x+y\| \leq \|x\| + \|y\|$
- $\|n \cdot x\| = n \cdot \|x\|$
- $\|-x\| = \|x\|$

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

Normed Zmodule:

- $\|\cdot\| : T \to \mathbb{R}$
- $\|x+y\| \leq \|x\| + \|y\|$
- $\|n \cdot x\| = n \cdot \|x\|$
- $\|-x\| = \|x\|$
- $\|x\| = 0 \to x = 0$

```
HB.mixin Record Zmodule_isNormed
  (R : POrderedZmodule.type)
  M of GRing.Zmodule M := {
norm : M -> R;
ler_normD :
 forall x y, norm (x + y) <= norm x + norm y;
normrMn : forall x n, norm (x *+ n) = norm x *+ n;
normrN : forall x, norm (- x) = norm x;
normr0_eq0 : forall x, norm x = 0 -> x = 0 }.
```

BACKWARDS-COMPATIBLE HIERARCHY REFINEMENT

Seminormed Zmodule:

- $\|\cdot\| : T \to \mathbb{R}$
- $\|x+y\| \leq \|x\| + \|y\|$
- $\|n \cdot x\| = n \cdot \|x\|$
- $\|-x\| = \|x\|$

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

Normed Zmodule:

- $\|\cdot\| : T \to \mathbb{R}$
- $\|x+y\| \leq \|x\| + \|y\|$
- $\|n \cdot x\| = n \cdot \|x\|$
- $\|-x\| = \|x\|$
- $\|x\| = 0 \to x = 0$

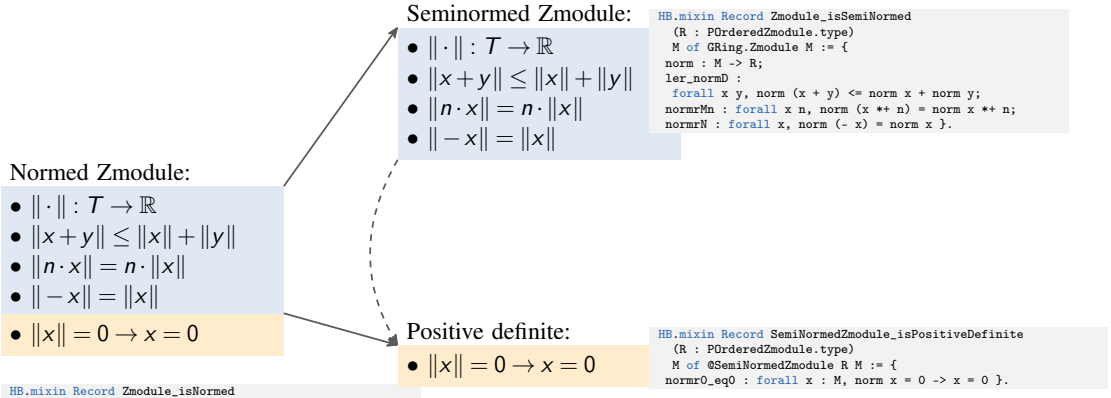
Positive definite:

- $\|x\| = 0 \to x = 0$

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

```
HB.mixin Record Zmodule_isNormed
  (R : POrderedZmodule.type)
 M of GRing.Zmodule M := {
norm : M -> R;
ler_normD :
 forall x y, norm (x + y) <= norm x + norm y;
normrMn : forall x n, norm (x *+ n) = norm x *+ n;
normrN : forall x, norm (- x) = norm x;
normr0_eq0 : forall x, norm x = 0 -> x = 0 }.
```

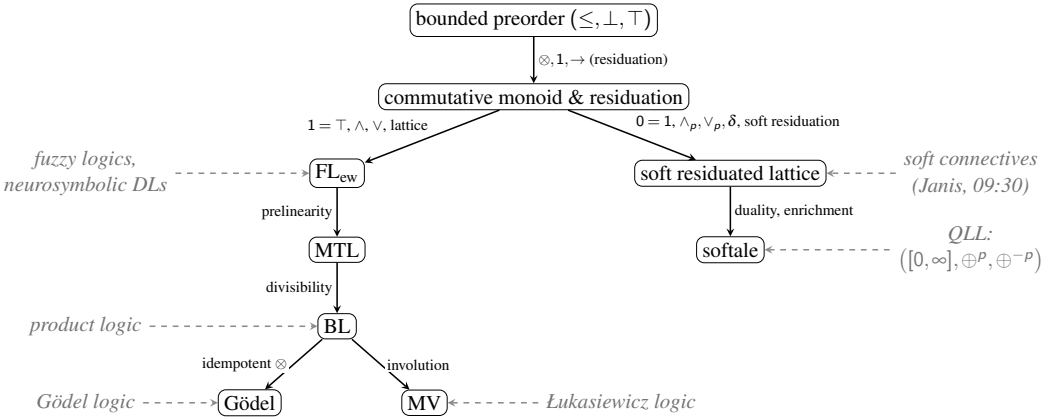
BACKWARDS-COMPATIBLE HIERARCHY REFINEMENT



old `Zmodule_isNormed` *becomes a factory: every existing instance still builds*

THE ALGEBRAIC HIERARCHY FOR QUANTITATIVE LOGICS

Bailitis, Affeldt, Bruni, Coltellacci, Komendantskaya, Stark. *Soft Residuated Lattices and Quantitative Substructural Logics in Rocq.* Draft, 2026.



A JOIN ON THE FUZZY SIDE

Bailitis, Affeldt, Bruni, Coltellacci, Komendantskaya, Stark. *Soft Residuated Lattices and Quantitative Substructural Logics in Rocq.* Draft, 2026.

```
HB.structure Definition Fuzzy d :=
  { T of Tnorm d T & Tconorm d T & Residuated d T & Negation d T
       & Fuzzy_isNegImpl d T & MulComulNeg_isDeMorgan d T }.
HB.structure Definition FLewAlgebra d :=
  { T of Fuzzy d T & Order.Lattice d T }.

HB.mixin Record Residuated_isPrelinear d T
    of Residuated d T & Order.Lattice d T :=
  { mimpl_prelinear : forall x y, (mimpl x y) `|` (mimpl y x) = \top }.
HB.mixin Record Residuated_isDivisible d T
    of Residuated d T & Order.Lattice d T :=
  { mand_divisible : forall x y, mand x (mimpl x y) = x `&` y }.
HB.mixin Record POComMonoid_isIdempotent d T of POComMonoid d T :=
  { mandxx : forall x, mand x x = x }.

HB.structure Definition MTLAlgebra d :=
  { T of FLewAlgebra d T & Residuated_isPrelinear d T }.
HB.structure Definition GodelAlgebra d :=
  { T of MTLAlgebra d T & Residuated_isDivisible d T
       & POComMonoid_isIdempotent d T }.
```

$\mathbf{FL}_{ew}$: a bounded lattice with a commutative monoid $\otimes$, unit $\top$, residuated by $\to$: $x \otimes y \le z \iff x \le y \to z$; negation $\neg x = x \to 0$

prelinear: $(x \to y) \vee (y \to x) = \top$

divisible: $x \otimes (x \to y) = x \wedge y$

idempotent: $x \otimes x = x$

MTL = FL_{ew} + prelinear; Gödel = MTL + divisible + idempotent. HB computes the inclusions, so a theorem on `mtlType` applies to every Gödel algebra.

THE SPLIT: SOFT LATTICE VERSUS SOFTALE

Bailitis, Affeldt, Bruni, Coltellacci, Komendantskaya, Stark. *Soft Residuated Lattices and Quantitative Substructural Logics in Rocq.* Draft, 2026.

Soft residuated lattice, the truth values T: $\wedge_p$, $\vee_p$ commutative, associative, monotone, units $\top$, $\bot$; $\delta \otimes x \leq x \wedge_p x$, $\delta \otimes (x \vee_p x) \leq x$; $x \otimes (y \vee_p z) = (x \otimes y) \vee_p (x \otimes z)$

```
HB.mixin Record Residuated_isSoftLattice d T
    of Residuated d T := {
  smeet : T -> T -> T;  sjoin : T -> T -> T;  slack : T;
  smeetC : commutative smeet;  smeetA : associative smeet;
  smeetTx : left_id \top smeet;
  le_smeet2l : forall x, {homo smeet x : y z / y <= z};
  (* ... dually for sjoin ... *)
  smeet_softidem : forall x, mand slack x <= smeet x x;
  sjoin_softidem : forall x, mand slack (sjoin x x) <= x;
  mand_sjoinDr : forall x y z,
    mand x (sjoin y z) = sjoin (mand x y) (mand x z) }.

HB.structure Definition SoftResiduatedLattice d :=
  { T of Residuated d T & Residuated_isSoftLattice d T }.
```

Instance: $[0,\infty]$ with $\oplus^p$, $\oplus^{-p}$, slack $2^{-1/p}$.

Softale, the propositions S, enriched over V: $a \sqsubseteq b \in V$, $(a \sqsubseteq b) \otimes (b \sqsubseteq c) \leq a \sqsubseteq c$; $(a \bullet b \sqsubseteq c^{\perp}) = (a \sqsubseteq (b \bullet c)^{\perp})$; $(a \sqsubseteq c) \wedge_p (b \sqsubseteq c) \leq a \sqcup b \sqsubseteq c$

```
HB.mixin Record isEnrichedOrder dV dS (V : nonnegERealType dV) S
    of Order.TBPreorder dS S := {
  sord : S -> S -> V;                    (* a V-valued order *)
  sord_trans : forall a b c, mand (sord a b) (sord b c) <= sord a c;
  ... }.

HB.mixin Record EnrichedOrder_isMulSoftale dV dS V S
    of EnrichedOrder dV dS V S & DeMorgan dS S := {
  mand_mneg : forall a b c,              (* star-autonomy *)
    sord (mand a b) (mneg c) = sord a (mneg (mand b c));
  ... }.
HB.structure Definition MulSoftale dV dS V := { S of EnrichedOrder dV dS V S & DeMorgan dS S & ... }.

HB.mixin Record EnrichedOrder_isAddSoftale dV dS V S
    of EnrichedOrder dV dS V S := {
  sbot : S;  ssup : S -> S -> S;
  ssup_min : forall a b c, smeet (sord a c) (sord b c) <= sord (ssup a b) c;
  ssup_maj : forall a b c, sjoin (sord c a) (sord c b) <= sord c (ssup a b);
  ... }.
HB.structure Definition AddSoftale dV dS V := { S of EnrichedOrder dV dS V S & ... }.

HB.structure Definition Softale dV dS V :=
  { S of MulSoftale dV dS V S & AddSoftale dV dS V S }.
```

Tensor, dual and par come from `DeMorgan`; the soft meet and join of V grade the sup of S.

Why this is comfortable in Rocq

Hierarchy Builder

- one *mixin* per connective; structures are sets of mixins, their lattice is *computed*, not declared
- *factories* refine a hierarchy after the fact without touching an instance (Affeldt, Cohen, Kerjean, Mahboubi, Rouhling, Sakaguchi, IJCAR 2020)
- QLLib adds the soft structures from *outside*, no patch to MathComp
- classical reasoning is quarantined: MathComp is axiom-free; Analysis keeps its axioms in one package and `constructive_ereal.v` beside `ereal.v`
- with class-and-`extends` hierarchies (Mathlib[2]), refining a structure means revisiting its instances

Elpi

- HB itself is an Elpi program: the hierarchy is *data*
- Coq-Elpi tactics are λProlog: unification and backtracking come free, so sequent-calculus proof search is the rules as clauses. Done for Incorrectness Logic: iterative deepening, symbolic inversion (Inclogic-rocq, 2026)
- next: one calculus and one proof-search strategy over the whole hierarchy, each logic an instance, derivations checked. MoSeL, for quantitative logics

[2]Mathlib is larger. For building logics algebraically and searching for proofs in them, this toolchain is the better fit.

PROOF SEARCH AS CLAUSES: PQLL IN ELPI (SKETCH)

Sketch over the one-sided calculus `Oprv` of QLLib (`qll_defs.v`), cut-free.

```
Elpi Tactic qll_search.
Elpi Accumulate lp:{{
% split a context into two sub-multisets; a generator, no cut: every split is tried
pred split i:list term, o:list term, o:list term.
split [] [] [].
split [A|G] [A|L] R :- split G L R.
split [A|G] L [A|R] :- split G L R.

% prove G D : D is a cut-free derivation of  |-O G ; one clause per rule of Oprv
pred prove i:list term, o:term.
prove [{{ neg lp:A }}, A]          {{ OAX lp:A }}.
prove [{{ one }}]                  {{ Oone }}.
prove [{{ top }} | G]              {{ Otop lp:G }}.
prove [{{ bin par lp:A lp:B }} | G] {{ Opar _ _ _ lp:D }} :- prove [A, B | G] D.
prove [{{ bin tensor lp:A lp:B }} | G] {{ Otensor _ _ _ _ lp:D1 lp:D2 }} :-
  split G G1 G2, prove [A | G1] D1, prove [B | G2] D2.
prove [{{ bin add_and lp:A lp:B }} | G] {{ Oand _ _ _ lp:D1 lp:D2 }} :-   % soft: both premises,
  prove [A | G] D1, prove [B | G] D2.                                      % validity is a p-sum
prove [{{ bin add_or lp:A lp:B }} | G] {{ Oor _ _ _ lp:D1 lp:D2 }} :-
  prove [A | G] D1, prove [B | G] D2.
prove G {{ OEXCH _ _ _ _ lp:D }} :-  % exchange: bring any other formula to the front
  pick G A G1, not (is-front G A), prove [A | G1] D.

solve (goal _ _ {{ Oprv lp:Gamma }} _ _ as Goal) GL :-
  list-of-term Gamma G, prove G D, refine D Goal GL.   % D is checked by the kernel
}}.
```

SOFT CONNECTIVES REUSE THE PROBABILITY LAYER

A p-sum *is* an L^p norm over the counting measure on two points:

```
Definition p_sum_int_fun (a b : {nonneg \bar R}) (n : nat) :=
  if n == 0 then a else if n == 1 then b else 0%:E%:nng.    (* the sequence a, b, 0, 0, ... *)
Definition p_sum (p : \bar R) (a b : {nonneg \bar R}) :=
  'N[counting]_p [fun n => (p_sum_int_fun a b n)%:num].

Definition p_sum_de_morgan p a b :=          (* negative p: De Morgan dual *)
  if p > 0 then p_sum_nng p a b
  else if p < 0 then invnnge (p_sum_nng (-p) (invnnge a) (invnnge b)) else 0.
```

$a \oplus^p b = (a^p + b^p)^{1/p}$ $\quad a \oplus^{-p} b = \left((a^{-1})^p + (b^{-1})^p\right)^{-1/p}$ $\quad \oplus^p \to \max$, $\oplus^{-p} \to \min$ as $p \to \infty$: the idempotent connectives are limits

- Hölder and Minkowski come for free; monotonicity and differentiability in p are proved once, in `nonneg_ereal.v`
- used in the proof theory:

```
Theorem cut_elimination {S G : list qll_formula} (P : S |- G) :
  exists Q : S |- G, cut_free Q /\ (validity P <= validity Q)%O.
```

CExtMet in Rocq

Coltellacci, Bruni. ExtMet-QLL, `analysis_extras.v`, `extmet.v`, `cextmet.v`.

```
(* objects: separated extended pseudometric spaces *)
HB.structure Definition ExtMetric R :=
  { M of PseudoMetric R M & Topological_isHausdorff M }.
HB.factory Record PseudoMetric_isExtMetric R M
    & PseudoMetric R M :=
  { edist_eq0 : forall x y : M, edist (x, y) = 0 -> x = y }.
HB.structure Definition CompleteExtMetric R :=
  { M of CompletePseudoMetric R M & Topological_isHausdorff M }.

(* morphisms: non-expansive maps; X -o Y carries the sup distance *)
Record nemap X Y := NEMap
  { nemap_fun :> X -> Y; nemap_ne : nonexpansive nemap_fun }.

(* constructions: fresh types over the same carriers *)
Notation tensor := (lp_prod itv1).                (* l^1 product *)
Definition scale r X := X.                  (* distance r * edist *)
Definition nemap_curry (f : tensor X Y -o Z) : X -o (Y -o Z) := ...
Lemma nemap_complete X Y :
  complete_space Y -> complete_space (X -o Y).
```

MathComp-Analysis
pseudoMetricType, completeType, edist

one factory

extMetricType
pseudometric + Hausdorff

fresh types

CExtMet: 1, $X \otimes Y$, rX, $X + Y$, $X \multimap Y$ closed monoidal, complete

$$d_{X \otimes Y} = d_X + d_Y \qquad d_{rX} = r \cdot d_X$$
$$d_{X+Y}(\mathrm{inl}\, x, \mathrm{inr}\, y) = +\infty$$
$$d_{X \multimap Y}(f, g) = \sup_x d_Y(fx, gx)$$

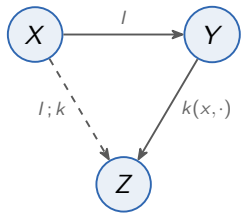
KERNELS IN MATHCOMP-ANALYSIS

Affeldt, Cohen, Saito. *Semantics of probabilistic programs using s-finite kernels in Coq.* CPP 2023; `kernel.v`, after Staton, ESOP 2017.

```
(* a kernel: a measure on Y for each x, measurable in x *)
HB.mixin Record isKernel d d' (X : measurableType d)
    (Y : measurableType d') (R : realType)
    (k : X -> {measure set Y -> \bar R}) := {
  measurable_kernel : forall U, measurable U ->
    measurable_fun [set: X] (k ^~ U) }.
HB.structure Definition Kernel d d' X Y R :=
  { k & isKernel _ _ X Y R k }.                (* R.-ker X ~> Y *)
(* refined by mixins: R.-sfker (s-finite), R.-pker (probability) *)

(* parameterised composition: the context X is kept *)
Variable l : X -> {measure set Y -> \bar R}.
Variable k : X * Y -> {measure set Z -> \bar R}.
Definition kcomp x (U : set Z) := \int[l x]_y k (x, y) U.
(* mkcomp l k : again an s-finite kernel *)

Lemma integral_kcomp x f :
  (forall z, 0 <= f z) -> measurable_fun [set: Z] f ->
  \int[kcomp l k x]_z f z = \int[l x]_y (\int[k (x, y)]_z f z).
```



$k : X \to \mathscr{M}(Y)$, $x \mapsto k(x)(U)$ measurable for each U

$(l;k)(x)(U) = \int k(x,y)(U)\, l(x)(dy)$

$\int f\, d(l;k)(x) = \iint f\, dk(x,y)\, l(x)(dy)$

The last line is the Fubini-style fact behind the Giry laws and the Wasserstein triangle inequality.

THE GIRY MONAD FROM KERNELS

Bruni, Affeldt. *Giry monad for probabilities.* math-comp/analysis PR #1177, 2024.

MathComp-Analysis has s-finite kernels, their composition `kcomp`, and `integral_kcomp`. So encode the monad as kernels and let the library do the work:

```
Definition ret : R.-pker T ~> T := kdirac (@measurable_id _ _ setT).

Definition bind (mu : probability T R) (f : R.-pker T ~> T') :=
  kcomp (kprobability (measurable_cst mu)) (pker_snd f) tt.
Lemma bindE A : bind mu f A = \int[mu]_x f x A.   Proof. by []. Qed.
```

$\mathrm{ret}\, x = \delta_x$ $\quad$ $\mathrm{bind}\, \mu\, f\, (A) = \int f(x)(A)\, \mu(dx)$: μ as a kernel $1 \rightsquigarrow T$, composed with f lifted to $1 \times T \rightsquigarrow T'$

The three laws are three library lemmas:

`bind (ret x) f = f x`	$\int f\, d\delta_x = f(x)$	left unit	`integral_dirac`
`bind mu ret = mu`	$\int \delta_x(A)\, \mu(dx) = \mu(A)$	right unit	`integral_indic`
`bind (bind mu f) g = bind mu (bindfg f g)`	Fubini	associativity	`integral_kcomp`

Left open there: measurability of `bind` in `mu`, the σ-algebra on probabilities (now in `giry.v` upstream). Next: the same shape for transport plans.

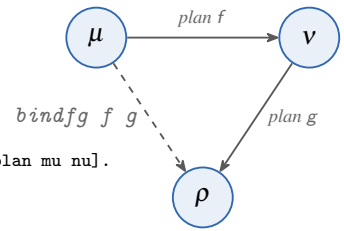
WASSERSTEIN DISTANCES VIA KERNELS, NOT COUPLINGS

Coltellacci, Bruni. ExtMet-QLL, `wasserstein.v`.

A transport plan from μ to ν is a probability kernel, not a coupling:

```
Definition is_plan mu nu (k : R.-pker X ~> Y) :=
  forall A, measurable A -> nu A = bind mu k A.
Definition wcost p mu k : \bar R :=
  (\int[mu]_x \int[k x]_y edist (x, y) `^ p) `^ p^-1.
Definition wdir p mu nu := ereal_inf [set wcost p mu k | k in is_plan mu nu].
Definition wdist p mu nu := maxe (wdir p mu nu) (wdir p nu mu).

Lemma wcost_comp p mu nu f g : is_plan mu nu f ->
  wcost p mu (bindfg f g) <= wcost p mu f + wcost p nu g.
Lemma wdist_triangle p mu nu rho :
  wdist p mu rho <= wdist p mu nu + wdist p nu rho.
```



$$\mathrm{cost}(\mu, f; g) \le \mathrm{cost}(\mu, f) + \mathrm{cost}(\nu, g)$$

- couplings exist today (`is_coupling`, `wass`) but do not compose: gluing needs disintegration
- plans compose by `bindfg`; the triangle inequality is Minkowski on the composite

$W_p^{\rightarrow}(\mu,\nu) = \inf_k \left(\iint d^p \, dk(x) \, d\mu \right)^{1/p}$
over plans k with $\mathrm{bind}\,\mu\, k = \nu$
$W_p = \max(W_p^{\rightarrow}, W_p^{\leftarrow})$
a plan k induces the coupling $\mu \otimes k$ (`plan_coupling`); on standard Borel spaces every coupling arises this way, so W_p is the coupling distance

The Wasserstein monad: Giry plus a metric

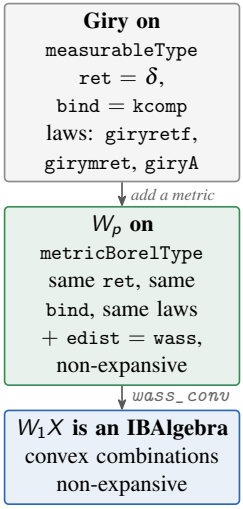
Coltellacci, Bruni. ExtMet-QLL, `wasserstein.v`; Giry monad ported from PR #1177.

```
(* W_p X: the probabilities on X, with the Wasserstein distance *)
Definition wspace p (X : metricBorelType R d) : Type := probability X R.
HB.instance Definition _ := isExtPseudoMetric.Build R (wspace p X) ...
Lemma wspace_edistE mu nu : edist (mu, nu) = wass p mu nu.

(* unit: Giry's, now non-expansive *)
Definition wret (x : X) : wspace p X := \d_x.
Lemma nonexpansive_wret : nonexpansive wret.

(* the monad laws are Giry's, up to distance 0 *)
Lemma wbind_retr mu : edist (bind mu ret, mu) = 0.
Proof. by apply: wspace_edist0 => A mA; exact: girymret. Qed.
Lemma wbindA mu f g :
  edist (bind (bind mu f) g, bind mu (bindfg f g)) = 0.
Proof. by apply: wspace_edist0 => A mA; exact: giryA. Qed.

(* bind is non-expansive, given optimal plans *)
Lemma wass_bind mu : wass w1 (bind mu f) (bind mu g) <=
  ereal_sup [set wass w1 (f t) (g t) | t in setT].
```



MATHCOMP-QBS: THE PROBABILITY MONAD ON QBS

Baudart, with Claude Opus 4.6 and Rocq-MCP. *mathcomp-qbs*, 2026, after Heunen, Kammar, Staton, Yang, LICS 2017.

```
(* a QBS: a type with its random elements R -> T *)
HB.mixin Record isQBS R T := { qbs_Mx : set (measurableTypeR R -> T);
  (* closed under measurable reparametrisation, constants, countable gluing *) ... }.

(* P(X): pairs (alpha, mu), mu a probability on R, quotiented *)
Record preprob X := { preprob_alpha : mR -> X;
  preprob_mu : probability mR R; _ : qbs_Mx preprob_alpha }.
Definition preprob_equiv p1 p2 := forall U, sigma_Mx U ->
  mu p1 (alpha p1 @^-1‘ U) = mu p2 (alpha p2 @^-1‘ U).
Definition probQ X : Type := {eq_quot preprob_equiv}.

Definition qbs_return x : probQ X := mkQBSProb (fun=> x) \d_0 _.
Definition qbs_bind (p : probQ X) (f : qbsHomType X (probQ Y)) : probQ Y :=
  mkQBSProb bind_a (giry_bind (qbs_mu p) _) _.      (* Giry bind on R *)

HB.instance Definition _ := isKernel.Build _ _ _ _ _ kg mkg. (* strength *)
```

random elements
$\alpha : \mathbb{R} \to X$
closed under reparametrisation, constants, countable gluing

↓

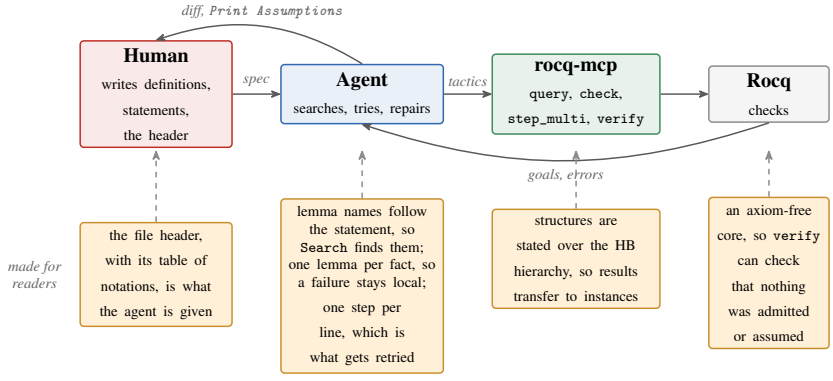
$P(X) = \{(\alpha, \mu)\}/\sim$,
μ on $\mathbb{R}$
quotient as L^p
in `hoelder.v`:
laws are equalities

↓

bind = `giry_bind`
on $\mathbb{R}$ (`giry.v`)
strength via a
probability **kernel**
bridge `to_giry`
to Giry on M

What changes when agents write the proofs

Claude Code with the rocq skills, through rocq-mcp (`https://github.com/LLM4Rocq/rocq-mcp`).



The effort moves from proving to stating. A human writes the definitions, statements and headers, and reviews the assumptions and the definitions against the paper. The proof scripts remain readable, one step per line, and a golfing pass keeps them that way.

THE THREE CONSTRUCTIONS, SEEN THREE TIMES

	probability	algebra	metric semantics
L^p, Hölder	$\mathscr{L}^p$ spaces, Chernoff; Bochner (wip)	p-sums are `'N[counting]_p`	Wasserstein cost is an L^p norm of `edist`
convexity	`convex_powR`, `ConvexSpace`	p-sums monotone in p	$W_1 X$ is a barycentric algebra; Kantorovich soundness of ICA
kernels	Giry monad via kernels (PR #1177)	s-finite kernels as program semantics	plans are kernels; composition is the triangle inequality

One library, because the mathematics is shared. The agents only make the sharing cheaper to exploit.

Links

- ITP 2025 paper and code: `https://doi.org/10.4230/LIPIcs.ITP.2025.21`, `https://github.com/hoheinzollern/analysis/tree/sampling_20250619`
- QLLib (Rocqshop 2026): `https://janis-bai.github.io/QLLib/Rocqshop_2026/toc.html`
- ExtMet-QLL: distributed with the λ-FQLL draft, in its `formalization` folder
- mathcomp-qbs: `https://llm4rocq.github.io/mathcomp-qbs/`
- rocq-mcp: `https://github.com/LLM4Rocq/rocq-mcp`
- MathComp-Analysis: `https://github.com/math-comp/analysis`

Thank you. Questions?