

Lean for Measure-Theoretic Probability: Verifying the π - λ Theorem

Le Chen

Auburn University

Department of Mathematics and Statistics

Math 7800: Probability Theory

Auburn University

Tuesday, September 29, 2026, 2:00–3:15 PM

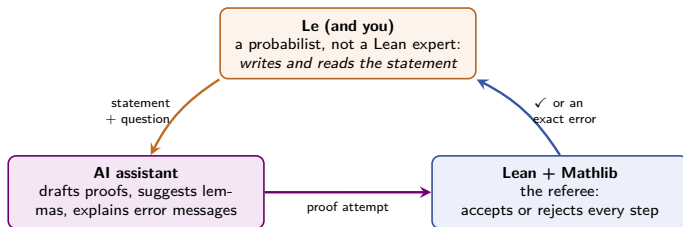
Billingsley §3 (uniqueness of measures)

Two probability measures that agree on a π -system generating the σ -field agree everywhere.

The proof, checked by a computer, line by line:

```
-- To show  $\mu = \nu$  it suffices to check  $\mu t = \nu t$  for every measurable  $t$ .
ext t ht
-- "Let  $L$  be the class of sets where  $\mu$  and  $\nu$  agree;  $L$  is a  $\lambda$ -system  $\supseteq P$ ."
induction t, ht using MeasurableSpace.induction_on_inter hgen hP with
| empty => simp
| basic t ht => exact hagree t ht
| compl t htm ih =>
  --  $P(A^c) = 1 - P(A)$ : no subtraction on  $\mathbb{R}_{\geq 0}^\infty$  without this lemma
  rw [prob_compl_eq_one_sub htm, prob_compl_eq_one_sub htm, ih]
| iUnion f hdisj hfm ih =>
  -- countable additivity on a disjoint union
  rw [measure_iUnion hdisj hfm, measure_iUnion hdisj hfm]
  simp [ih]
```

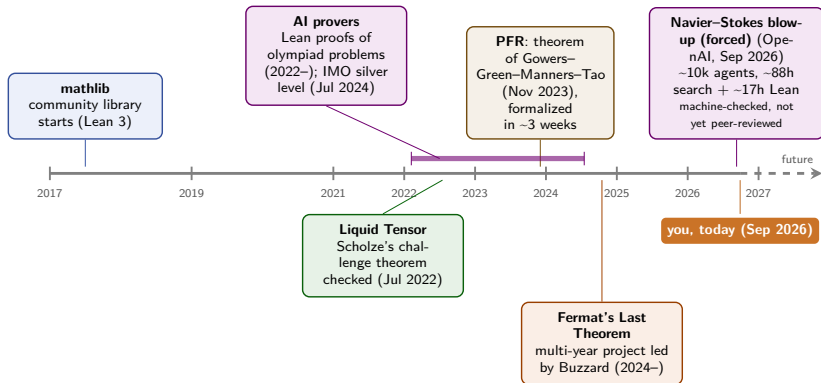
I'm learning with you



House rules for this class

- ▶ Every Lean code block on these slides is cut from a file that compiles against the course pin (Lean v4.34.0, Mathlib v4.34.0).
- ▶ When I don't know, I say so, and we ask Lean together.
- ▶ My own Lean self-grade today: 50/100. Let's raise it together.
- ▶ Through-line: **Lean checks every step; you check the statement.**

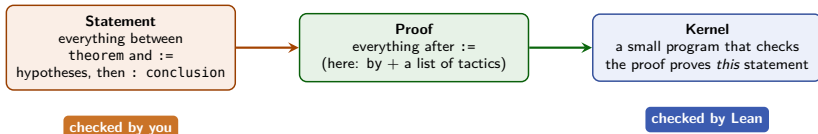
Lean in 2026: how we got here



Mathlib today: most of the measure theory of Billingsley Chapters 2–3 is already there. Today we learn to *read* it.

A theorem has three parts

```
theorem uniqueness_of_extension {Ω : Type*} [m : MeasurableSpace Ω]
  (μ v : Measure Ω) [IsProbabilityMeasure μ] [IsProbabilityMeasure v]
  (P : Set (Set Ω)) (hgen : m = MeasurableSpace.generateFrom P)
  (hP : IsPiSystem P) (hagree : ∀ s ∈ P, μ s = v s) : μ = v := by
```



- (hP : IsPiSystem P): a hypothesis with a name you can use later. [IsProbabilityMeasure μ]: an instance, found automatically. {Ω : Type*}: inferred.

The Infoview: what you have, what you must show

Lean Infoview

▼ Tactic state 1 goal

context

Ω : Type u_1
m : MeasurableSpace Ω
 μ v : Measure Ω
inst+¹ : IsProbabilityMeasure μ
inst⁺ : IsProbabilityMeasure v
P : Set (Set Ω)
hgen : m = MeasurableSpace.generateFrom P
hP : IsPiSystem P
hagree : $\forall s \in P, \mu s = v s$
t : Set Ω
ht : MeasurableSet t
 $\vdash \mu t = v t$

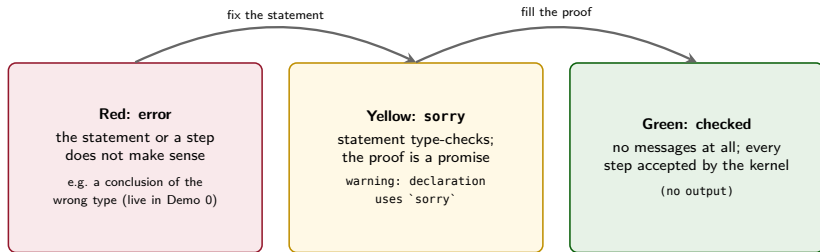
above $\vdash$: what you **have**
one hypothesis per
line, name : type

inst+: a dagger means "exists,
but you cannot type its name"

after $\vdash$: what you must **show**

Mock-up; the text is the real state after `ext t ht in Uniqueness.lean`.

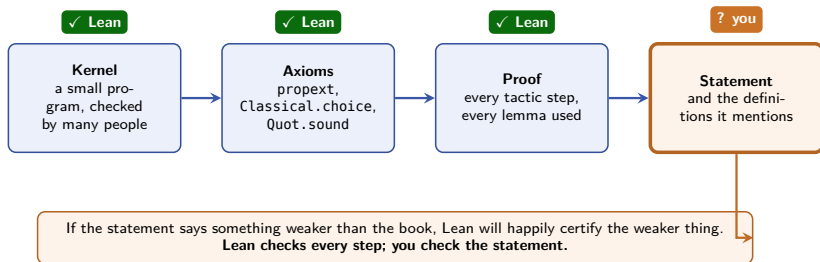
sorry: what “checked” means



A sorry cannot hide

`#print axioms` on a theorem that depends on a `sorry` anywhere upstream prints an axiom list; here the list contains `sorryAx`. Even a failed proof is turned into a hidden `sorryAx`, so the axiom list catches it. The warm-up file starts with every proof equal to `sorry`; your job is to replace them.

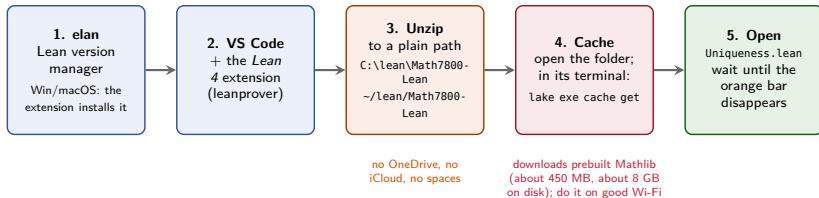
The trust chain



Live example (Sep 2026)

The Lean-checked Navier–Stokes proof is of forced finite-time blow-up (Clay (C)/(D), smooth external force); the unforced problem is open and the proof is not yet peer-reviewed. Lean checked the proof; people check the statement.

Install in five steps (announcement of 9/23)



The one rule people forget

Open the **folder** (the one with lakefile.toml), not a single .lean file. Otherwise Lean reports unknown package Mathlib. Blank Infoview: Command Palette → “Lean 4: Restart Server”. More pitfalls per system: README.md and the backup slides.

Verify your install, or use the browser for now

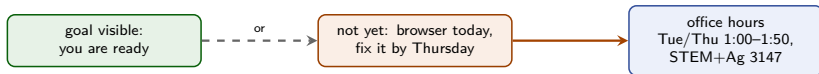
Verify (30 seconds)

1. Open `Math7800Lean/Uniqueness.lean`.
2. Click just after `ext t ht`.
3. The Infoview ends with $\vdash \mu t = v t$.
4. Screenshot it: that is your proof the install works.

Browser fallback

1. Go to `live.lean-lang.org`.
2. Paste the file's contents; wait.
3. Version picker: choose the Stable release preset if offered.

For reading and experimenting only. Check anything you submit on your own install or in office hours.



A dictionary: Billingsley $\leftrightarrow$ Mathlib

Billingsley	Mathlib	Note
sample space Ω	$\Omega : \text{Type}^*$	a type; subsets are $\text{Set } \Omega$
class of subsets $\mathcal{A}$	$\text{Set } (\text{Set } \Omega)$	
σ -field $\mathcal{F}$ on Ω	$m : \text{MeasurableSpace } \Omega$	usually an instance
$A \in \mathcal{F}$	$\text{MeasurableSet } A$	$\text{MeasurableSet}[m]$ A names it
measure μ , value $\mu(A)$	$\mu : \text{Measure } \Omega, \mu A$	values in $\mathbb{R}_{\geq 0}^\infty = [0, \infty]$
probability measure P	$[\text{IsProbabilityMeasure } \mu]$	$\text{measure_univ} : \mu \text{ univ} = 1$

Every Mathlib name here was checked with #check at the course pin. More rows (fields, $\sigma(\mathcal{A})$, $\sqcap$, the lemmas of our proof): backup glossary.

Warm-up 1: inclusion–exclusion, the Mathlib way

```
-- Billingsley §2:  $P(A \cup B) + P(A \cap B) = P(A) + P(B)$ . -/
theorem union_add_inter (μ : Measure Ω) (A B : Set Ω) (hB : MeasurableSet B) :
  μ (A ∪ B) + μ (A ∩ B) = μ A + μ B :=
  measure_union_add_inter A hB
```

Goal at the sorry in the student file:

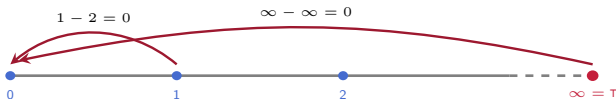
```
Ω : Type u_1
inst† : MeasurableSpace Ω
μ : Measure Ω
A B : Set Ω
hB : MeasurableSet B
⊢ μ (A ∪ B) + μ (A ∩ B) = μ A + μ B
```

Two things to notice

- ▶ Stated with $+$ on both sides, not $P(A \cup B) = P(A) + P(B) - P(A \cap B)$: subtraction in $\mathbb{R}_{\geq 0}$ is truncated (next frame).
- ▶ Only B must be measurable: a measurable set splits every set additively.

Warm-ups 3–5 (measure_mono', prob_le_one', compl_prob): exact? finds each one. You will try one yourself in the demo.

The $\mathbb{R}_{\geq 0}^\infty$ trap: subtraction is truncated



So $\mu(A^c) = \mu(\Omega) - \mu(A)$ needs $\mu(A) \neq \infty$. The general lemma `measure_compl` asks for it; the probability lemma has it built in:

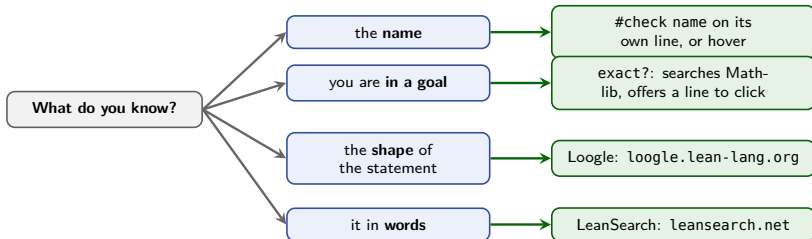
```
| compl t htm ih =>
-- P(A^c) = 1 - P(A): no subtraction on  $\mathbb{R}_{\geq 0}^\infty$  without this lemma
rw [prob_compl_eq_one_sub htm, prob_compl_eq_one_sub htm, ih]
```

Forget it and Lean answers with two leftover goals
($\mu \text{ } t \neq T$, $\forall t \neq T$), not “wrong”. You’ll see this live in
Demo 3.

Rule of thumb

For probabilities use `prob_compl_eq_one_sub`; write
inclusion–exclusion additively.

How to find a lemma



You'll see `exact?` live in Demo 3, and use it yourself on a warm-up.

π -system (paraphrasing Billingsley §3)

A class $\mathcal{P}$ of subsets of Ω is a π -system if it is closed under finite intersections: $A, B \in \mathcal{P} \implies A \cap B \in \mathcal{P}$.

λ -system (paraphrasing Billingsley §3)

A class $\mathcal{L}$ of subsets of Ω is a λ -system if

- (λ_1) $\Omega \in \mathcal{L}$;
- (λ_2) $A \in \mathcal{L} \implies A^c \in \mathcal{L}$;
- (λ_3) $A_1, A_2, \dots \in \mathcal{L}$ pairwise disjoint $\implies \bigcup_n A_n \in \mathcal{L}$.

Generated classes (paraphrasing Billingsley §3)

$\sigma(\mathcal{P})$ and $\lambda(\mathcal{P})$: the smallest σ -field and the smallest λ -system containing $\mathcal{P}$ (the intersection of all of them).

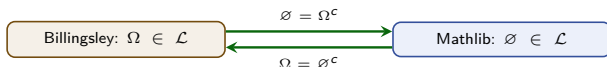
The same definition, twice in Lean

Our midterm (Billingsley's version)

```
/-- λ-system (Billingsley): contains  $\Omega$ , closed under  
complements and  
countable disjoint unions. -/  
structure IsLambdaSys (L : Set (Set  $\Omega$ )) : Prop where  
  univ_mem : univ  $\in$  L  
  compl_mem :  $\forall A \in L, A^c \in L$   
  iUnion_mem :  $\forall f : \mathbb{N} \rightarrow \text{Set } \Omega, \text{Pairwise (Disjoint on } f) \rightarrow$   
     $\hookrightarrow (\forall n, f\ n \in L) \rightarrow$   
     $(\bigsqcup n, f\ n) \in L$ 
```

Mathlib: MeasurableSpace.DynkinSystem

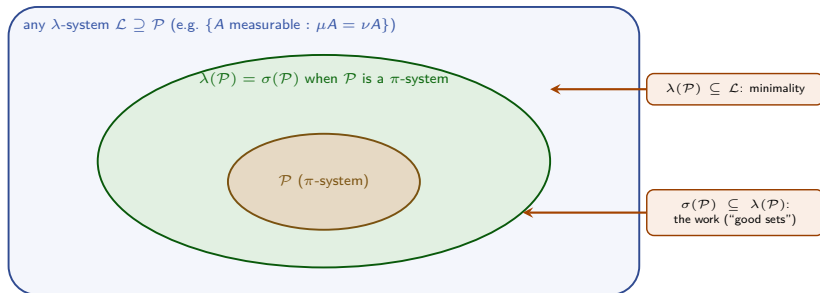
```
has_empty: contains  $\emptyset$   
has_compl: closed under complements  
has_iUnion_nat: countable pairwise disjoint  
unions
```



Also different: Mathlib's IsPiSystem only requires closure when $A \cap B \neq \emptyset$.

Checking the statement includes checking the definitions: Ω vs $\emptyset$, the nonempty clause. This is audit point (1).

The picture: $\mathcal{P} \subseteq \lambda(\mathcal{P}) = \sigma(\mathcal{P}) \subseteq \mathcal{L}$



Every σ -field is a λ -system, so $\lambda(\mathcal{P}) \subseteq \sigma(\mathcal{P})$ is free; the content is the other inclusion.

The π - λ theorem, and how Mathlib packages it

π - λ theorem (paraphrasing Billingsley §3)

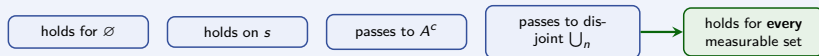
If $\mathcal{P}$ is a π -system and $\mathcal{L}$ is a λ -system with $\mathcal{P} \subseteq \mathcal{L}$, then $\sigma(\mathcal{P}) \subseteq \mathcal{L}$.

Uniqueness (paraphrasing Billingsley §3)

If probability measures μ, ν agree on a π -system $\mathcal{P}$, they agree on $\sigma(\mathcal{P})$.

Mathlib: `MeasurableSpace.induction_on_inter`, an induction principle

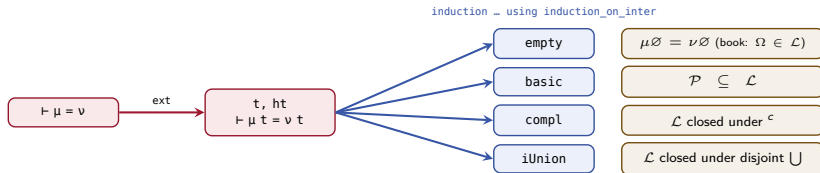
If $m = \sigma(s)$ with s a π -system, and a property C of measurable sets



The four boxes are exactly “ $\{A \text{ measurable} : C(A)\}$ is a (Dynkin) λ -system containing s ”.

Two tactics do the architecture

```
-- To show  $\mu = \nu$  it suffices to check  $\mu t = \nu t$  for every measurable  $t$ .
ext t ht
-- "Let  $L$  be the class of sets where  $\mu$  and  $\nu$  agree;  $L$  is a  $\lambda$ -system  $\supseteq P$ ."
induction t, ht using MeasurableSpace.induction_on_inter hgen hP with
```



$\mathcal{L} = \{A \text{ measurable} : \mu A = \nu A\}$. The one induction line is the π - λ theorem; it leaves four goals.

Cases empty and basic: $\emptyset \in \mathcal{L}$, $\mathcal{P} \subseteq \mathcal{L}$

```
induction t, ht using MeasurableSpace.induction_on_inter hgen hP with
| empty => simp
| basic t ht => exact hagree t ht
| compl t htm ih =>
  --  $P(A^c) = 1 - P(A)$ : no subtraction on  $\mathbb{R}_{\geq 0}^\infty$  without this lemma
  rw [prob_compl_eq_one_sub htm, prob_compl_eq_one_sub htm, ih]
| iUnion f hdisj hfm ih =>
  -- countable additivity on a disjoint union
  rw [measure_iUnion hdisj hfm, measure_iUnion hdisj hfm]
  simp [ih]
```

Goals before simp, and before exact:

```
case empty
[CTX]
t : Set  $\Omega$ 
 $\vdash \mu \emptyset = \nu \emptyset$ 

case basic
[CTX]
t $\vdash$  t : Set  $\Omega$ 
ht : t  $\in$  P
 $\vdash \mu t = \nu t$ 
```

[CTX] = the nine context lines of frame 2b, down to hagree.

Paper (paraphrasing Billingsley §3): $\mu\Omega = 1 = \nu\Omega$, and $\mathcal{P} \subseteq \mathcal{L}$ by hypothesis. Mathlib's form checks $\emptyset$: $\mu\emptyset = 0 = \nu\emptyset$.

simp rewrites with library lemmas; here measure_empty : $\mu \emptyset = 0$ on both sides. hagree is a function: feed it a set t and a proof ht : t $\in$ P, get $\mu t = \nu t$. exact says “this term is the proof”.

Case compl: $\mathcal{L}$ is closed under complements

```
induction t, ht using MeasurableSpace.induction_on_inter hgen hP with
| empty => simp
| basic t ht => exact hagree t ht
| compl t htm ih =>
  --  $P(A^c) = 1 - P(A)$ : no subtraction on  $\mathbb{R}_{\geq 0}^\infty$  without this lemma
  rw [prob_compl_eq_one_sub htm, prob_compl_eq_one_sub htm, ih]
| iUnion f hdisj hfm ih =>
  -- countable additivity on a disjoint union
  rw [measure_iUnion hdisj hfm, measure_iUnion hdisj hfm]
  simp [ih]
```

Goal before, then after each rewrite:

```
case compl
[CTX]
t† t : Set  $\Omega$ 
htm : MeasurableSet t
ih :  $\mu t = \nu t$ 
 $\vdash \mu t^c = \nu t^c$ 
 $\vdash 1 - \mu t = \nu t^c$ 
 $\vdash 1 - \mu t = 1 - \nu t$ 
```

Paper (paraphrasing Billingsley §3): if $A \in \mathcal{L}$ then $\mu A^c = 1 - \mu A = 1 - \nu A = \nu A^c$.

rw [a, b, c] is three rewrites. The third (ih) makes both sides $1 - \nu t$, and rw closes an identical goal by itself. ih: $\mu t = \nu t$ is the induction hypothesis: C already holds for t ; we show it for t^c .

Case iUnion: countable disjoint unions

```
induction t, ht using MeasurableSpace.induction_on_inter hgen hP with
| empty => simp
| basic t ht => exact hagree t ht
| compl t htm ih =>
  --  $P(A^c) = 1 - P(A)$ : no subtraction on  $\mathbb{R}_{\geq 0}^\infty$  without this lemma
  rw [prob_compl_eq_one_sub htm, prob_compl_eq_one_sub htm, ih]
| iUnion f hdisj hfm ih =>
  -- countable additivity on a disjoint union
  rw [measure_iUnion hdisj hfm, measure_iUnion hdisj hfm]
  simp [ih]
```

Goal before, then after each rewrite:

```
case iUnion
[CTX]
t : Set Ω
f : ℕ → Set Ω
hdisj : Pairwise (Function.onFun Disjoint f)
hfm : ∀ (i : ℕ), MeasurableSet (f i)
ih : ∀ (i : ℕ), μ (f i) = ν (f i)
⊢ μ (⊔ i, f i) = ν (⊔ i, f i)
⊢ ∑' (i : ℕ), μ (f i) = ν (⊔ i, f i)
⊢ ∑' (i : ℕ), μ (f i) = ∑' (i : ℕ), ν (f i)
```

context

Paper (paraphrasing Billingsley §3): if $A_n \in \mathcal{L}$ are disjoint, $\mu(\bigcup A_n) = \sum \mu A_n = \sum \nu A_n = \nu(\bigcup A_n)$.

`measure_iUnion` is countable additivity; `hdisj`, `hfm` are its hypotheses. `simp [ih]` rewrites inside the sum, under the bound `i`, where `rw` cannot.

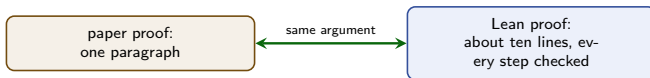
Payoff: a law on $\mathbb{R}$ is determined by its CDF

```
/-- Payoff: a law on  $\mathbb{R}$  is determined by its CDF. -/
theorem law_eq_of_cdf_eq (μ v : Measure ℝ) [IsProbabilityMeasure μ]
  [IsProbabilityMeasure v] (h : ∀ x, μ (Iic x) = v (Iic x)) : μ = v := by
  refine uniqueness_of_extension μ v (range Iic) ?_ isPiSystem_Iic ?_
  · exact BorelSpace.measurable_eq.trans (borel_eq_generateFrom_Iic ℝ)
  · rintro _ (x, rfl)
    exact h x
```

Plug $\mathcal{P} = \text{half-lines } (-\infty, x]$ into our theorem; two small facts remain.

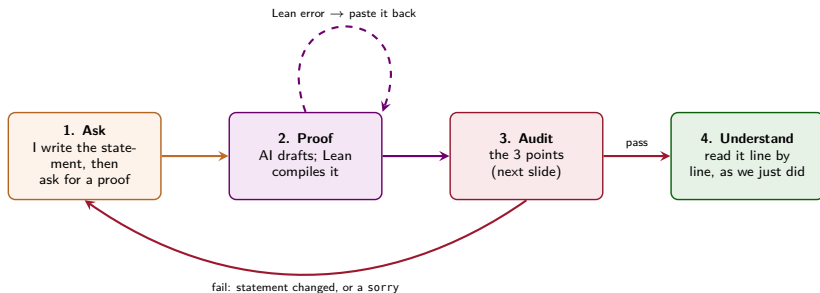
What we just did

Case	Paper (Billingsley §3)	Lean
empty	$\Omega \in \mathcal{L}$ (Mathlib: $\emptyset$)	<code>simp</code> (uses <code>measure_empty</code>)
basic	$\mathcal{P} \subseteq \mathcal{L}$ by hypothesis	<code>exact hagree t ht</code>
compl	$\mu A^c = 1 - \mu A = 1 - \nu A = \nu A^c$	<code>rw [prob_compl_eq_one_sub htm, ..., ih]</code>
iUnion	$\mu \bigcup A_n = \sum \mu A_n = \sum \nu A_n$	<code>rw [measure_iUnion ...]; simp [ih]</code>



Questions? (pause here)

How I actually work



Why step 4 is not optional

Lean has already told you the proof is *correct*. Reading it is for *you*: the oral check asks you to explain each step, and the next proof you write will reuse these moves.

The 3-point audit

(1) The statement is the one you meant. Compare it, character by character, with what you asked for: hypotheses added or strengthened, conclusion weakened ($\leq$ for $=$, $\exists$ for $\forall$), a changed definition.

you

(2) No sorry. The file compiles with no declaration uses `sorry` warning (also look for `admit`, `new axiom`).

Lean

(3) #print axioms is clean. Only `propext`, `Classical.choice`, `Quot.sound` (a subset is fine); `sorryAx` or any other name fails.

Lean

Lean checks (2) and (3) for you; only you can check (1). The midterm checker runs all three, and point (1) exactly: your theorem must type-check against the frozen statement.

Exhibit: the AI changed a hypothesis

```
-- The AI's version, compiled: the statement was changed. --
theorem uniqueness_AI {Ω : Type *} [m : MeasurableSpace Ω]
  (μ v : Measure Ω) [IsProbabilityMeasure μ] [IsProbabilityMeasure v]
  (P : Set (Set Ω)) (hP : ∀ s, MeasurableSet s → s ∈ P)
  (hagree : ∀ s ∈ P, μ s = v s) : μ = v := by
  ext t ht
  exact hagree t (hP t ht)

#print axioms uniqueness_AI
-- 'uniqueness_AI' depends on axioms:
--   [propext, Classical.choice, Quot.sound]
```

Real statement

hgen : m = generateFrom P
hP : IsPiSystem P

(2) ✓ (3) ✓ (1) ✓

AI edit

AI statement

hP : ∀ s, MeasurableSet s → s ∈ P
P already holds every measurable set

(2) ✓ (3) ✓ (1) ✗: trivial, useless for CDFs

Tools, and what Lean cannot tell you

Tools I use

- ▶ `exact?`, `#check`, `hover`: inside VS Code.
- ▶ Loogle, LeanSearch: find a Mathlib name.
- ▶ An AI chat: drafts, and explains error messages.
- ▶ Lean Zulip: real people, very helpful to beginners.

Lean cannot tell you

- ▶ whether the statement matches the book;
- ▶ whether the definitions match the book;
- ▶ whether the result is interesting;
- ▶ whether *you* understand it.

Honesty: using AI is allowed in this course. Say how you used it. The write-up is in your own words, and the oral check is you explaining your proof.

Take-home midterm: two Lean projects

Project A: the π - λ theorem

```
/-- Project A: Dynkin's  $\pi$ - $\lambda$  theorem (Billingsley §3). -/  
theorem pi_lambda (P L : Set (Set  $\Omega$ )) (hP : IsPiSys P) (hL : IsLambdaSys L)  
  (hPL :  $P \subseteq L$ ) : sigmaGen P  $\subseteq$  L := by
```

If $\mathcal{P}$ is a π -system, $\mathcal{L}$ a λ -system (Billingsley's, with Ω) and $\mathcal{P} \subseteq \mathcal{L}$, then $\sigma(\mathcal{P}) \subseteq \mathcal{L}$. Scaffold: four lemmas with sorry.

Project B: the monotone class theorem (not in Mathlib)

```
/-- Project B: monotone class theorem (Billingsley §3). -/  
theorem monotone_class (F $_{\sigma}$  : Set (Set  $\Omega$ )) (hF $_{\sigma}$  : IsField F $_{\sigma}$ ) :  
  monoGen F $_{\sigma}$  = sigmaGen F $_{\sigma}$  := by
```

If $\mathcal{F}_0$ is a field, the monotone class it generates equals the σ -field it generates: $m(\mathcal{F}_0) = \sigma(\mathcal{F}_0)$.

Statements are frozen: point (1) is enforced by the checker.

Part 1: Thu 10/1, no class — self-study (video + FPIL Ch.1)

Part 2: A/B released today, end of class

due Thu 10/29

Gradescope: two .lean files + one PDF

Rules, grading, and where to get help

Rules

- ▶ Learn with AI however you like (ChatGPT, Claude, ...): what matters is that you can read and write Lean yourself.
- ▶ PDF write-up: every step in your own words, plus an honest self-assessment (see below).
- ▶ **10-minute oral check** on your proofs.
- ▶ Project A may **not** use anything in `MeasurableSpace.DynkinSystem.*` or `MeasurableSpace.generateFrom*`, nor `MeasurableSpace.induction_on_inter`.

Checker

compiles; statement matches; axioms clean; no forbidden lemma

default 50%

Write-up

your own words

default 25%

Oral

10 minutes

default 25%

Help

- ▶ Office hours Tue/Thu 1:00–1:50, STEM+Ag 3147.
- ▶ Lean Zulip: leanprover.zulipchat.com
- ▶ CHEATSHEET.md, Mathlib docs, Loogle.

Self-assessment (end of write-up, our 100-pt scale): *if you were the professor, how satisfied would you be with yourself as a student?* Le's own grade today: 50/100.

Backup: the paper proof of the π - λ theorem

Let $\mathcal{P}$ be a π -system and $\mathcal{L} \supseteq \mathcal{P}$ a λ -system (Billingsley's argument, paraphrased).

1. $\lambda(\mathcal{P})$ is a λ -system containing $\mathcal{P}$, and $\lambda(\mathcal{P}) \subseteq \mathcal{L}$ by minimality.
2. A λ -system contains $\emptyset = \Omega^c$ and is closed under finite disjoint unions (pad with $\emptyset$).
3. **Good sets.** For $A \in \lambda(\mathcal{P})$ let $\mathcal{G}_A = \{B : A \cap B \in \lambda(\mathcal{P})\}$. Then $\mathcal{G}_A$ is a λ -system: $\Omega \in \mathcal{G}_A$; if $B \in \mathcal{G}_A$ then $A \cap B^c = (A^c \cup (A \cap B))^c$, a complement of a disjoint union; disjoint unions pass through $A \cap \cdot$.
4. **Round 1.** For $A \in \mathcal{P}$: $\mathcal{P} \subseteq \mathcal{G}_A$ (π -system), so $\lambda(\mathcal{P}) \subseteq \mathcal{G}_A$. **Round 2.** For $A \in \lambda(\mathcal{P})$: round 1 gives $\mathcal{P} \subseteq \mathcal{G}_A$, so again $\lambda(\mathcal{P}) \subseteq \mathcal{G}_A$. Hence $\lambda(\mathcal{P})$ is closed under $\cap$.
5. A λ -system closed under $\cap$ is a σ -field: write $\bigcup A_n = \bigcup B_n$ with $B_n = A_n \cap A_1^c \cap \cdots \cap A_{n-1}^c$ disjoint.
6. So $\lambda(\mathcal{P})$ is a σ -field containing $\mathcal{P}$: $\sigma(\mathcal{P}) \subseteq \lambda(\mathcal{P}) \subseteq \mathcal{L}$.

Project A's four scaffold lemmas cover steps 1 (two lemmas), 3–4 and 5; step 6 is the main theorem.

Backup: tactic cheat sheet

Tactic	What it does
intro	introduce a variable or hypothesis for a $\forall$ / $\rightarrow$ goal
exact	close the goal with a term that matches it
apply	match the lemma's conclusion; its hypotheses become goals
refine	like exact, with <code>?_</code> holes left as goals
rw [h]	rewrite the goal left-to-right with an equation
simp	simplify with the library's <code>@[simp]</code> lemmas
constructor	split $\wedge$, Iff , or a structure into its fields
obtain, rintro	take apart $\exists$, $\wedge$, $\{_, _\}$
ext	equality of measures/sets/functions, pointwise
induction ... using	induction with a custom principle
calc	chain equalities and inequalities
exact?	search Mathlib for a closing lemma
sorry	placeholder; the file compiles with a warning

Full version with examples: `student/CHEATSHEET.md`.

Backup: error catalogue (all systems)

Symptom	Fix
lake: command not found	restart VS Code or the terminal after installing elan
Mathlib builds for hours	you ran lake build first; Ctrl+C, then lake exe cache get
Windows: "filename too long"	git config --global core.longpaths true
project in OneDrive / iCloud / path with spaces	move the folder to C:\lean\Math7800-Lean or ~/\lean/Math7800-Lean
unknown package Mathlib	open the <i>folder</i> with lakefile.toml, not a single file
Infoview blank	Command Palette: "Lean 4: Restart Server"
Infoview stuck on one file	Command Palette: "Lean 4: Restart File"
macOS: tools missing	xcode-select --install

Backup: more of the glossary

Billingsley	Mathlib	Note
field (algebra) of sets	(course) <code>IsField</code>	midterm only; not in Mathlib
$\sigma(\mathcal{A})$	<code>MeasurableSpace.generateFrom</code>	
$\mathcal{F}_1 \cap \mathcal{F}_2$	<code>m1 ∩ m2</code>	"inf" in a lattice
$\mathcal{F}_1 \subseteq \mathcal{F}_2$	<code>m1 ≤ m2</code>	
π -system	<code>IsPiSystem</code>	only when $A \cap B \neq \emptyset$
λ -system	<code>MeasurableSpace.DynkinSystem</code>	contains $\emptyset$, not Ω
π - λ theorem	<code>induction_on_inter</code>	an induction principle
finite measure	<code>[IsFiniteMeasure μ]</code>	automatic for probability
$\mu(A^c) = \mu(\Omega) - \mu(A)$	<code>measure_compl</code>	needs $\mu \ A \neq \top$
$P(A^c) = 1 - P(A)$	<code>prob_compl_eq_one_sub</code>	needs <code>MeasurableSet A</code>
$\mu(A) < \infty$, finite μ	<code>measure_ne_top</code>	
countable additivity	<code>measure_iUnion</code>	sum is $\sum'$
$A \subseteq B \Rightarrow \mu A \leq \mu B$	<code>measure_mono</code>	no measurability
countable subadditivity	<code>measure_iUnion_le</code>	
$\mu = \nu$ iff equal on $\mathcal{F}$ (iff form)	<code>Measure.ext</code> <code>Measure.ext_iff</code>	applied by ext
Borel sets from $(-\infty, x]$	<code>borel_eq_generateFrom_Iic</code>	
distribution function F	<code>ProbabilityTheory.cdf μ</code>	
image measure $\mu \circ f^{-1}$	<code>Measure.map f μ</code>	

Backup: top two pitfalls on each system

Windows

1. Project in OneDrive, Desktop, or a path with spaces.
Fix: move it to `C:\lean\Math7800-Lean`.
2. "filename too long".
Fix: `git config --global core.longpaths true`

macOS

1. Command-line tools missing.
Fix: `xcode-select --install` first.
2. Project inside iCloud Drive (synced folder).
Fix: use `~/lean/Math7800-Lean`.

Linux

1. `lake`: command not found.
Fix: open a new terminal (or restart VS Code) after `elan`.
2. Ran `lake build` first: `Mathlib` builds for hours.
Fix: `Ctrl+C`, then `lake exe cache get`.

Blank Infoview on any system: Command Palette → "Lean 4: Restart Server". Full table: `README.md`, and the backup slide "Error catalogue".

Backup: Warm-up 2, a pair of proofs, $\langle h_1, h_2 \rangle$

```
-- The intersection of two  $\sigma$ -fields is a  $\sigma$ -field (Lean: ' $m_1 \sqcap m_2$ '). -/  
theorem inf_measurableSet (m1 m2 : MeasurableSpace  $\Omega$ ) (s : Set  $\Omega$ )  
  (h1 : MeasurableSet[m1] s) (h2 : MeasurableSet[m2] s) :  
  MeasurableSet[m1  $\sqcap$  m2] s :=  
   $\langle h_1, h_2 \rangle$ 
```

