Why I am doing L-series in Lean

Michael Stoll
Universität Bayreuth

Rutgers Lean Seminar
March 27, 2024

Where I Come From

Where I Come From

- Some early interest in ATP
  (wrote a resolution prover in Scheme as an undergrad)

Where I Come From

- Some early interest in ATP
  (wrote a resolution prover in Scheme as an undergrad)

- Took some courses in Mathematical Logic (e.g., on proof theory)

Where I Come From

- Some early interest in ATP
  (wrote a resolution prover in Scheme as an undergrad)

- Took some courses in Mathematical Logic (e.g., on proof theory)

- Played a bit with Coq in 2016

```
Theorem pyth_triples_relprime_even_pos :
  forall x y z : Z, Zeven x -> z >= 0 -> rel_prime x y -> x^2 + y^2 = z^2
    -> exists r s : Z, rel_prime r s /\ x = 2*r*s /\ y = r^2-s^2 /\ z = r^2+s^2.
```

Where I Come From

- Some early interest in ATP
  (wrote a resolution prover in Scheme as an undergrad)

- Took some courses in Mathematical Logic (e.g., on proof theory)

- Played a bit with Coq in 2016

```
Theorem pyth_triples_relprime_even_pos :
  forall x y z : Z, Zeven x -> z >= 0 -> rel_prime x y -> x^2 + y^2 = z^2
    -> exists r s : Z, rel_prime r s /\ x = 2*r*s /\ y = r^2-s^2 /\ z = r^2+s^2.
```

- Got interested in Lean through watching one of Kevin's talks (IIRC)
  and got addicted (ca. Feb. 2022)

Motivation

Motivation

Corollary 9.10. *Suppose that C/k is a smooth projective curve of genus 2 given by an integral Weierstrass model $\mathcal{C}$ such that there are three nodes in the special fiber of $\mathcal{C}$. We say that $\mathcal{C}$ is split if the two components A and E of the special fiber of $\mathcal{C}^{\min}$ are defined over $\mathfrak{k}$; otherwise $\mathcal{C}$ is nonsplit. Let $v(\Delta) = m_1 + m_2 + m_3$ as above and set $M = m_1m_2 + m_1m_3 + m_2m_3$.*

⋮

(c) *If two of the nodes lie in a quadratic extension of $\mathfrak{k}$ and are conjugate over $\mathfrak{k}$ and one is $\mathfrak{k}$-rational, then*

$$\beta = \begin{cases} \dfrac{m_1}{M} \max\left\{ \left\lfloor \dfrac{m_1^2}{2} \right\rfloor + m_1m_3, \left\lfloor \dfrac{m_3^2}{2} \right\rfloor + m_1 \left\lfloor \dfrac{m_3}{2} \right\rfloor \right\} & \text{if } \mathcal{C} \text{ is split,} \\ \dfrac{m_1}{2} & \text{if } \mathcal{C} \text{ is nonsplit and } m_1 \text{ is even,} \\ 0 & \text{otherwise,} \end{cases}$$

where m_3 corresponds to the rational node (and $m_1 = m_2$).

Motivation

Proof. The proof of (a) follows easily from Proposition 9.4.

For the other cases, note that in the nonsplit case some power of Frobenius acts as negation on the component group $\Phi(\bar{\mathfrak{k}})$, so the only elements of $\Phi(\mathfrak{k})$ are elements of order 2 in $\Phi(\bar{\mathfrak{k}})$, which correspond to $[B_{m_1/2} - C_{m_2/2}]$ if m_1 and m_2 are even $\big($where μ takes the value $\frac{1}{4}(m_1 + m_2)\big)$, and similarly with the obvious cyclic permutations.

In the situation of (c), we must have $m_1 = m_2$. If $P = [(P_1) - (P_2)] \in J(k)$ and $P_1 \in C(\bar{k})$ maps to one of the conjugate nodes, then P_2 must map to the other, so all $P \in J(k)$ must map to a component of the form $[B_i - C_j]$ or $[D_i - D_j]$. Now the result in the split case follows from a case distinction depending on whether $m_1 \leq m_3$ or not. In the nonsplit case, the only element of order 2 that is defined over $\mathfrak{k}$ is $[B_{m_1/2} - C_{m_1/2}]$ if it exists.

In the situation of (d), the group $\Phi(\mathfrak{k})$ is of order 3 (generated by $[E - A]$) in the split case and trivial in the nonsplit case. □

Motivation

Proof. The proof of (a) follows easily from Proposition 9.4.

For the other cases, note that in the nonsplit case some power of Frobenius acts as negation on the component group $\Phi(\bar{\mathfrak{k}})$, so the only elements of $\Phi(\mathfrak{k})$ are elements of order 2 in $\Phi(\bar{\mathfrak{k}})$, which correspond to $[B_{m_1/2} - C_{m_2/2}]$ if m_1 and m_2 are even (where μ takes the value $\frac{1}{4}(m_1 + m_2)$), and similarly with the obvious cyclic permutations.

In the situation of (c), we must have $m_1 = m_2$. If $P = [(P_1) - (P_2)] \in J(k)$ and $P_1 \in C(\bar{k})$ maps to one of the conjugate nodes, then P_2 must map to the other, so all $P \in J(k)$ must map to a component of the form $[B_i - C_j]$ or $[D_i - D_j]$. Now the result in the split case follows from a case distinction depending on whether $m_1 \leq m_3$ or not. In the nonsplit case, the only element of order 2 that is defined over $\mathfrak{k}$ is $[B_{m_1/2} - C_{m_1/2}]$ if it exists.

In the situation of (d), the group $\Phi(\mathfrak{k})$ is of order 3 (generated by $[E - A]$) in the split case and trivial in the nonsplit case. □

Motivation

Motivation

There are actually two mistakes in the statement and proof (but one is not visible here).

Motivation

There are actually two mistakes in the statement and proof
(but one is not visible here).

It would be nice to be able to avoid such mistakes!

Motivation

There are actually two mistakes in the statement and proof (but one is not visible here).

It would be nice to be able to avoid such mistakes!

Goal: Be able to formalize my papers!

Motivation

There are actually two mistakes in the statement and proof (but one is not visible here).

It would be nice to be able to avoid such mistakes!

Goal: Be able to formalize my papers!

Problem: Lean+Mathlib is very far away from this.

Motivation

There are actually two mistakes in the statement and proof (but one is not visible here).

It would be nice to be able to avoid such mistakes!

Goal: Be able to formalize my papers!

Problem: Lean+Mathlib is very far away from this.

(**But:** See `https://github.com/MichaelStollBayreuth/Weights`)

Motivation

There are actually two mistakes in the statement and proof (but one is not visible here).

It would be nice to be able to avoid such mistakes!

Goal: Be able to formalize my papers!

Problem: Lean+Mathlib is very far away from this.

(**But:** See `https://github.com/MichaelStollBayreuth/Weights`)

New Goal: Teach more number theory to Lean!

Motivation

There are actually two mistakes in the statement and proof (but one is not visible here).

It would be nice to be able to avoid such mistakes!

Goal: Be able to formalize my papers!

Problem: Lean+Mathlib is very far away from this.

(**But:** See `https://github.com/MichaelStollBayreuth/Weights`)

New Goal: Teach more number theory to Lean!

For example: Get the Hasse-Minkowski Theorem into Mathlib!

Hasse-Minkowski

Hasse-Minkowski

Theorem.
Let $Q(x_1, \ldots, x_n)$ be a non-degenerate quadratic form over $\mathbb{Q}$.
If Q has nontrivial zeros in all completions of $\mathbb{Q}$, then also in $\mathbb{Q}$.

Hasse-Minkowski

Theorem.
Let $Q(x_1, \ldots, x_n)$ be a non-degenerate quadratic form over $\mathbb{Q}$.
If Q has nontrivial zeros in all completions of $\mathbb{Q}$, then also in $\mathbb{Q}$.

What we need:

Hasse-Minkowski

Theorem.
Let $Q(x_1, \ldots, x_n)$ be a non-degenerate quadratic form over $\mathbb{Q}$.
If Q has nontrivial zeros in all completions of $\mathbb{Q}$, then also in $\mathbb{Q}$.

What we need:

➊ Legendre's Theorem on ternary quadratic forms

Hasse-Minkowski

Theorem.
Let $Q(x_1, \ldots, x_n)$ be a non-degenerate quadratic form over $\mathbb{Q}$.
If Q has nontrivial zeros in all completions of $\mathbb{Q}$, then also in $\mathbb{Q}$.

What we need:

➊ Legendre's Theorem on ternary quadratic forms ✔

Hasse-Minkowski

Theorem.
Let $Q(x_1, \ldots, x_n)$ be a non-degenerate quadratic form over $\mathbb{Q}$.
If Q has nontrivial zeros in all completions of $\mathbb{Q}$, then also in $\mathbb{Q}$.

What we need:

❶ Legendre's Theorem on ternary quadratic forms ✔

❷ The product formula for the quadratic Hilbert symbol

Hasse-Minkowski

Theorem.
Let $Q(x_1, \ldots, x_n)$ be a non-degenerate quadratic form over $\mathbb{Q}$.
If Q has nontrivial zeros in all completions of $\mathbb{Q}$, then also in $\mathbb{Q}$.

What we need:

❶ Legendre's Theorem on ternary quadratic forms ✔

❷ The product formula for the quadratic Hilbert symbol

❸ Dirichlet's Theorem on primes in arithmetic progression

Hasse-Minkowski

Theorem.
Let $Q(x_1, \ldots, x_n)$ be a non-degenerate quadratic form over $\mathbb{Q}$.
If Q has nontrivial zeros in all completions of $\mathbb{Q}$, then also in $\mathbb{Q}$.

What we need:

❶ Legendre's Theorem on ternary quadratic forms ✔

❷ The product formula for the quadratic Hilbert symbol

❸ Dirichlet's Theorem on primes in arithmetic progression

❷ and ❸ are needed to go from $n = 3$ to $n = 4$ ($n \leq 2$ is easy).

Hasse-Minkowski

Theorem.
Let $Q(x_1, \ldots, x_n)$ be a non-degenerate quadratic form over $\mathbb{Q}$.
If Q has nontrivial zeros in all completions of $\mathbb{Q}$, then also in $\mathbb{Q}$.

What we need:

➊ Legendre's Theorem on ternary quadratic forms ✔

➋ The product formula for the quadratic Hilbert symbol

➌ Dirichlet's Theorem on primes in arithmetic progression

➋ and ➌ are needed to go from $n = 3$ to $n = 4$ ($n \leq 2$ is easy).

I did some work on ➋ as my first larger Lean(3) project.

Dirichlet

Dirichlet

So let us focus on Dirichlet's Theorem!

Dirichlet

So let us focus on Dirichlet's Theorem!

Goal: Formalize the proof via Dirichlet L-series.

Dirichlet

So let us focus on Dirichlet's Theorem!

Goal: Formalize the proof via Dirichlet L-series.

What we need:

Dirichlet

So let us focus on Dirichlet's Theorem!

Goal: Formalize the proof via Dirichlet L-series.

What we need:

➊ Euler product for L-series of multiplicative functions ✔

Dirichlet

So let us focus on Dirichlet's Theorem!

Goal: Formalize the proof via Dirichlet L-series.

What we need:

❶ Euler product for L-series of multiplicative functions ✔

❷ Holomorphic continuation of Dirichlet L-series (D. Loeffler)

Dirichlet

So let us focus on Dirichlet's Theorem!

Goal: Formalize the proof via Dirichlet L-series.

What we need:

❶ Euler product for L-series of multiplicative functions ✔

❷ Holomorphic continuation of Dirichlet L-series (D. Loeffler)

❸ Non-vanishing of $L(\chi, 1)$

Dirichlet

So let us focus on Dirichlet's Theorem!

Goal: Formalize the proof via Dirichlet L-series.

What we need:

❶ Euler product for L-series of multiplicative functions ✔

❷ Holomorphic continuation of Dirichlet L-series (D. Loeffler)

❸ Non-vanishing of $L(\chi, 1)$

❹ Some limits and asymptotics to glue things

PNT+

PNT+

This plan has now been subsumed
as part of the PrimeNumberTheorem+ project,
which aims at proving the Prime Number Theorem and various extensions
(Dirichlet, Chebotarëv, . . .), even with good error terms!

PNT+

This plan has now been subsumed
as part of the PrimeNumberTheorem+ project,
which aims at proving the Prime Number Theorem and various extensions
(Dirichlet, Chebotarëv, . . .), even with good error terms!

In this context, I have formalized what is necessary to reduce PNT
to some version of the Wiener-Ikehara Theorem:

PNT+

This plan has now been subsumed
as part of the PrimeNumberTheorem+ project,
which aims at proving the Prime Number Theorem and various extensions
(Dirichlet, Chebotarëv, . . .), even with good error terms!

In this context, I have formalized what is necessary to reduce PNT
to some version of the Wiener-Ikehara Theorem:

```
open Filter Topology Nat in
/-- A version of the *Wiener-Ikehara Tauberian Theorem*: If `f` is a nonnegative arithmetic
function whose L-series has a simple pole at `s = 1` with residue `A` and otherwise extends
continuously to the closed half-plane `re s ≥ 1`, then `∑ n < N, f n` is asymptotic to `A*N`. -/
def WienerIkeharaTheorem : Prop :=
  ∀ {f : ℕ → ℝ} {A : ℝ} {F : ℂ → ℂ}, (∀ n, 0 ≤ f n) →
    Set.EqOn F (fun s ↦ L ↗f s - A / (s - 1)) {s | 1 < s.re} →
    ContinuousOn F {s | 1 ≤ s.re} →
    Tendsto (fun N : ℕ ↦ ((Finset.range N).sum f) / N) atTop (𝓝 A)
```

Non-vanishing

Non-vanishing

```
/-- For positive `x` and nonzero `y` we have that
$|L(\chi^0, x)^3 \cdot L(\chi, x+iy)^4 \cdot L(\chi^2, x+2iy)| \ge 1$. -/
lemma norm_dirichlet_product_ge_one {N : ℕ} (χ : DirichletCharacter ℂ N) {x : ℝ} (hx : 0 < x)
    (y : ℝ) :
    ‖L ↗(1 : DirichletCharacter ℂ N) (1 + x) ^ 3 * L ↗χ (1 + x + I * y) ^ 4 *
      L ↗(χ ^ 2 :) (1 + x + 2 * I * y)‖ ≥ 1 := by
```

Non-vanishing

```
/-- For positive `x` and nonzero `y` we have that
$|L(\chi^0, x)^3 \cdot L(\chi, x+iy)^4 \cdot L(\chi^2, x+2iy)| \ge 1$. -/
lemma norm_dirichlet_product_ge_one {N : ℕ} (χ : DirichletCharacter ℂ N) {x : ℝ} (hx : 0 < x)
    (y : ℝ) :
    ‖L ↗(1 : DirichletCharacter ℂ N) (1 + x) ^ 3 * L ↗χ (1 + x + I * y) ^ 4 *
      L ↗(χ ^ 2 :) (1 + x + 2 * I * y)‖ ≥ 1 := by


/-- For positive `x` and nonzero `y` we have that
$|\zeta(x)^3 \cdot \zeta(x+iy)^4 \cdot \zeta(x+2iy)| \ge 1$. -/
lemma norm_zeta_product_ge_one {x : ℝ} (hx : 0 < x) (y : ℝ) :
    ‖ζ (1 + x) ^ 3 * ζ (1 + x + I * y) ^ 4 * ζ (1 + x + 2 * I * y)‖ ≥ 1 := by
  have ⟨h₀, h₁, h₂⟩ := one_lt_re_of_pos y hx
  simpa only [one_pow, norm_mul, norm_pow, DirichletCharacter.LSeries_modOne_eq,
    LSeries_one_eq_riemannZeta, h₀, h₁, h₂] using norm_dirichlet_product_ge_one χ₁ hx y
```

Non-vanishing

```
/-- For positive `x` and nonzero `y` we have that
$|L(\chi^0, x)^3 \cdot L(\chi, x+iy)^4 \cdot L(\chi^2, x+2iy)| \ge 1$. -/
lemma norm_dirichlet_product_ge_one {N : ℕ} (χ : DirichletCharacter ℂ N) {x : ℝ} (hx : 0 < x)
    (y : ℝ) :
    ‖L ↗(1 : DirichletCharacter ℂ N) (1 + x) ^ 3 * L ↗χ (1 + x + I * y) ^ 4 *
      L ↗(χ ^ 2 :) (1 + x + 2 * I * y)‖ ≥ 1 := by


/-- For positive `x` and nonzero `y` we have that
$|\zeta(x)^3 \cdot \zeta(x+iy)^4 \cdot \zeta(x+2iy)| \ge 1$. -/
lemma norm_zeta_product_ge_one {x : ℝ} (hx : 0 < x) (y : ℝ) :
    ‖ζ (1 + x) ^ 3 * ζ (1 + x + I * y) ^ 4 * ζ (1 + x + 2 * I * y)‖ ≥ 1 := by
  have ⟨h₀, h₁, h₂⟩ := one_lt_re_of_pos y hx
  simpa only [one_pow, norm_mul, norm_pow, DirichletCharacter.LSeries_modOne_eq,
    LSeries_one_eq_riemannZeta, h₀, h₁, h₂] using norm_dirichlet_product_ge_one χ₁ hx y


/-- The Riemann Zeta Function does not vanish on the closed half-plane `re z ≥ 1`. -/
lemma riemannZeta_ne_zero_of_one_le_re ⦃z : ℂ⦄ (hz : z ≠ 1) (hz' : 1 ≤ z.re) : ζ z ≠ 0 := by
```

Deduction from WIT

Deduction from WIT

```
/-- The function obtained by "multiplying away" the pole of `ζ`. Its (negative) logarithmic
derivative is the function used in the Wiener-Ikehara Theorem to prove the Prime Number
Theorem. -/
noncomputable def ζ₁ : ℂ → ℂ := Function.update (fun z ↦ ζ z * (z - 1)) 1 1
```

Deduction from WIT

```
/-- The function obtained by "multiplying away" the pole of `ζ`. Its (negative) logarithmic
derivative is the function used in the Wiener-Ikehara Theorem to prove the Prime Number
Theorem. -/
noncomputable def ζ₁ : ℂ → ℂ := Function.update (fun z ↦ ζ z * (z - 1)) 1 1
```

```
open Filter Nat ArithmeticFunction in
/-- The *Wiener-Ikehara Theorem* implies the *Prime Number Theorem* in the form that
`ψ x ∼ x`, where `ψ x = ∑ n < x, Λ n` and `Λ` is the von Mangoldt function. -/
theorem PNT_vonMangoldt (WIT : WienerIkeharaTheorem) :
    Tendsto (fun N : ℕ ↦ ((Finset.range N).sum Λ) / N) atTop (nhds 1) := by
  have hnv := riemannZeta_ne_zero_of_one_le_re
  refine WIT (F := fun z ↦ -deriv ζ₁ z / ζ₁ z) (fun _ ↦ vonMangoldt_nonneg) (fun s hs ↦ ?_) ?_
  · have hs₁ : s ≠ 1 := by
      rintro rfl
      simp at hs
    simp only [ne_eq, hs₁, not_false_eq_true, LSeries_vonMangoldt_eq_deriv_riemannZeta_div hs,
      ofReal_one]
    exact neg_logDeriv_ζ₁_eq hs₁ <| hnv hs₁ (Set.mem_setOf.mp hs).le
  · refine continuousOn_neg_logDeriv_ζ₁.mono fun s _ ↦ ?_
    specialize @hnv s
    simp at *
    tauto
```

L-Series

L-Series

When I started looking at this,
there was a rudimentary L-series package in Mathlib doing roughly this:

```
def LSeries (f : ArithmeticFunction ℂ) (s : ℂ) : ℂ :=
  ∑' n : ℕ, f n / n ^ s
```

where `ArithmeticFunction R` is a wrapper around $\mathbb{N} \to_0 R$.

L-Series

When I started looking at this,
there was a rudimentary L-series package in Mathlib doing roughly this:

```
def LSeries (f : ArithmeticFunction ℂ) (s : ℂ) : ℂ :=
  ∑' n : ℕ, f n / n ^ s
```

where `ArithmeticFunction R` is a wrapper around $\mathbb{N} \to_0 R$.

This makes mathematical sense, as the terms of a Dirichlet series have the form a_n/n^s for $n \geq 1$.

L-Series

When I started looking at this,
there was a rudimentary L-series package in Mathlib doing roughly this:

```
def LSeries (f : ArithmeticFunction ℂ) (s : ℂ) : ℂ :=
  ∑' n : ℕ, f n / n ^ s
```

where `ArithmeticFunction R` is a wrapper around $\mathbb{N} \to_0 R$.

This makes mathematical sense, as the terms of a Dirichlet series have the form a_n/n^s for $n \geq 1$.

But that does not mean it is the best way to implement it in Lean!

L-Series

```
def LSeries (f : ArithmeticFunction ℂ) (s : ℂ) : ℂ := ...
```

L-Series

```
def LSeries (f : ArithmeticFunction ℂ) (s : ℂ) : ℂ := ...
```

We would like to write (using notation from `ArithmeticFunction`)

- `LSeries` ζ ✔
- `LSeries` μ ✔
- `LSeries` log ✖
- `LSeries` Λ ✖
- `LSeries` χ for a Dirichlet character χ ✖

L-Series

```
def LSeries (f : ArithmeticFunction ℂ) (s : ℂ) : ℂ := ...
```

We would like to write (using notation from `ArithmeticFunction`)

- `LSeries` ζ ✔
- `LSeries` μ ✔
- `LSeries` log ✖
- `LSeries` Λ ✖
- `LSeries` χ for a Dirichlet character χ ✖

Problem: There are coercions

- `ArithmeticFunction` $\mathbb{N} \to$ `ArithmeticFunction R` (for `[Semiring R]`)
- `ArithmeticFunction` $\mathbb{Z} \to$ `ArithmeticFunction R` (for `[Ring R]`)

L-Series

```
def LSeries (f : ArithmeticFunction ℂ) (s : ℂ) : ℂ := ...
```

We would like to write (using notation from `ArithmeticFunction`)

- `LSeries` ζ ✔
- `LSeries` μ ✔
- `LSeries` log ✖
- `LSeries` Λ ✖
- `LSeries` χ for a Dirichlet character χ ✖

Problem: There are coercions

- `ArithmeticFunction` $\mathbb{N} \to$ `ArithmeticFunction R` (for `[Semiring R]`)
- `ArithmeticFunction` $\mathbb{Z} \to$ `ArithmeticFunction R` (for `[Ring R]`)

but not, e.g.,

- `ArithmeticFunction` $\mathbb{R} \to$ `ArithmeticFunction` $\mathbb{C}$

L-Series

```
def LSeries (f : ArithmeticFunction ℂ) (s : ℂ) : ℂ := ...
```

We would like to write (using notation from `ArithmeticFunction`)

- `LSeries` ζ ✔
- `LSeries` μ ✔
- `LSeries` log ✖
- `LSeries` Λ ✖
- `LSeries` χ for a Dirichlet character χ ✖

Problem: There are coercions

- `ArithmeticFunction` $\mathbb{N}$ $\to$ `ArithmeticFunction R` (for `[Semiring R]`)
- `ArithmeticFunction` $\mathbb{Z}$ $\to$ `ArithmeticFunction R` (for `[Ring R]`)

but not, e.g.,

- `ArithmeticFunction` $\mathbb{R}$ $\to$ `ArithmeticFunction` $\mathbb{C}$

There does not seem to be a good way
to set this up in the desirable generality.

L-Series: Take Two

L-Series: Take Two

What about

`def LSeries (f : ℕ+ → ℂ) (s : ℂ) : ℂ := ...`   ?

L-Series: Take Two

What about

```
def LSeries (f : ℕ+ → ℂ) (s : ℂ) : ℂ := ...   ?
```

Problems:

L-Series: Take Two

What about

```
def LSeries (f : ℕ+ → ℂ) (s : ℂ) : ℂ := ...    ?
```

Problems:

- Going between $\mathbb{N}+$ and $\mathbb{N}$ is somewhat painful

L-Series: Take Two

What about

`def LSeries (f : ℕ+ → ℂ) (s : ℂ) : ℂ := ...` ?

Problems:

- Going between $\mathbb{N}+$ and $\mathbb{N}$ is somewhat painful
- Some API for $\mathbb{N}+$ is missing

L-Series: Take Two

What about

`def LSeries (f : ℕ+ → ℂ) (s : ℂ) : ℂ := ...` ?

Problems:

- Going between $\mathbb{N}+$ and $\mathbb{N}$ is somewhat painful
- Some API for $\mathbb{N}+$ is missing
- Have to redo `divisorsAntidiagonal` for $\mathbb{N}+$ (for Dirichlet convolution)

L-Series: Take Three

L-Series: Take Three

After some discussions and experiments, we settled on

```
def LSeries (f : ℕ → ℂ) (s : ℂ) : ℂ := ...
```

where the value `f 0` is simply ignored.

L-Series: Take Three

After some discussions and experiments, we settled on

```
def LSeries (f : ℕ → ℂ) (s : ℂ) : ℂ := ...
```

where the value `f 0` is simply ignored.

We can solve the coercion problem as follows.

```
scoped[LSeries.notation] notation:max "↗" f:max => fun n : ℕ ↦ (f n : ℂ)
```

L-Series: Take Three

After some discussions and experiments, we settled on

```
def LSeries (f : ℕ → ℂ) (s : ℂ) : ℂ := ...
```

where the value `f 0` is simply ignored.

We can solve the coercion problem as follows.

```
scoped[LSeries.notation] notation:max "↗" f:max => fun n : ℕ ↦ (f n : ℂ)
```

We can then write (abbreviating `LSeries` to `L`)

`L ↗ζ`, `L ↗μ`, `L ↗Λ`, `L ↗χ`, and it works!

L-Series Terms

L-Series Terms

To get a clean separation of the implementation details and the L-series “logic”, we define

```
def LSeries.term (f : ℕ → ℂ) (s : ℂ) (n : ℕ) : ℂ :=
  if n = 0 then 0 else f n / n ^ s
```

L-Series Terms

To get a clean separation of the implementation details and the L-series "logic", we define

```
def LSeries.term (f : ℕ → ℂ) (s : ℂ) (n : ℕ) : ℂ :=
  if n = 0 then 0 else f n / n ^ s
```

```
@[simp] lemma term_zero (f : ℕ → ℂ) (s : ℂ) : term f s 0 = 0 := rfl
```

```
@[simp] lemma term_of_ne_zero {n : ℕ} (hn : n ≠ 0) (f : ℕ → ℂ) (s : ℂ) :
    term f s n = f n / n ^ s := if_neg hn
```

L-Series Terms

To get a clean separation of the implementation details and the L-series "logic", we define

```
def LSeries.term (f : ℕ → ℂ) (s : ℂ) (n : ℕ) : ℂ :=
  if n = 0 then 0 else f n / n ^ s
```

```
@[simp] lemma term_zero (f : ℕ → ℂ) (s : ℂ) : term f s 0 = 0 := rfl
```

```
@[simp] lemma term_of_ne_zero {n : ℕ} (hn : n ≠ 0) (f : ℕ → ℂ) (s : ℂ) :
    term f s n = f n / n ^ s := if_neg hn
```

and then

```
def LSeries (f : ℕ → ℂ) (s : ℂ) : ℂ := ∑' n : ℕ, term f s n
```

L-Series Terms

To get a clean separation of the implementation details and the L-series "logic", we define

```
def LSeries.term (f : ℕ → ℂ) (s : ℂ) (n : ℕ) : ℂ :=
  if n = 0 then 0 else f n / n ^ s

@[simp] lemma term_zero (f : ℕ → ℂ) (s : ℂ) : term f s 0 = 0 := rfl

@[simp] lemma term_of_ne_zero {n : ℕ} (hn : n ≠ 0) (f : ℕ → ℂ) (s : ℂ) :
    term f s n = f n / n ^ s := if_neg hn
```

and then

```
def LSeries (f : ℕ → ℂ) (s : ℂ) : ℂ := ∑' n : ℕ, term f s n

def LSeriesHasSum (f : ℕ → ℂ) (s a : ℂ) : Prop := HasSum (term f s) a

def LSeriesSummable (f : ℕ → ℂ) (s : ℂ) : Prop := Summable (term f s)
```

Example: Convolution

Example: Convolution

```
lemma term_convolution (f g : ℕ → ℂ) (s : ℂ) (n : ℕ) :
    term (f ⊛ g) s n =
      ∑ p in n.divisorsAntidiagonal, term f s p.1 * term g s p.2 := ...
```

Example: Convolution

```
lemma term_convolution (f g : ℕ → ℂ) (s : ℂ) (n : ℕ) :
    term (f ⊛ g) s n =
      ∑ p in n.divisorsAntidiagonal, term f s p.1 * term g s p.2 := ...

lemma term_convolution' (f g : ℕ → ℂ) (s : ℂ) :
    term (f ⊛ g) s =
      fun n ↦ ∑' (b : (fun p : ℕ × ℕ ↦ p.1 * p.2) ⁻¹' {n}),
                    term f s b.val.1 * term g s b.val.2 := ...
```

Example: Convolution

```
lemma term_convolution (f g : ℕ → ℂ) (s : ℂ) (n : ℕ) :
    term (f ⊛ g) s n =
      ∑ p in n.divisorsAntidiagonal, term f s p.1 * term g s p.2 := ...

lemma term_convolution' (f g : ℕ → ℂ) (s : ℂ) :
    term (f ⊛ g) s =
      fun n ↦ ∑' (b : (fun p : ℕ × ℕ ↦ p.1 * p.2) ⁻¹' {n}),
                   term f s b.val.1 * term g s b.val.2 := ...

lemma LSeriesHasSum.convolution {f g : ℕ → ℂ} {s a b : ℂ}
    (hf : LSeriesHasSum f s a) (hg :  LSeriesHasSum g s b) :
    LSeriesHasSum (f ⊛ g) s (a * b) := by
```

Example: Convolution

```
lemma term_convolution (f g : ℕ → ℂ) (s : ℂ) (n : ℕ) :
    term (f ⍟ g) s n =
      ∑ p in n.divisorsAntidiagonal, term f s p.1 * term g s p.2 := ...

lemma term_convolution' (f g : ℕ → ℂ) (s : ℂ) :
    term (f ⍟ g) s =
      fun n ↦ ∑' (b : (fun p : ℕ × ℕ ↦ p.1 * p.2) ⁻¹' {n}),
                 term f s b.val.1 * term g s b.val.2 := ...

lemma LSeriesHasSum.convolution {f g : ℕ → ℂ} {s a b : ℂ}
    (hf : LSeriesHasSum f s a) (hg : LSeriesHasSum g s b) :
    LSeriesHasSum (f ⍟ g) s (a * b) := by
  simp only [LSeriesHasSum, term_convolution']
  have hsum :=
    summable_mul_of_summable_norm hf.summable.norm hg.summable.norm
  exact (HasSum.mul hf hg hsum).tsum_fiberwise (fun p ↦ p.1 * p.2)
```

Thank You!