Formalization of p-adic L-functions in Lean 3

Ashvni Narayanan

London School of Geometry and Number Theory, Imperial College London

Abstract

The Euler–Riemann zeta function is a largely studied numbertheoretic object, and the birthplace of several conjectures, such as the Riemann Hypothesis. Different approaches are used to study it, including p-adic analysis : deriving information from p-adic zeta functions. A generalized version of p-adic zeta functions (Riemann zeta function) are p-adic L-functions (resp. Dirichlet L-functions). This paper describes formalization of p-adic L-functions in an interactive theorem prover Lean 3. Kubota–Leopoldt p-adic L-functions are meromorphic functions emerging from the special values they take at negative integers in terms of generalized Bernoulli numbers. They also take twisted values of the Dirichlet L-function at negative integers. This work has never been done before in any theorem prover. Our work is done with the support of **mathlib** 3, one of Lean's mathematical libraries. It required formalization of a lot of associated topics, such as Dirichlet characters, Bernoulli polynomials etc. We formalize these first, then the definition of a p-adic L-function in terms of an integral with respect to the Bernoulli measure, proving that they take the required values at negative integers.

2012 ACM Subject Classification Mathematics of computing → Mathematical software; Theory of computation → Formal languages and automata theory

Keywords and phrases formal math, algebraic number theory, Lean, mathlib

Supplementary Material Copies of the source files relevant to this paper are available in a separate repository.
Software: https://github.com/laughinggas/p-adic-L-functions

Funding *Ashvni Narayanan*: EPSRC Grant EP/S021590/1 (UK)

Acknowledgements The author is supported by EPSRC. The author would like to thank her PhD supervisor Prof Kevin Buzzard for several helpful insights. She would also like to thank Dr Filippo A. E. Nuccio for helpful conversations that helped give shape to the project. She is very grateful to the entire Lean community for their timely help and consistent support.

1 Introduction

We are working on formalizing mathematics in an interactive theorem prover called Lean. Formal verification involves the use of logical and computational methods to establish claims that are expressed in precise mathematical terms [1]. Lean is a powerful tool that facilitates formalization of a system of mathematics supported by a basic set of axioms. There is a large mathematical library of theorems verified by Lean called **mathlib**, maintained by a community of computer scientists and mathematicians. One can then formally verify proofs of new theorems dependent on preexisting theorems in **mathlib**. **mathlib** contained 100579 theorems(as of early October 2022). It would be impossible to construct such a vast library without a highly collaborative spirit and a communal decentralized effort, one of Lean's best features.

p-adic L-functions are a well studied numbertheoretic object. They were initially constructed by Kubota and Leopoldt in [4]. Their motivation was to construct a meromorphic function that helps study the Kummer congruence for Bernoulli numbers, and gives information regarding p-adic class numbers. As a result, these functions take twisted values of the Dirichlet L-function at negative integers, and are also related to the generalized Bernoulli numbers and the p-adic zeta function. There are several different ways of constructing p-adic L-functions, we refer to the constructions given in Chapter 12 of [6]. As a result, one needs to build a lot of background (in the maximum possible generality) before embarking on the main goal.

It is difficult to explain all the mathematical terms used here, we attempt to describe as many as possible. To that effect, a basic knowledge of algebra is assumed. Since **mathlib** works in utmost generality, one often finds that the terminology used is less common. Thanks to the community's endeavour to maintain adequate documentation, we have added links which serve as explanations wherever possible. When clear, we will explicitly skip writing hypotheses in the code, since these can get quite long.

We give a mathematical overview in this section, then discuss background in Section 2, define Dirichlet characters in

Section 3, introduce generalized Bernoulli numbers in Section 4, construct the p-adic L-function in Section 5, and evaluate it at negative integers in Section 6, finishing with a summary in Section 7.

1.1 Mathematical overview

We give a brief overview of the mathematics formalized in this project. L-functions are a fundamental object, appearing almost everywhere in modern number theory. The Dirichlet L-function associated to a Dirichlet character χ is given by

$$L(s,\chi) = \sum_{n=1}^{\infty} \frac{\chi(n)}{n^s} = \prod_{p \text{ prime}} \frac{1}{1-\chi(p)p^{-s}}$$

where s is a complex variable with $Re(s) > 1$. This can be analytically extended to the entire complex plane, with a simple pole at $s = 1$ when $\chi = 1$. Note also that $L(s, 1)$ is the same as the Riemann zeta function. Moreover, it is known that $L(1-n,\chi) = -\frac{B_{n,\chi}}{n}$, where $B_{n,\chi}$ are the generalized Bernoulli numbers.

In this paper, we construct, for an integer prime p, a p-adic analogue of $L(s,\chi)$, called the Kubota–Leopoldt p-adic L-function, denoted $L_p(s,\chi)$. This is generally done by continuously extending the function $L_p(1-n,\chi) := (1-\chi(p)p^{n-1})L(1-n,\chi)$ to the complete p-adic space $\mathbb{C}_p$. In fact, $L_p(s, 1)$ is analytic except for a pole at $s = 1$ with residue $1 - \frac{1}{p}$ (Theorem 5.11, [6]).

Formalization of the p-adic L-functions via analytic continuation was hard, since $\mathbb{C}_p$ did not exist in `mathlib` at the time. Following [6], we instead define it in terms of an "integral" with respect to the Bernoulli measure. We explain these terms below.

A profinite space is a compact, Hausdorff and totally disconnected space. The p-adic integers $\mathbb{Z}_p$, which are the completion of the integers $\mathbb{Z}$ with respect to the valuation $\nu_p(p^\alpha \prod_{p_i \neq p} p_i^{\alpha_i}) = \alpha$ are a profinite space. One may also think of them as the inverse limit of the discrete topological spaces $\mathbb{Z}/p^n\mathbb{Z}$, that is, $\mathbb{Z}_p = \operatorname{proj\,lim}_n \mathbb{Z}/p^n\mathbb{Z}$.

Locally constant functions are those for which the preimage of any set is open. Given a profinite space X and a normed ring R, one can show that the locally constant functions from X to R (denoted $LC(X,R)$) are dense in the space of continuous functions from X to R (denoted $C(X,R)$).

Given an abelian group A, a distribution is defined to be an A-linear map from $LC(X,A)$ to A. A measure ϕ is defined to be a bounded distribution, that is, $\forall f \in LC(X,R)$, $\exists K > 0$ such that $||\phi(f)|| \leq K||f||$, where $||f|| = \sup_{x\in X} ||f(x)||$. An example of a measure is the Bernoulli measure. Given a natural number d coprime to p and a clopen set $U_{n,a}$ of $\mathbb{Z}/d\mathbb{Z} \times \mathbb{Z}_p$, the characteristic function $\chi_{n,a}$ (defined to be 1 on $U_{n,a}$ and 0 otherwise) is a locally constant function. Given a natural number c that is coprime to d and p, we then define the Bernoulli measure E_c by :

$$E_c(\chi_{n,a}) := \left\{\frac{a}{dp^{n+1}}\right\} - c\left\{\frac{c^{-1}a}{dp^{n+1}}\right\} + \frac{c-1}{2}$$

Given a measure μ, the integral with respect to μ is $\int f d\mu := \mu(f)$ for any locally constant function f, and extending this definition to $C(X,R)$. In fact, this is an R-linear map.

Finally, the p-adic L-function is defined to be an integral with respect to the Bernoulli measure. The characterizing property of the p-adic L-function is its evaluation at negative integers :

$$L_p(1-n,\chi) = -(1-\chi\omega^{-n}(p)p^{n-1})\frac{B_{n,\chi\omega^{-n}}}{n}$$

for $n \geq 1$. When defined as an integral, additional work is needed to prove this.

Our contributions to this theory include a formalized definition of the p-adic L-function in generality, taking values in a normed complete non-Archimedean $\mathbb{Q}_p$-algebra, instead of just $\mathbb{C}_p$. Further, it takes as input continuous monoid homomorphisms, also known as elements of the weight space. We have also developed an extensive theory for Dirichlet characters, Bernoulli numbers and polynomials, generalized Bernoulli numbers, properties of p-adic integers and modular arithmetic, making substantial contributions to the `number_theory` section of `mathlib`. We use non-traditional methods to define and prove classical results, often choosing to work with those which are easier to formalize, later proving their equivalence to the original.

1.2 Lean and `mathlib`

Lean 3 is a functional programming language and interactive theorem prover based on dependent type theory. This project is based on Lean's mathematical library `mathlib 3`, which is characterized by its decentralized nature with over 300 contributors. Thus, it is impossible to cite every author who contributed a piece of code that we used.

We assume the reader is familiar with structures such as `def`, `abbreviation`, `lemma`, `theorem`, which are used constantly. An important property of Lean is its typeclass inference system - Lean "remembers" properties given to a `structure` or `class` embedded in an `instance` structure. This is explained in detail in [5]. We shall also use several tactics in proofs, such as `rw`, `apply`, `conv` and `refine` [1].

2 Preliminaries

2.1 Filters and convergence

None of our mathematical proofs require filters on paper, however, we find that working with them makes formalizing our proofs significantly less cumbersome. Due to the efforts of Johannes Hölzl, Jeremy Avigad, Patrick Massot and several others, we have a vast API for filters in Lean. We shall not delve into the details of what a filter is, but instead explain how they are used to formalize convergence and limits.

For a sequence of functions $(f_n)_{n\in\mathbb{N}}$, the expression $\lim_{n\to\infty} f_n(x) = a$ is represented as :

```
tendsto (λ n : ℕ, f_n) filter.at_top ( 𝓝 a)
```

Here, `filter.at_top` (for the naturals) is a filter on $\mathbb{N}$ generated by the collection of sets $\{b \mid a \le b\}$ for all $a \in \mathbb{N}$. The following lemma is particulary useful :

```
/-- If f₁ and f₂ are equal almost everywhere, then f₁ converges if and only if f₂ converges. -/
lemma filter.tendsto_congr' {α : Type} {β : Type u_1} {f₁ f₂ : α → β}
  {l₁ : filter α} {l₂ : filter β} (h : f₁ =ᶠ[l₁] f₂) :
  tendsto f₁ l₁ l₂ ↔ tendsto f₂ l₁ l₂
```

This lemma shows that sequences that are the same after finitely many elements have the same limit. Given two sequences `f₁` and `f₂` (thought of as functions from $\mathbb{N}$),
`f₁ =ᶠ[at_top] f₂ ⟷ ∃ (a : ℕ), ∀ (b : ℕ), b ≥ a, f₁ b = f₂ b.`

An equivalent condition to convergence on metric spaces is :

```
lemma metric.tendsto_at_top : ∀ {α : Type u_1} {β : Type}
  [pseudo_metric_space α] [nonempty β] [semilattice_sup β]
  {u : β → α} {a : α} :
  tendsto u at_top ( 𝓝 a) ↔ ∀ (ε : ℝ) (h : ε > 0),
  (∃ (N : β), ∀ (n : β), n ≥ N → ‖ u n - a ‖ < ε)
```

Thus, in order to prove lemmas about convergence, one can either choose to continue doing computations in the `tendsto` framework, or prove normed inequalities. Working with the former really simplified calculations. As an example, suppose we want to prove the convergence of the sequence g given by $g(0) = g(2) = 1$ and $g(n) = 3f(n)$, where f is a convergent sequence. This is a one-line proof using `filter.tendsto_congr'`. Using the above lemma, one must obtain N corresponding to $\varepsilon/3$, and also prove that $0 < \varepsilon/3$. With more complex expressions, this gets computationally difficult to handle.

Hence, we try to avoid using `metric.tendsto_at_top` when possible. The only cases where it is used is when direct inequalities need to be dealt with; this happens precisely when the non-Archimedean condition on R is used. Thus, this is a good indicator of where the non-Archimedean condition is needed.

[1] https://leanprover-community.github.io/mathlib_docs/tactics.html has a full list of tactics in Lean

2.2 Modular arithmetic and units

Some fundamental objects with which we shall work throughout are the finite spaces $\mathbb{Z}/n\mathbb{Z}$. Note that proving properties for `zmod n` is equivalent to proving them for any finite cyclic group. Given a positive $n \in \mathbb{N}$, $\mathbb{Z}/n\mathbb{Z}$ is the same as `fin n`(and $\mathbb{Z}$ for $n = 0$), the set of natural numbers upto n. It is also the set of equivalence classes obtained via the relation on $\mathbb{Z} : a \sim b \iff n \mid a - b$. It has a natural group structure, and is given the discrete topology, making it a topological group. Some maps used constantly include
`val:zmod n → ℕ`, which takes any element to its smallest nonnegative reprentative less than `n`; and `cast_hom:zmod n → R`, a coercion to a ring, obtained by composing the canonical coercion with `val`. If `R` has characteristic dividing `n`, the map is a ring homomorphism. Given coprime naturals m, n, an important equivalence is
`chinese_remainder:zmod (m * n) ≃+* zmod m × zmod n`. About 45 additional lemmas were required, which have been put in a separate file, `zmod/properties.lean`.

Every `monoid M` has an associated space of invertible elements or units, denoted
`units M` or `Mˣ`. We use the map `units.coe_hom:Mˣ → M` to identify a unit in its parent space frequently. Given a `monoid_hom` (abbreviated as `→*`) `R →* S` for monoids `R` and `S`, one can obtain a homomorphism `Rˣ →* Sˣ` by `units.map`.

3 Dirichlet characters and the Teichmüller character

An important task was to formalize Dirichlet characters, an integral part of the definition of the p-adic L-function. Dirichlet characters are often not found to be defined in this technical manner. Another addition is the definition of Dirichlet characters of level and conductor 0. The words character and Dirichlet character are used interchangeably.

Dirichlet characters are usually defined as group homomorphisms from $\mathbb{Z}/n\mathbb{Z}^\times$ to $\mathbb{C}^\times$ for some natural number n. A lot of properties traditionally known for groups hold more generally and are defined in greater generality in `mathlib` for monoids. In the same spirit, we define Dirichlet characters to be monoid homomorphisms on any `monoid` :

```
abbreviation dirichlet_character (R : Type*) [monoid R] (n : ℕ) :=
  (zmod n)ˣ →* R ˣ
/-- The level of a Dirichlet character. -/
abbreviation lev {R : Type*} [monoid R] {n : ℕ}
  (χ : dirichlet_character R n) : ℕ := n
```

If we gave the definition of Dirichlet characters a `def` structure, `dirichlet_character` would become a `Type` distinct from `(zmod n)ˣ →* Rˣ`, making compositions with `monoid_hom` complicated; hence we used `abbreviation` instead. Note that the linter returns an extra unused argument warning (for χ) for the latter definition.

Given a Dirichlet character χ, `asso_dirichlet_character` χ returns a monoid homomorphism from $\mathbb{Z}/n\mathbb{Z}$ to R, which is χ on the units and 0 otherwise.

```
noncomputable abbreviation asso_dirichlet_character {R : Type*}
  [monoid_with_zero R] {n : ℕ} (χ : dirichlet_character R n) :
  zmod n →* R := { to_fun :=
  function.extend (units.coe_hom (zmod n)) ((units.coe_hom R) ∘ χ) 0, ..}
```

Lean requires us to tag this definition `noncomputable`, since we are producing data from an existential statement, `classical.some` (appearing in `function.extend`), which has no computational content (see Chapter 11 of [1]). One would like to shift between compatible Dirichlet characters of different levels. For this, we construct the following tools :

```
/-- Extends the Dirichlet character χ of level n to level m, for n ∣ m. -/
def change_level {m : ℕ} (hm : n ∣ m) :
  dirichlet_character R n →* dirichlet_character R m :=
{ to_fun := λ ψ, ψ.comp (units.map (zmod.cast_hom hm (zmod n))), .. }
```

```
/-- χ₀ of level d factors through χ of level n if d | n and
    χ₀ = χ ∘ (zmod n → zmod d). -/
structure factors_through (d : ℕ) : Prop :=
(dvd : d ∣ n)
(ind_char : ∃ χ₀ : dirichlet_character R d, χ = χ₀.change_level dvd)
```

The notions of primitivity and conductor of a Dirichlet character follow easily :

```
/-- The set of numbers for which a Dirichlet character is periodic. -/
def conductor_set : set ℕ := {x : ℕ | χ.factors_through x}
/-- The minimum natural number n for which a character is periodic. -/
noncomputable def conductor : ℕ := Inf (conductor_set χ)
/-- A character is primitive if its level is equal to its conductor. -/
def is_primitive : Prop := χ.conductor = n
/-- The primitive character associated to a Dirichlet character. -/
noncomputable def asso_primitive_character : dirichlet_character R χ.conductor := classical.some (
    χ.factors_through_conductor).ind_char
```

Here, `classical.some` makes an arbitrary choice of an element from a nonempty space, and `classical.some_spec` lists down properties of this element coming from the space.

When $a = b$, while `dirichlet_character R a` and `dirichlet_character R b` are “mathematically” equal, Lean does not think of them as the same type. This gets complicated when additional layers, such as `change_level` are added to the equation. A general method to resolve such problems is by using the tactic `subst`, which would substitute a with b; however, that failed. Instead, we used the concept of heterogeneous equality (`heq`, or `==`) to deal with this. The tactic `congr'` helped reduce to expressions of heterogeneous equality, which were then solved with the help of lemmas such as :

```
lemma change_level_heq {a b : ℕ} {S : Type*} [comm_monoid_with_zero S]
  (χ : dirichlet_character S a) (h : a = b) :
  change_level (show a ∣ b, from by {rw h}) χ == χ
```

This states that, for $a = b$, changing the level of a Dirichlet character of level a to b is heterogeneously equal to itself. Traditionally only for primitive characters, our definition of multiplication of characters extends to any two characters. This takes as input characters χ_1 and χ_2 of levels `n` and `m` respectively, and returns the primitive character associated to $\chi_1'\chi_2'$, where χ_1' and χ_2' are obtained by changing the levels of χ_1 and χ_2 to `lcm n m`.

```
noncomputable def mul {m n : ℕ} (χ₁ : dirichlet_character R n)
  (χ₂ : dirichlet_character R m) :=
asso_primitive_character(change_level χ₁ (dvd_lcm_left n m) *
  change_level χ₂ (dvd_lcm_right n m))
```

This multiplication is not trivially commutative or associative, with respect to this definition.

We need the notion of odd and even characters. A character χ is odd if $\chi(-1) = -1$, and even if $\chi(-1) = 1$. For a commutative ring, any character is either odd or even :

```
lemma is_odd_or_is_even {S : Type*} [comm_ring S] [no_zero_divisors S]
  {m : ℕ} (ψ : dirichlet_character S m) : ψ.is_odd ∨ ψ.is_even
```

3.1 Teichmüller character

The initial effort was to formalize the definition of the Teichmüller character (denoted ω) directly. However, it was discovered that Witt vectors, and in particular Teichmüller lifts had previously been added to `mathlib` by Johan Commelin and Robert Lewis. This reiterates the importance of the collaborative spirit of Lean, and of making definitions in the correct generality.

It is beyond the scope of this text to define Witt vectors and do it justice. We refer interested readers to Section 2.4 of [3]. For a commutative ring R and a prime number p, one can obtain a ring of Witt vectors $\mathbb{W}(R)$. When we take $R = \mathbb{Z}/p\mathbb{Z}$, we get

```
def equiv : 𝕎 (zmod p) ≃+* ℤ_[p]
```

One also obtains the Teichmüller lift $R \to \mathbb{W}(R)$. Given $r \in R$, the 0-th coefficient is r, and the other coefficients are 0. This map is a multiplicative monoid homomorphism and is denoted `teichmuller`.

Combining this with the previous two definitions, we obtain our definition of the Teichmüller character :

```
noncomputable abbreviation teichmuller_character_mod_p (p : ℕ)
  [fact (nat.prime p)] : dirichlet_character ℤ_[p] p := units.map
  (((witt_vector.equiv p).to_monoid_hom).comp (witt_vector.teichmuller p))
```

We use `[fact p.prime]` to make the primality of p an instance. This map takes $x \in \mathbb{Z}/p\mathbb{Z}^\times$ to a root of unity $y \in \mathbb{Z}_p$ such that $y \equiv x(\texttt{mod p})$. Often we view this as taking values in a $\mathbb{Q}_p$-algebra R, by composing it with `algebra_map ℚ_[p] R`, which identifies elements of $\mathbb{Q}_p$ in R. Since we mostly deal with ω^{-1} taking values on $R^\times$, we define this as `teichmuller_character_mod_p'`.

We proved properties of Teichmüller characters in `teichmuller_character.lean`, such as, for odd primes p, the Teichmüller character is odd, and `1` otherwise :

```
lemma eval_neg_one (hp : 2 < p) : teichmuller_character_mod_p p (-1) = -1
```

4 Bernoulli polynomials and the generalized Bernoulli number

The Bernoulli numbers B'_n are generating functions given by $\sum B'_n \frac{t^n}{n!} = \frac{t}{e^t-1}$. They appear in the computation of sums of powers of naturals, $\sum_n n^k$. Note that several authors think of Bernoulli numbers B_n to be defined as $\sum B_n \frac{t^n}{n!} = \frac{t}{1-e^{-t}}$. The difference between these two is : $B_n = (-1)^n B'_n$, with $B'_1 = -\frac{1}{2}$. A reformulation gives :

$$B'_n = 1 - \sum_{k=0}^{n-1} \binom{n}{k} \frac{B'_k}{n-k+1}$$

In `mathlib`, B'_n was already defined (by Johan Commelin) as above. However, we needed B_n, which we then defined as :

```
def bernoulli (n : ℕ) : ℚ := (-1)^n * bernoulli' n
```

The Bernoulli polynomials, denoted $B_n(X)$, a generalization of the Bernoulli numbers, are generating functions $\sum_{n=0}^{\infty} B_n(X) \frac{t^n}{n!} = \frac{te^{tX}}{e^t-1}$. This gives :

$$B_n(X) = \sum_{i=0}^{n} \binom{n}{i} B_i X^{n-i}$$

We defined the Bernoulli polynomials as :

```
def polynomial.bernoulli (n : ℕ) : polynomial ℚ :=
  Σ i in range (n + 1), monomial (n - i) ((bernoulli i) * (choose n i))
```

Here, `monomial n a` translates to aX^n, and `∑ i in s, f i` translates to $\sum_{i \in s} f(i)$, for a `finset` (or finite set) `s`. A small aspect of this naming convention is that if the namespaces for Bernoulli numbers and polynomials are both open (which is often the case), in order to use the Bernoulli numbers, one needs to use `_root_.bernoulli`. We shall use them interchangeably here, when the context is clear.

An important fact is, $\forall n$, $(n+1)X^n = \sum_{k=0}^{n} \binom{n}{k} B_k(X)$:

```
theorem sum_bernoulli (n : ℕ) : monomial n (n + 1 : ℚ) =
  Σ k in range (n + 1), ((n + 1).choose k : ℚ) · bernoulli k
```

These proofs are relatively straightforward. Most of this work is now part of `mathlib`, and has been used to give a formalized proof of Faulhaber's theorem.

4.1 Generalized Bernoulli numbers

Generalized Bernoulli numbers are integral to our work, since these are related to the special values of p-adic L-functions and Dirichlet L-functions. Given a primitive Dirichlet character χ of conductor f, the generalized Bernoulli numbers are defined as (section 4.1, [6]) $\sum_{n=0}^{\infty} B_{n,\chi} \frac{t^n}{n!} = \sum_{a=1}^{f} \frac{\chi(a)te^{at}}{e^{ft}-1}$. For any multiple F of f, Proposition 4.1 of [6] gives us :

$$B_{n,\chi} = F^{n-1} \sum_{a=1}^{F} \chi(a) B_n\left(\frac{a}{F}\right)$$

This is much easier to work with, so we use this as our definition instead, taking $F = f$:

```
def general_bernoulli_number {S : Type*} [comm_semiring S] [algebra ℚ S]
  {n : ℕ} (ψ : dirichlet_character S n) (m : ℕ) : S :=
  (algebra_map ℚ S ((ψ.conductor)^(m - 1 : ℤ))) *
  Σ a in finset.range ψ.conductor,
  asso_dirichlet_character (asso_primitive_character ψ) a.succ *
  algebra_map ℚ S ((bernoulli m).eval (a.succ / ψ.conductor : ℚ))
```

Contrary to the traditional definition, this is for all characters, and ψ takes values in any commutative $\mathbb{Q}$-algebra, instead of $\mathbb{C}$. One had to also explicitly mention that `m - 1` must be taken to have type $\mathbb{Z}$, since Lean would otherwise infer it to have type $\mathbb{N}$, which might have caused errors (subtraction on $\mathbb{N}$ and $\mathbb{Z}$ are different).

4.2 A special property of generalized Bernoulli numbers

An important property of these numbers is :

▶ **Theorem 1.** *Let χ be an even Dirichlet character of level dp^m for d coprime to the odd prime p, with m positive. Suppose R is a nontrivial commutative non-Archimedean normed $\mathbb{Q}_p$-algebra with no zero divisors. For $k > 1$,*

$$\lim_{n\to\infty} \frac{1}{dp^n} \sum_{0<a<dp^n;(a,dp)=1} \chi\omega^{-k}(a)a^k = (1 - \chi\omega^{-k}(p)p^{k-1})B_{k,\chi\omega^{-k}}$$

Instead of giving `R` a non-Archimedean structure(which did not exist in `mathlib` when this project began), we give as input its consequences, conditions `na` and `na'`. This is formulated in Lean as :

```
theorem lim_even_character' (na' : ∀ (n : ℕ) (f : (zmod n)ˣ → R),
  ‖Σ i : (zmod n)ˣ, f i ‖ ≤ ⨆ (i : (zmod n)ˣ), ‖f i ‖)
(na : ∀ (n : ℕ) (f : ℕ → R),
  ‖ Σ (i : ℕ) in finset.range n, f i ‖ ≤ ⨆ (i : zmod n), ‖f i.val ‖) :
  tendsto (λ (n : ℕ), (1 / ↑(d * p ^ n)) ·
  Σ (i : ℕ) in finset.range (d * p ^ n),
  asso_dirichlet_character (χ.mul (teichmuller_character_mod_p' p R ^ k)) ↑i * ↑i ^ k) at_top (
    𝓝 (general_bernoulli_number
  (χ.mul (teichmuller_character_mod_p' p R ^ k)) k))
```

The proof of this theorem follows from the proof in Lemma 7.11 of [6], a point of difference being that our theorem holds more generally for R being a non-Archimedean normed commutative $\mathbb{Q}_p$-algebra with no zero divisors, instead of $\mathbb{C}_p$. Majorly, it equates the two sides modulo p^n for a sufficiently large n, and uses the fact that

▶ **Theorem 2.**

$$\lim_{n \to \infty} \frac{1}{dp^n} \sum_{0<a<dp^n;(a,dp)=1} \chi\omega^{-m}(a)a^m = 0$$

The formalization is very calculation intensive, and is a good example of a small proof on paper being magnified in Lean, because there are multiple coercions and arithmetic calculations to be dealt with. Unfortunately, tactics such as `ring` and `simp` that usually help with these fail here. It is translated in Lean as :

```
lemma sum_even_character_tendsto_zero_of_units :
  tendsto (λ n, Σ (i : (zmod (d * p^n))ˣ), ((asso_dirichlet_character
  (χ.mul (teichmuller_character_mod_p' p R^k))) i * i^(k - 1)))
  at_top ( 𝓝 0)
```

The proof of this theorem is in `tendsto_zero_of_sum_even_char.lean`.

5 Construction of the p-adic L-function

5.1 Density of locally constant functions

For any compact Hausdorff totally disconnected space X and a commutative normed ring A, we have proved that $LC(X, A)$ is a dense subset of $C(X, A)$. Formalizing this took about 500 lines of code (now in `mathlib`), and is based on the fact that locally compact Hausdorff totally disconnected spaces have a clopen basis :

```
lemma loc_compact_Haus_tot_disc_of_zero_dim {H : Type*} [t2_space H]
  [locally_compact_space H] [totally_disconnected_space H] :
  is_topological_basis {s : set H | is_clopen s}
```

This turned out to be hard to formalize. Given a set s of H, Lean gives a subset V of s the type `V:set s`; however, Lean does not recognize V as a subset of H. As a result, to use `compact_space s` $\longleftrightarrow$ `is_compact (s:set H)`, one must construct `V':set H` to be the image of V under the closed embedding `coe:s → H`. This process must be repeated each time a subset of H, which is also a topological subspace, is considered. Finally, it must be shown that all these coercions match up in the big topological space H.

5.2 Clopen sets of the p-adic integers

$\mathbb{Z}_p$ is a profinite space (as shown in section 2.4 of [3]). It is the inverse limit of finite discrete topological spaces $\mathbb{Z}/p^n\mathbb{Z}$ for all n, and has a clopen basis of the form $U_{a,n} := proj_n^{-1}(a)$ for $a \in \mathbb{Z}/p^n\mathbb{Z}$, where $proj_n$ is the canonical projection ring homomorphism `to_zmod_pow n:ℤ_[p] →+* zmod (p ^ n)`. We first define the collection of sets $(U_{a,n})_{a,n}$:

```
def clopen_basis : set (set ℤ_[p]) :=
  {x : set ℤ_[p] | ∃ (n : ℕ) (a : zmod (p^n)),
    x = set.preimage (padic_int.to_zmod_pow n) {a} }
```

We show that `clopen_basis` forms a topological basis and that every element is clopen :

```
theorem clopen_basis_clopen : (clopen_basis p).is_topological_basis ∧
  ∀ x ∈ (clopen_basis p), is_clopen x
```

The mathematical proof is to show that for any ϵ-ball, one can find $U_{a,n}$ inside it. This is true because, given $n \in \mathbb{N}$ and $x \in \mathbb{Z}/p^n\mathbb{Z}$, the preimage of x under `to_zmod_pow n` is the same as the ball centered at x (now considered as an element of $\mathbb{Z}_p$) with radius p^{1-n}. The following lemmas prove useful :

```
lemma appr_spec (n : ℕ) (x : ℤ_[p]) :
  x - appr x n ∈ (ideal.span {p^n} : ideal ℤ_[p])
lemma has_coe_t_eq_coe (x : ℤ_[p]) (n : ℕ) :
  (((appr x n) : zmod (p^n)) : ℤ_[p]) = ((appr x n) : ℤ_[p])
```

For `x:ℤ_[p]`, `appr x n` is the smallest natural number in `x (mod p^n)`. In the latter lemma, the RHS is a coercion of `appr x n`, which has type ℕ, to `ℤ_p`. The LHS is a coercion of `appr x n` to `zmod (p ^ n)` to `ℤ_p`. This statement is not true in general, that is, given any natural number n, it is not true that the lift of n to $\mathbb{Z}_p$ is the same as the composition of its lift to $\mathbb{Z}/p^n\mathbb{Z}$ and $\mathbb{Z}_p$. It works here because the coercion from $\mathbb{Z}/p^n\mathbb{Z}$ to $\mathbb{Z}_p$ is not the canonical lift. It is a composition of a coercion from $\mathbb{Z}/p^n\mathbb{Z}$ to $\mathbb{N}$, which takes $a \in \mathbb{Z}/p^n\mathbb{Z}$ to the smallest natural number in its $\mathbb{Z}/p^n\mathbb{Z}$ equivalence class.

One can similarly show that the sets $U_{b,a,n} := proj_1^{-1}(b) \times proj_{2,n}^{-1}(a)$ form a clopen basis for $\mathbb{Z}/d\mathbb{Z} \times \mathbb{Z}_p$, where $proj_1$ is the first canonical projection on $b \in \mathbb{Z}/d\mathbb{Z}$ and $proj_{2,n}$ the composition of the second projection on $a \in \mathbb{Z}_p$ with $proj_n$ described above. We call this set `clopen_basis' p d`. Its properties are formalized in `padic_int.clopen_properties.lean`.

5.3 Distributions and measures

In this section, $X = \varprojlim_{i \in \mathbb{N}} X_i$ denotes a profinite space with X_i finite and projection maps $\pi_i : X \to X_i$ and surjective maps $\pi_{ij} : X_i \to X_j$ for all $i \geq j$. Henceforth, we use G to denote an abelian group, A for a commutative normed ring, R for a commutative complete normed ring which is also a $\mathbb{Q}_p$-algebra, and $LC(X, Y)$ for the space of locally constant functions from X to Y. We fix a prime p and an integer d such that $gcd(d, p) = 1$.
The topology on $C(X, A)$ comes from its normed group structure induced by the norm on A : $||f-g|| = sup_{x \in X}||f(x)-g(x)||$. In fact, this topology is the same as the topology defined on bounded functions on X, since X is a compact space. Since the API for bounded continuous functions on compact spaces was developed at around the same time (created by Oliver Nash), we used the existing lemmas such as `equiv_bounded_of_compact`.

A distribution (from Section 12.1 of [6]) is a G-linear function $\phi : LC(X, G) \to G$. This is already a `Type`, hence we do not redefine it. Measures (not to be confused with measure theory measures) are bounded distributions :

```
def measures := {φ : (locally_constant X A) →ₗ[A] A // ∃ K : ℝ, 0 < K ∧
    ∀ f : (locally_constant X A), ‖φ f‖ ≤ K * ‖inclusion X A f‖ }
```

The map `inclusion` identifies the locally constant function `f` as a continuous function. The boundedness of the distribution makes the measure continuous.

5.4 The Bernoulli measure

The Bernoulli measure is an essential measure. We make a choice of an integer c with $gcd(c, dp) = 1$, and c^{-1} is an integer such that $cc^{-1} \equiv 1 \bmod dp^{2n+1}$. For a clopen set $U_{a,n}$, we define

$$E_c(\chi_{U_{a,n}}) = E_{c,n}(a) = \left\{ \frac{a}{dp^{n+1}} \right\} - c \left\{ \frac{c^{-1}a}{dp^{n+1}} \right\} + \frac{c-1}{2}$$

In Lean, this translates to (note that `fract x` represents the fractional part of x) :

```
def bernoulli_distribution := λ (n : ℕ) (a : (zmod (d * (p^n)))),
  fract ((a : ℤ) / (d*p^(n + 1)))
  - c * fract ((a : ℤ) / (c * (d*p^(n + 1)))) + (c - 1)/2
```

The original plan was to define a set of the form :

```
def bernoulli_measure (hc : c.gcd p = 1) :=
 {x : locally_constant (zmod d × ℤ_[p]) R →ₗ[R] R | ∀ (n : ℕ)
  (a : zmod (d * (p^n))), x (char_fn R (clopen_from.is_clopen p d n a)) =
  (algebra_map ℚ R) (E_c p d hc n a) }
```

and to show that it is nonempty. `char_fn` is a locally constant characteristic function on a clopen set (1 on the set and 0 otherwise), taking as input the range of the function and the fact that the set is clopen. However, information is lost this way, since one then has to use `classical.some` to extract the underlying measure. We use an elegant approach :

```
/-- A sequence has the ‘is_eventually_constant‘ predicate if all the elements of the sequence are
    eventually the same. -/
def is_eventually_constant {α : Type*} (a : ℕ → α) : Prop :=
  { n | ∀ m, n ≤ m → a (nat.succ m) = a m }.nonempty
structure eventually_constant_seq {α : Type*} :=
(to_seq : ℕ → α)
(is_eventually_const : is_eventually_constant to_seq)
```

Given a locally constant function f from $\mathbb{Z}/d\mathbb{Z} \times \mathbb{Z}_p$ to R, we define the eventually constant sequence `from_loc_const` :

```
noncomputable abbreviation from_loc_const : @eventually_constant_seq R :=
{ to_seq := λ (n : ℕ),
    Σ a in (zmod' (d * p^n) _),
    f(a) · ((algebra_map ℚ_[p] R) (bernoulli_distribution p d c n a)),
  is_eventually_constant := _, }
```

for all natural numbers n. `zmod'` is the universal `finset` of `zmod`. We shall look into the proof of this sequence being eventually constant later.

Given a locally constant function `f:locally_constant ((zmod d)ˣ × ℤ_[p]ˣ) R`, an element of the set `bernoulli_measur` is given by :

```
  sequence_limit (from_loc_const p d R (loc_const_ind_fn _ p d f))
```

where `loc_const_ind_fn` is a locally constant function on $\mathbb{Z}/d\mathbb{Z} \times \mathbb{Z}_p$ that takes value f on the units of the domain, and 0 otherwise. The linearity properties follow easily. Notice that `bernoulli_distribution` takes locally constant functions on $\mathbb{Z}/d\mathbb{Z} \times \mathbb{Z}_p$, while `bernoulli_measure` takes locally constant functions on $\mathbb{Z}/d\mathbb{Z}^* \times \mathbb{Z}_p^*$. This had to be done since our clopen basis was defined on $\mathbb{Z}/d\mathbb{Z} \times \mathbb{Z}_p$, and while it is easy to show the same results for the units on paper, it requires a bit of work in Lean.

We now prove that `bernoulli_measure` is indeed a measure, that is, it is bounded. The bound we choose is $K := 1+ \parallel c \parallel + \parallel \frac{c-1}{2} \parallel$. The proof is as follows : let ϕ denote `loc_const_ind_fn`. We want $\parallel E_c(\phi(f)) \parallel \leq K \parallel f \parallel$. It suffices to prove this for $\chi_{n,a}$, because one can find an n such that $\phi(f) = \sum_{a \in \mathbb{Z}/d\mathbb{Z} \times \mathbb{Z}/p^n\mathbb{Z}} \phi(f)(a) \dot{\chi}_{n,a}$:

```
lemma loc_const_eq_sum_char_fn (f : locally_constant ((zmod d) × ℤ_[p]) R) (hd : d.gcd p = 1) : ∃
    n : ℕ, f = Σ a in (finset.range (d * p^n)),
      f(a) · char_fn R (clopen_from.is_clopen p d n a)
```

This proof is akin to proving that `from_loc_const` is eventually constant, using discrete quotients. The discrete quotient on a topological space is given by an equivalence relation such that all equivalence classes are clopen :

```
structure (X : Type*) [topological_space X] discrete_quotient :=
(rel : X → X → Prop)
(equiv : equivalence rel)
(clopen : ∀ x, is_clopen (set_of (rel x)))
```

The last statement translates to, $\forall x \in X, \{y | y \sim x\}$ is clopen. Given two discrete quotients A and B, $A \leq B$ means $\forall x, y \in X$, $x \sim_A y \implies x \sim_B y$. Any locally constant function induces a discrete quotient via its clopen fibers :

```
def locally_constant.discrete_quotient : discrete_quotient X :=
{ rel := λ a b, f b = f a, .. }
```

We now define a function :

```
/-- A discrete quotient induced by ‘to_zmod_pow‘. -/
def discrete_quotient_of_to_zmod_pow :
  ℕ → discrete_quotient (zmod d × ℤ_[p]) :=
  λ n, ⟨λ a b, to_zmod_pow n a.2 = to_zmod_pow n b.2 ∧ a.1 = b.1, _, _⟩
```

For $a = (a_1, a_2)$ and $b = (b_1, b_2)$ in $\mathbb{Z}/d\mathbb{Z} \times \mathbb{Z}_p$, this represents the relation $a \sim b \iff a_2(\text{mod } p^n) = b_2(\text{mod } p^n) \wedge a_1 = b_1$. Then, given a locally constant function f on $\mathbb{Z}/d\mathbb{Z} \times \mathbb{Z}_p$, for N large enough, the fibers of f mod p^N are contained in the basic clopen sets of p^N :

```
lemma le : ∃ N : ℕ,
  discrete_quotient_of_to_zmod_pow p d N ≤ discrete_quotient f
```

The proofs now follow from this fact : $\exists N, \forall m \geq N$,

$$\sum_{a \in \mathbb{Z}/dp^{m+1}\mathbb{Z}} f(a) E_{c,m+1}(a) = \sum_{a \in \mathbb{Z}/dp^m\mathbb{Z}} f(a) E_{c,m}(a)$$

The required N is `classical.some (discrete_quotient_of_to_zmod_pow.le f) + 1`. We also define the following :

```
/-- Set of all ‘b ∈ zmod (d * p^m)‘ such that ‘b = a mod (d * p^n)‘ for
  ‘a ∈ zmod (d * p^n)‘. -/
def equi_class (n m : ℕ) (a : zmod (d * p^n)) :=
  {b : zmod (d * p^m) | (b : zmod (d * p^n)) = a}
```

Then, we have the following lemma :

```
lemma zmod'_succ_eq_bUnion :
  zmod' (d*p^(m + 1)) = (zmod' (d*p^m)).bUnion
  (λ a : zmod (d * p ^ m), set.to_finset (equi_class m (m + 1)) a)
```

This lemma says that any element of $\mathbb{Z}/dp^{m+1}\mathbb{Z}$ comes from `equi_class m (m + 1) b` for some $b \in \mathbb{Z}/dp^m\mathbb{Z}$. The proof is now complete with the following lemma :

```
lemma bernoulli_distribution_sum' (x : zmod (d * p^m)) :
  Σ (y : zmod (d * p ^ m.succ)) in
  (λ a : zmod (d * p ^ m), ((equi_class m.succ) a).to_finset) x,
  bernoulli_distribution p d c m.succ y = bernoulli_distribution p d c m x
```

which says, for $x \in \mathbb{Z}/dp^m\mathbb{Z}$, $E_{c,m}(x) = \sum_y' E_{c,m+1}(y)$, for `y ∈ equi_class m (m + 1) x`.

5.5 Integrals

The last piece in the puzzle is the integral. We use the same notation as in the previous section. Given a measure μ, and a function $f \in LC(X, R)$, $\int f d\mu := \mu(f)$. As in Theorem 12.1 of [6], this can be extended to a continuous R-linear map $\int_X f d\mu : C(X, R) \to R$. This follows from the fact that $LC(X, R)$ is dense in $C(X, R)$; as a result, the map from $LC(X, R)$ to $C(X, R)$ is `dense_inducing`, that is, it has dense range and the topology on $LC(X, R)$ is induced from the topology on $C(X, R)$.

The continuity of the extension of the integral follows from the fact that every measure μ is uniformly continuous :

```
lemma uniform_continuous (φ : measures X A) : uniform_continuous ⇑φ
```

5.6 Construction

There are several possible definitions for the p-adic L-function, the most common being a meromorphic function $L_p(s, \chi)$ on
$\{s \in \mathbb{C}_p \mid |s| < p\}$ obtained by analytic continuation, such that

$$L_p(1-n, \chi) = -(1 - \chi\omega^{-n}(p)p^{n-1})\frac{B_{n,\chi\omega^{-n}}}{n}$$

for $n \geq 1$ (Theorem 5.11, [6]). Due to the absence of $\mathbb{C}_p$ in `mathlib` at the time, and the difficulty of showing analytic continuity(even on paper), our definition is instead motivated by Theorem 12.2, [6], which states that, for $s \in \mathbb{Z}_p$, and Dirichlet character χ with conductor dp^m, with $gcd(d,p) = 1$ and $m \geq 0$, for a choice of $c \in \mathbb{Z}$ with $gcd(c, dp) = 1$:

$$(1 - \chi(c)\langle c\rangle^{s+1})L_p(-s, \chi) = \int_{(\mathbb{Z}/d\mathbb{Z})^\times \times \mathbb{Z}_p^\times} \chi\omega^{-1}(a)\langle a\rangle^s dE_c \quad (1)$$

where $\langle a\rangle = \omega^{-1}(a)a$, and $b^s = exp(log_p(b))$ (the exponential and logarithm are defined in terms of power series expansions).

Instead of using the variable s (which takes values in a subset of $\mathbb{C}_p$), we choose to use an element of the weight space, the set of continuous monoid homomorphisms from $\mathbb{Z}/d\mathbb{Z}^\times \times \mathbb{Z}_p^\times$ to R. We replace $\langle a\rangle^s$ with `w:continuous_monoid_hom A`. The advantage is that our p-adic L-function can now be defined over a more general space : a nontrivial normed commutative complete non-Archimedean $\mathbb{Q}_p$-algebra with no zero divisors.

Given a Dirichlet character χ of level dp^m with $gcd(d,p) = 1$ and $m > 0$, we now define the p-adic L-function to be :

$$L_p(w, \chi) := \int_{(\mathbb{Z}/d\mathbb{Z})^\times \times \mathbb{Z}_p^\times} \chi\omega^{-1}(a)w dE_c$$

```
def p_adic_L_function :=
  measure.integral (bernoulli_measure R hc hc' hd na)
  ⟨(units.coe_hom R).comp (dirichlet_char_extend p d R m hd
  (change_level _ (χ.mul ((teichmuller_character_mod_p' p R))))) *
  w.to_monoid_hom, cont_paLf m hd _ w⟩
```

Here, `dirichlet_char_extend` extends χ from $(\mathbb{Z}/dp^m\mathbb{Z})^\times$ to $(\mathbb{Z}/d\mathbb{Z})^\times \times \mathbb{Z}_p^\times$ via the restriction map. The last term `cont_paLf` proves the continuity of the given function, since Lean takes an element of type `C((zmod d)ˣ × ℤ_[p]ˣ, R)`. We have absorbed the constant term given in the LHS of (1). This was done because Theorem 12.2 lets $L_p(-s, \chi)$ take values in $\mathbb{C}_p$. In a general ring R, as we have chosen, division need not exist. One would then need the factor to be a unit, which may not always happen (for example, consider $R = \mathbb{Q}_p$). Thus, our p-adic L-function differs from the original by a constant factor. This factor can be easily removed if one assumes R has division.

6 Evaluation at negative integers

We shall now prove that our chosen definition of the p-adic L-function is equivalent to the original one, that is, it takes the same values at negative integers : for $n > 1$,

$$L_p(1-n, \chi) = -(1 - \chi\omega^{-n}(p)p^{n-1})\frac{B_{n,\chi\omega^{-n}}}{n} \quad (2)$$

For this section, we assume that R is a non-Archimedean normed commutative $\mathbb{Q}_p$-algebra, which is complete, nontrivial, and has no zero divisors. The scalar multiplication structure obtained from $\mathbb{Q}$ and $\mathbb{Q}_p$ are compatible, given by `is_scalar_tower ℚ ℚ_[p] R` (see Section 4.2 of [2]). The prime p is odd, and we choose positive natural numbers d and c which are mutually coprime and are also coprime to p. The Dirichlet character χ has level dp^m, where m is positive. We also assume χ is even and d divides its conductor. Let us first explain why we need the latter condition.

6.1 Factors of the conductor

We explain here why we need d to divide the conductor of χ. In this section, we do not differentiate between the associated Dirichlet character and the Dirichlet character.

Recall that $\chi\omega^{-1}$ actually denotes the Dirichlet character multiplication of χ and ω^{-1}, as explained in Section 3. In order to translate between sums on $\mathbb{Z}/dp^n\mathbb{Z}^\times$ and $\mathbb{Z}/dp^n\mathbb{Z}$, one needs that, for all $x \in \mathbb{Z}/dp^n\mathbb{Z}$ such that x is not a unit, $\chi\omega^{-k}(x) = 0$ for all $k > 0$. This is equivalent to saying, $\forall y \in \mathbb{N}$, such that $gcd(y,d) \neq 1$ and $gcd(y,p) \neq 1$, $gcd(y, (\chi\omega^{-k})$`.conductor`$) \neq 1$.

Given coprime natural numbers k_1, k_2 and a character ψ of level k_1k_2, one can find primitive characters ψ_1 and ψ_2 of levels k_1 and k_2 respectively such that $\psi = \psi_1\psi_2$:

```
lemma eq_mul_of_coprime_of_dvd_conductor {m n : ℕ} [fact (0 < m * n)]
  (χ : dirichlet_character R (m * n)) (hχ : m | χ.conductor)
  (hcop : m.coprime n) : ∃ (χ₁ : dirichlet_character R m)
  (χ₂ : dirichlet_character R n), χ₁.is_primitive ∧ χ =
  χ₁.change_level (dvd_mul_right m n) * χ₂.change_level (dvd_mul_left n m)
```

Thus, given $k > 0$, we can find primitive characters χ_1 and χ_2 with conductors z_1 and z_2 such that $z_1 \mid d$ and $z_2 \mid p^m$ and $\chi_1\chi_2 = \chi\omega^{-k}$. The condition that d divides the conductor of χ ensures that $z_1 = d$. As a result, if $gcd(y,d) \neq 1$, then $gcd(y, z_1z_2) \neq 1$, so $\chi\omega^{-k}(y) = 0$ as needed.

6.2 Main Result

Note that the same result holds when χ is odd or when $p = 2$, the proofs differ slightly. We shall skip most of the details of the proof, since these are heavily computational. We shall instead highlight the key concepts that are used. Our reformulation of (2) is :

```
theorem p_adic_L_function_eval_neg_int_new :
  (p_adic_L_function m χ c na (mul_inv_pow (n - 1))) =
  (algebra_map ℚ R) (1 / n : ℚ) *
  (1 - (χ (zmod.unit_of_coprime c _) *
    (mul_inv_pow n (zmod.unit_of_coprime c hc’, _)))) *
  (1 - ((asso_dirichlet_character
    (χ.mul ((teichmuller_character_mod_p’ p R)^n))) p * p^(n - 1))) *
  (general_bernoulli_number
    (χ.mul ((teichmuller_character_mod_p’ p R)^n)) n)
```

Here, `mul_inv_pow` is our translation of $\langle a \rangle^s$.
The proof consists of two steps : breaking up the integral in the LHS into three sums, and evaluating each of these sums. This is very calculation intensive, and was the longest part of the project. The proof is very similar to the proof of Theorem 12.2 in [6].

Since $LC((\mathbb{Z}/d\mathbb{Z})^\times \times \mathbb{Z}_p^\times, R)$ is dense in $C((\mathbb{Z}/d\mathbb{Z})^\times \times \mathbb{Z}_p^\times, R)$, we observe that the integral $L_p(1-n,\chi)$ is the same as :

$$L_p(1-n,\chi) = \lim_{j\to\infty} \sum_{a\in(\mathbb{Z}/dp^j\mathbb{Z})^\times} E_{c,j}(\chi\omega^{-1}(a)\langle a\rangle^{n-1})$$

$$= \lim_{j\to\infty} \left(\sum_{a\in(\mathbb{Z}/dp^j\mathbb{Z})^\times} \chi\omega^{-n}a^{n-1}\left\{\frac{a}{dp^j}\right\} \right. \tag{3}$$

$$\left. - \sum_{a\in(\mathbb{Z}/dp^j\mathbb{Z})^\times} \chi\omega^{-n}a^{n-1}\left(c\left\{\frac{c^{-1}a}{dp^j}\right\}\right)\right) \tag{4}$$

$$+\left(\frac{c-1}{2}\right)\sum_{a\in(\mathbb{Z}/dp^j\mathbb{Z})^\times}\chi\omega^{-n}a^{n-1}\Bigg) \tag{5}$$

Going from the first equation to the second took about 600 lines of code, which can be found in `neg_int_eval.lean`. While the proof (on paper) is only a page long, this is very calculation heavy in Lean, because one needs to shift between elements coerced to different types, such as $\mathbb{Z}/(dp^j)\mathbb{Z}$, $\mathbb{Z}/d\mathbb{Z}\times\mathbb{Z}/p^j\mathbb{Z}$, $\mathbb{Z}/d\mathbb{Z}\times\mathbb{Z}_p$, R and their units. Moreover, when each of these types occur as locally constant or continuous functions, one needs to separately prove that each of these functions is also (respectively) locally constant or continuous. Other difficulties include several different ways to obtain the same term, such as `equiv.inv_fun`, `equiv.symm`, `ring_equiv.symm` and `ring_equiv.to_equiv.inv_fun`. We have constructed several lemmas to simplify traversing between these terms.

Each of these sums are then evaluated separately. The first sum in (3) follows from Theorem 1, after translations between `zmod (d * p^n)`$^\times$ and `finset.range (d * p^n)`. This is done by the following lemma, which says

$$\mathbb{Z}/dp^k\mathbb{Z}\simeq\{x\in\mathbb{N}\mid gcd(x,d)\neq 1\}\cup\{x\in\mathbb{N}\mid gcd(x,p)\neq 1\}\cup(\mathbb{Z}/dp^k\mathbb{Z})^\times$$

```
lemma helper_U_3 (x : ℕ) : range (d * p^x) =
  finite.to_finset (finite_of_finite_inter
    (range (d * p^x)) ({x | ¬ x.coprime d}))
  ∪ ((finite.to_finset (finite_of_finite_inter
    (range (d * p^x)) ({x | ¬ x.coprime p})))
  ∪ finite.to_finset (finite_of_finite_inter
    (range (d * p^x)) ({x | x.coprime d} ∩ {x | x.coprime p})))
```

Each of these are made to be a `finset`, since `finset.sum` requires the sum to be over a `finset`.We use this lemma to break our sum over `finset.range (d * p^n)` into units and non-units. The condition that d divides the conductor is then used to show that the associated Dirichlet character is 0 everywhere on the non-units. These calculations can be found in `lim_even_character_of_units.lean`.

Evaluating the middle sum (4) is the most tedious. It is first broken into two sums, so that the previous result can be used. Then, a change of variable from a to $c^{-1}a$ is applied. The variable c is coerced to $\mathbb{Z}/dp^{2k}\mathbb{Z}$, increasing the number of coercions significantly, thus lengthening the calculations. This can be found in `second_sum.lean`.

Finally, the last sum (5) is 0. This is where one uses that χ is even. This follows from Theorem 2. On paper, it is a one-line proof, done by substituting a in the summand with $-a$ and doing caluclations mod p^n. However, since we work in a more general setting, we must go through lengthy roundabout ways instead.
Putting these sums together concludes the proof.

7 Conclusion

7.1 Analysis

We list some of the observations that arose while working on this paper.
The tactic `rw` does not always work inside sums. As a result, one must use the `conv` tactic to get to the expression inside the sum. While using the `conv` tactic, one is said to be working in `conv` mode. Using the `conv` tactic not only lengthens the proof, but also limits the tactics one can use; Another way around sums is to use `simp_rw`, however, this increases compilation time of the proof. Moreover, `simp_rw` rewrites the lemma as many times as applicable, and is an unsuitable choice if one wants to apply the lemma just once.

Another recurring problem was the ratio of implicit to explicit variables. The p-adic L-function, for example, has 19 arguments, of which 7 are explicit, and p, d and R are implicit. Excluding R often means that either Lean guesses or abstracts the correct term, or it asks for them explicitly. In the latter case, one also gets as additional goals all the hypotheses that are dependent on R and implicit, such as `normed_comm_ring R`. The other alternative is to explicitly provide terms using `@`, however this leads to very large expressions.

We also ran into some instance errors. For example, since `char_zero` is a class, we would like to give the lemma `char_zero R` an `instance` structure. However, the proof is dependent on `R` having the `[algebra ℚ_[p] R]` structure. Lean would then claim that this is a dangerous instance (for p being an explicit variable) and that p is a `metavariable` (for p being an implicit variable). Thus, we made it a `lemma` instead, and had to explicitly feed it into implicit arguments.

While most properties regarding Bernoulli numbers and polynomials and locally constant functions have been put into `mathlib`, the rest of the work is on a private repository. The author hopes to push the work directly to Lean 4, once the required port is complete.

7.2 Statistics

Given the decentralized nature of `mathlib`, it is quite difficult to calculate the number of lines of code already existing in `mathlib` which were used in this project. When initially completed, this project had about 15000 lines of code. A major refactor was then conducted, in an effort to reduce length of individual proofs. We tried to uphold the spirit of `mathlib`, constructing lemmas in as much generality as possible. The code currently consists of 28 files and about 7500 lines, grouped into appropriate categories where possible, according to the sections of this paper.

7.3 Related work

There are several projects that require Dirichlet characters and properties of the p-adic integers. These include the project on the formalization of Fermat's last theorem for regular primes[2]. There is also an effort by Prof David Loeffler which involves formalization of the classical Dirichlet L-function, that is somewhat dependent on this work. Our work on Bernoulli numbers has been used to give a formal proof of Faulhaber's theorem.

In the future, the author hopes to be able to work on Iwasawa theory, for which the p-adic L-function is a key ingredient. She also hopes to formalize more properties of Bernoulli numbers, that are a fundamental component of number theory.

References

1 Jeremy Avigad, Leonardo de Moura, and Soonho Kong. Theorem proving in lean, Jun 2018. URL: https://kilthub.cmu.edu/articles/journal_contribution/Theorem_Proving_in_Lean/6492902/1, doi:10.1184/R1/6492902.v1.

2 Anne Baanen, Sander R. Dahmen, Ashvni Narayanan, and Filippo A. E. Nuccio Mortarino Majno di Capriglio. A formalization of dedekind domains and class groups of global fields. *Journal of Automated Reasoning*, 66(4):611–637, Nov 2022. doi:10.1007/s10817-022-09644-0.

3 Johan Commelin and Robert Y. Lewis. Formalizing the ring of witt vectors. In *Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs*. ACM, jan 2021. URL: https://doi.org/10.1145%2F3437992.3439919, doi:10.1145/3437992.3439919.

4 Tomio Kubota and Heinrich-Wolfgang Leopoldt. Eine p-adische Theorie der Zetawerte. I. Einführung der p-adischen Dirichletschen L-Funktionen. *J. Reine Angew. Math.*, 214(215):328–339, 1964.

5 The mathlib Community. The Lean mathematical library. In J. Blanchette and C. Hriţcu, editors, *CPP 2020*, page 367–381. ACM, 2020. doi:10.1145/3372885.3373824.

6 Lawrence C. Washington. *Introduction to cyclotomic fields*, volume 83 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, second edition, 1997. doi:10.1007/978-1-4612-1934-7.

[2] https://github.com/leanprover-community/flt-regular