Formalisation of p-adic L-functions in Lean 3

Ashvni Narayanan
London School of Geometry and Number Theory

Abstract

This paper decribes formalisation of p-adic L-functions in an interactive theorem prover Lean 3. p-adic L-functions are fundamental number theoretic objects. These special functions emerge from the special values they take at negative integers in terms of generalized Bernoulli polynomials. We formalise their definition in terms of a p-adic integral with respect to the Bernoulli measure, and prove 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

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

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

I am working on formalising 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 [?]. Lean is a powerful tool that facilitates formalisation 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` currently contains 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. I am working towards formalising the statement of the Iwasawa Main Conjecture. The statement unifies an “algebraic side", coming from characteristic polynomials, with an “analytic side", coming from p-adic L-functions. This article focuses on the formalisation of p-adic L-functions in Lean. This has never been done before in any theorem prover. As a result, one needs to build a lot of background (in the maximum possible generality) before embarking on the main goal. The project is complete.

p-adic L-functions are a very well studied number theoretic object. They were initially constructed by Kubota and Leopoldt in [?]. 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. I refer to the constructions given in Chapter 12 of [?]. An optimal definition is one that minimizes the amount of code needed to obtain the required properties, and that is the “most general", so that it can be used to its full potential. I have chosen a definition in terms of p-adic integrals because it seems to be the closest to what I need for stating the Iwasawa Main Conjecture. Moreover, it minimizes the code needed to show analytic continuity of several functions that would be needed in other definitions.

The following tools are needed to construct p-adic L-functions: Bernoulli numbers and polynomials, locally compact Hausdorff totally disconnected spaces, and the p-adic integers and its topological properties. I introduce these, and then define the p-adic L-function, finishing with a summary of what has been accomplished and what will be.

All the code discussed in this article can be found on the branch p-adic[1]. The most current version of `mathlib` (which is not the one I am using) is [?].

1.1 Mathematical overview

We shall give a brief overview of the mathematics formalised in this project. L-functions are a fundamental object, appearing almost everywhere in modern number theory. There are several types of L-functions. The L-function attached to an elliptic curve gives information about its rank. The Dirichlet L-function associated to a 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 Reimann 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 try to construct, for an integer prime p, a p-adic analogue of $L(s,\chi)$, called the Kubota-Leopoldt p-adic L-function, denoted $L_p(s,\chi)$. We fix, for the rest of this paper, a prime p. 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 p-adic integers, $\mathbb{Z}_p$. In fact, $L_p(s,1)$ is analytic except for a pole at $s = 1$ with residue $1 - \frac{1}{p}$ (Theorem 5.11, Washington).

Due to several reasons explained in the following sections, formalisation of the p-adic L-functions via analytic continuation would be hard. Thus, we chose to formalise it in as a measure theoretic object. Following Washington, we define it in terms of a "p-adic integral" with respect to the Bernoulli measure. We shall 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, $\mathbb{Z}_p = \operatorname{proj}\lim_n \mathbb{Z}/p^n\mathbb{Z}$. This is equivalent to saying that there are surjective maps $\phi_n : \mathbb{Z}_p \to \mathbb{Z}/p^n\mathbb{Z}$, such that .... A corollary of this is that $\mathbb{Z}_p$ has a topological basis of clopen sets (both open and closed), $(U_{a,n})_{(a,n)}$ for $n \in \mathbb{N}$ and $a \in \mathbb{Z}/p^n\mathbb{Z}$, where $U_{a,n} := \{x \in \mathbb{Z}_p | x \equiv a (\text{mod } p^n)\}$.

Given a profinite space X and a normed ring (?) R, one can show that the locally constant functions (preimage of any set is open) 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 p-adic 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 p-adic measure is a Bernoulli measure. Given a clopen set $U_{a,n}$ of $\mathbb{Z}_p$, the characteristic function $\chi_{a,n}$ (defined to be 1 on $U_{a,n}$ and 0 otherwise) is a locally constant function. We then define the Bernoulli measure E_c to be :

$$E_c(U_{a,n}) := ...$$

We shall show later that this is sufficient to define E_c on all locally constant functions, and that E_c is indeed a measure. Given a p-adic measure μ, we define the p-adic integral with respect to μ to be $\int f d\mu := \mu(f)$. Since $LC(X,R)$ is dense in

[1] https://github.com/leanprover-community/mathlib/tree/p-adic

$C(X, R)$, we can then extend the definition of the p-adic integral to $C(X, R)$. In fact, this is an R-linear map.

Finally, the p-adic L-function is defined to be a p-adic integral with respect to the Bernoulli measure.
The characterizing property of the p-adic L-function is its evaluation at negative integers. If defined as an analytic continuation, this would follow directly. However, when defined as a p-adic integral, additional work is needed to prove this. The equivalence of these two definitions is proved in Theorem 12.2 of Washington. The same theorem also helps show the independence of the theorem from the additional variable c.
In this paper, we formalise this definition of the p-adic L-function and prove its values at negative integers in terms of the generalized Bernoulli numbers. Moreover, this is done in generality, taking values in a normed complete $\mathbb{Q}_p$-algebra, instead of just $\mathbb{C}_p$.

1.2 Lean and `mathlib`

Lean 3 is a functional programming language and interactive theorem prover based on dependent type theory, with proof irrelevance and non-cumulative universes. For an introduction to Lean, see for instance .

This project is based on Lean's mathematical library mathlib, which is characterized by its decentralized nature with over 300 contributors. Due to the distributed organization of mathlib, it is impossible to cite every author who contributed a piece of code that we used. However, we remark that our formalisation makes extensive use of the theory of Dedekind domains and of the theory of uniform spaces and completions, originally developed in the perfectoid space formalisation project.

In Lean's core library and mathlib, type classes are used to handle mathematical structures on types. For example, the type class ring packages two operations, addition and multiplication, as well as a list of properties they must satisfy. Then, given a type R, we can declare an instance [ring R], and Lean's instance resolution procedure will infer that R has a ring structure. Besides instance, whose behaviour we have just described, we use in this paper the keywords variables, def, lemma and theorem, which have the evident meaning.
In this section, we define some fundamental concepts which shall be used throughout this article. In a lot of these cases, the Lean definition is far more generalized (and hence different) than the usual mathematical definition. However, there is always a database of theorems associated to a definition, which connect it to the usual mathematical definition. So we sometimes skip the Lean definition and instead explain a theorem/lemma that shows an equivalence of the Lean definition with the mathematical one.

A subset s of a topological space has the property `is_compact` if for every filter f containing s, there exists $a \in s$ such that a is a limit point of f. This technical definition is not very easy to work with. Instead, we use the following lemma that gives an (equivalent) definition :

```
lemma is_compact.elim_finite_subcover
{α : Type u} [_inst_1 : topological_space α] {s : set α} {ι : Type v}
(hs : is_compact s) (U : ι → set α)
(hUo : ∀i, is_open (U i)) (hsU : s ⊆ ⋃ i, U i) :
∃ t : finset ι, s ⊆ ⋃ i ∈ t, U i
```

This lemma states that a subset s of a topological space is *compact* if for every cover $s \subseteq \cup_i U_i$ of s by open sets U_i, there exists a finite subcover $U_1, \ldots, U_m$ such that $s \subseteq \cup_{i=1}^m U_i$. A `compact_space` is a topological space which satisfies the property `is_compact` when thought of as a set :

```
class compact_space (α : Type*) [topological_space α] :=
(compact_univ : is_compact (univ : set α))
```

Note that, in Lean, a topological space is a term of a type, and is not automatically identified as a set. However, it can contain subsets, and `univ` denotes the universal set, or the topological space, thought of as a subset of itself. The difference between `is_compact univ` and `compact_space` α is subtle : `is_compact` takes as input a term of a type (which is not a type), while `compact_space` takes as input a type. These distinctions are a consequence of Lean being based on type theory, instead of set theory.

A topological space X is a *locally compact space* if for every element x of X, every neighbourhood n of x(a set containing an open set containing x) contains a compact neighbourhood of x :

```
  class locally_compact_space (α : Type*) [topological_space α] : Prop :=
  (local_compact_nhds : ∀ (x : α) (n ∈ 𝒩 x), ∃ s ∈ 𝒩 x, s ⊆ n ∧ is_compact s)
```

A set s of a topological space is *totally separated* if it has the property `is_totally_separated` : $\forall x, y \in s$ with $x \neq y$, there exist disjoint open sets U, V such that $x \in U$, $y \in V$, and $s \subseteq U \cup V$:

```
  def is_totally_separated {α : Type u} [topological_space α] (s : set α) : Prop :=
∀ x ∈ s, ∀ y ∈ s, x ≠ y → ∃ u v : set α, is_open u ∧ is_open v ∧ x ∈ u ∧ y ∈ v ∧ s ⊆ u ∪ v ∧
   u ∩ v = ∅
```

A topological space is `totally_separated` if the universal set of the space `univ` satisfies `is_totally_separated`. A set s of a topological space is called *totally disconnected* if it satisfies `is_totally_disconnected` : Every subset which has no non-trivial open partition is atmost a singleton. A topological space is `totally_disconnected` if it is totally disconnected as a subset of itself. An equivalent statement is, A space is totally disconnected iff its connected components are singletons :

```
lemma totally_disconnected_space_iff_connected_component_singleton
{α : Type u} [topological_space α] :
  totally_disconnected_space α ↔ ∀ x : α, connected_component x = {x}
```

A *profinite* space is a compact Hausdorff space which is totally disconnected :

```
structure Profinite :=
(to_CompHaus : CompHaus)
[is_totally_disconnected : totally_disconnected_space to_CompHaus]
```

A function on a topological space satisfies `is_locally_constant` if the preimage of every set is open :

```
def is_locally_constant {X Y : Type*} [topological_space X] (f : X → Y) : Prop :=
∀ s : set Y, is_open (f ⁻¹' s)
```

Lean has `locally_constant X Y`, which is a type of functions bundled with the property `is_locally_constant` :

```
structure locally_constant (X Y : Type*) [topological_space X] :=
(to_fun : X → Y)
(is_locally_constant : is_locally_constant to_fun)
```

A term `f'` of type `locally_constant X A`, looks like `f' = ⟨f, hf⟩`, where `f` is the function `f:X → A`, and `hf` is a proof that `f` is locally constant, that is, `hf:is_locally_constant f`. For this article, we use the notation $LC(X, A)$ to denote the set of locally constant functions from X to A.

Similarly, the type of *continuous functions* on topological spaces $\alpha \to \beta$, denoted $C(\alpha, \beta)$ here and `C(α,β)` in Lean, is defined as :

```
structure continuous_map (α : Type*) (β : Type*)
  [topological_space α] [topological_space β] :=
(to_fun             : α → β)
(continuous_to_fun  : continuous to_fun)
```

A *clopen set* of a topological space is a set which is both open and closed. This property is given by `is_clopen`. There is a subtype of clopen sets :

```
  def clopen_sets (H : Type*) [topological_space H] :=
  {s : set H // is_clopen s}
```

Given `U:clopen_sets X` for a topological space `X`, `U.val` is the set, and has type `set H`, and `U.prop` holds the property satisfied by `U`, that is, `U.prop:is_clopen U`.

The *characteristic function* of `U` on `X`, `char_fn X U`, takes value 1 for every element of `U`, and 0 otherwise. Characteristic functions of clopen sets are locally constant :

```
  def char_fn {R : Type*} [topological_space R] [ring R] [topological_ring R]
  (U : clopen_sets X) : locally_constant X R :=
  { to_fun := λ x, by classical; exact if (x ∈ U.val) then 1 else 0,
    is_locally_constant := _ }
```

A *Dirichlet character mod* n is a multiplicative group homomorphism $(\mathbb{Z}/n\mathbb{Z})^\times \to \mathbb{C}^\times$. This induces a Dirichlet character mod kn for all $k > 0$. The smallest such n is called the *conductor* of the character. A Dirichlet character mod n is called *primitive* if it has conductor n. In this article, all Dirichlet characters are assumed to be primitive.

For a `topological_group` A, define the *weight space*, denoted `weight_space`, to be the continuous monoid homomorphisms from $(\mathbb{Z}/d\mathbb{Z})^\times \times \mathbb{Z}_p^\times$ to $A^\times$, with $gcd(d, p) = 1$:

```
  structure weight_space extends monoid_hom ((units (zmod d)) × (units ℤ_[p])) A,
    C((units (zmod d)) × (units ℤ_[p]), A)
```

One of the aspects of defining p-adic L-functions in Lean is to understand its type. In [?], it takes values from a subset of $\mathbb{C}_p$ (the p-adic completion of the algebraic closure of the fraction field of $\mathbb{Z}_p$) and Dirichlet characters. Since $\mathbb{C}_p$ has not been defined in Lean yet, my attempt has been to find a space to replace it. According to this construction, the domain can be replaced with the product of the weight space and the space of Dirichlet characters. The range is contained in a commutative and complete normed ring with a coercion from $\mathbb{Z}_p$, which is denoted by R.

2 Preliminaries

2.1 Modular arithmetic and $\mathbb{Z}_p$

Some fundamental objects with which we shall work throughout are the finite spaces $\mathbb{Z}/n\mathbb{Z}$ for all $n \in \mathbb{N}$. Given $n \in \mathbb{N}$, $\mathbb{Z}/n\mathbb{Z}$ is the same as `fin n`, the set of natural numbers upto n. This has a natural group structure, and is given the discrete topology, making it a topological group. Some of the maps used constantly include `val:zmod n → ℕ`, which takes any element to its smallest reprentative in the equivalence class, 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 $m, n \in \mathbb{N}$ with m and n coprime, an important equivalence is given by the Chinese Remainder Theorem : `chinese_remainder:zmod (m * n) ≃+* zmod m × zmod n`. About 30 additional lemmas were required, which have been put in a separate file, `zmod_properties.lean`.

For this article, we shall assume p denotes a prime and d is a positive natural with $gcd(d,p) = 1$. The *ring of p-adic integers*, denoted `ℤ_[p]`, is the completion of $\mathbb{Z}$ by the p-adic valuation. It has the following properties :

- As a profinite limit, $\mathbb{Z}_p = \varprojlim \mathbb{Z}/p^n\mathbb{Z}$. As a result, one can find (compatible) surjective ring homomorphisms `to_zmod_pow : ℤ_[p] →+* zmod (p^n)` for all $n \in \mathbb{N}$.
- As a topological space, $\mathbb{Z}_p$ has the profinite topology, induced by discrete topology on the finite sets $\mathbb{Z}/p^n\mathbb{Z}$. Hence, $\mathbb{Z}_p$ is a compact, Hausdorff totally disconnected space with a clopen basis of the form $a + p^n\mathbb{Z}_p$, with $a \in \mathbb{Z}/p^n\mathbb{Z}$ for every n.
- For $p > 2$, $\mathbb{Z}_p^\times \simeq (\mathbb{Z}/p\mathbb{Z})^\times \times (1+p\mathbb{Z}_p)$[2]. By the Chinese Remainder Theorem, we have, for $gcd(d,p) = 1$, $(\mathbb{Z}/d\mathbb{Z})^\times \times \mathbb{Z}_p^\times \simeq (\mathbb{Z}/dp\mathbb{Z})^\times \times (1 + p\mathbb{Z}_p)$

These properties characterize the p-adic integers, and are integral to this work. While some preexisted in `mathlib`, about 40 additional lemmas have been proved in `padic_int_properties.lean`. Moreover, lots of facts regarding the clopen basis given above have been utilized and proved in `padic_int_clopen_properties.lean`. (add links)

2.2 Dirichlet characters and the Teichmüller character

An important task was to formalise Dirichlet characters and its properties. They are an integral part of the definition of the p-adic L-function. The Teichmüller character, especially, plays a significant role. This is novel, in the sense that Dirichlet characters are often not found to be defined in this technical manner in most texts. Another addition is the definition of Dirichlet characters of level and conducor 0. We also generalize the definition from group homomorphisms to $\mathbb{C}$ to monoid homomorphisms on any `comm_monoid_with_zero`. In this section, the words character and Dirichlet character are used interchangeably.

The Dirichlet characters are usually defined as group homomorphisms from $\mathbb{Z}/n\mathbb{Z}$ to $\mathbb{C}$ for some natural number n. We can then assign levels to Dirichlet characters and relate Dirichlet characters of different levels, constructing a "compatible" system of characters for certain values of n.

For any two monoids M and M', `M →* M'` denotes the type of monoid homomorphisms from M to M'. Thus, making a separate `def` for Dirichlet characters does not make sense. Instead, we use :

```
abbreviation dirichlet_character (R : Type*) [monoid R] (n : ℕ) :=
  units (zmod n) →* units R
abbreviation lev {R : Type*} [monoid R] {n : ℕ} (χ : dirichlet_character R n) :
  ℕ := n
```

In other words, given a monoid R, the space of Dirichlet characters of level n is the space of monoid homomorphisms from $(\mathbb{Z}/n\mathbb{Z})*$ to $R*$. Given a Dirichlet character $\chi : (\mathbb{Z}/n\mathbb{Z})* \to R*$, `χ.lev` returns the level n. Note that the linter returns an extra unused argument warning for the latter definition.

Another way of thinking about the Dirichlet characters of level n is to think of them as multiplicative functions on $\mathbb{Z}$ which are periodic with period dividing n. To incorporate that definition, we defined :

```
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,
  map_one' := _,
  map_mul' := _, }
```

[2] $\mathbb{Z}_2^\times \cong \{\pm 1\} \times (1 + 4\mathbb{Z}_2)$

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. Most of our work is on $\mathbb{Z}/n\mathbb{Z}$ instead of its units, hence this is a vital definition. The following useful theorem relates the two definitions :

```
lemma asso_dirichlet_character_eq_char {R : Type*} [monoid_with_zero R] {n : ℕ}
(χ : dirichlet_character R n) (a : units (zmod n)) :
asso_dirichlet_character χ a = χ a
```

2.2.1 Changing levels, primitivity and multiplication

As mentioned earlier, one would like to shift between compatible Dirichlet characters of different levels. For this, we have the following tools :

```
/-- Extends the Dirichlet character χ of level n to level m, where n ∣ m. -/
def change_level {m : ℕ} (hm : n ∣ m) : dirichlet_character R m :=
  χ.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)
```

`classical.some` makes an arbitrary choice of an element from a space, if the space is nonempty, and `classical.some_spec` lists down the properties of this random element coming from the space. In particular, we can use `classical.some (χ.factors_through d).ind_char` to extract χ_0, and `classical.some_spec (χ.factors_through d).ind_char` to get `χ = χ₀.change_level dvd`.

With the assistance of a few lemmas, it is easy to translate between `change_level` and `factors_through`. The notion of primitivity and conductor of a Dirichlet character now follows easily :

```
 /-- The set of natural 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 Dirichlet character is periodic.
 The Dirichlet character χ can then alternatively be reformulated on ℤ/nℤ. -/
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
```

Note that our definition of Dirichlet characters holds also for $n = 0$, ie, on $\mathbb{Z}$. This is easily separated from the rest by the lemma :

```
lemma conductor_eq_zero_iff_level_eq_zero : χ.conductor = 0 ↔ n = 0
```

An issue with the definition `is_primitive` is that it equates the levels of of two Dirichlet characters, however, this does not imply the types are equal. As an example, consider natural numbers a and b such that $a = b$. Then Lean does not identify `dirichlet_character R a` and `dirichlet_character R b` as the same. In fact, it is best avoided to make lemmas about equality of types. One way to deal with this is to use the `subst` tactic, which substitutes a with

b everywhere, however, this does not work here. Also, we can show that they are multiplicatively equivalent (this is a bijection which preserves multiplication) :

```
  /-- If m = n are positive natural numbers, then their Dirichlet character spaces are equivalent.
     -/
def equiv {a b : ℕ} (h : a = b) :
  dirichlet_character R a ≃* dirichlet_character R b
```

This is especially problematic when we must prove a theorem for a general n, because, for $n = 0$, Lean does not automatically identify `zmod n` with $\mathbb{Z}$.
Once we have the notion of primitivity, we can define multiplication of Dirichlet characters. Traditionally, this is defined only for primitive characters. We extend the definition for any two characters:

```
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 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 nm.

2.2.2 Additional properties

Finally, one needs the notion of odd and even characters. A character χ is odd if $\chi(-1) = -1$, and even otherwise (in this case, $\chi(-1) = 1$). If the target is a commutative ring, then 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
```

The proof is simple : $\psi(-1)^2 = 1$ must imply that $\psi(-1)^2 - 1^2 = (\psi(-1) - 1)(\psi(-1) + 1)$. This lemma, called `sq_sub_sq` is where commutativity of the ring is needed. To conclude that one of the factors must be 0, we need the lemma `mul_eq_zero`, which requires that S has no zero divisors.
An important consequence of this is :

```
lemma asso_odd_dirichlet_character_eval_sub (x : zmod m) (hψ : ψ.is_odd) :
  asso_dirichlet_character ψ (m - x) = -(asso_dirichlet_character ψ x)
lemma asso_even_dirichlet_character_eval_sub (x : zmod m) (hψ : ψ.is_even) :
  asso_dirichlet_character ψ (m - x) = (asso_dirichlet_character ψ x)
```

Other important properties include :

```
/-- Dirichlet characters are continuous. -/
lemma dirichlet_character.continuous {R : Type*} [monoid R] [topological_space R]
  {n : ℕ} (χ : dirichlet_character R n) : continuous χ
/-- Associated Dirichlet characters are continuous. -/
lemma dirichlet_character.asso_dirichlet_character_continuous
  {R : Type*} [monoid_with_zero R] [topological_space R] {n : ℕ}
  (χ : dirichlet_character R n) : continuous (asso_dirichlet_character χ)
/-- Associated Dirichlet characters are bounded. -/
lemma dirichlet_character.asso_dirichlet_character_bounded {R : Type*}
  [monoid_with_zero R] [normed_group R] {n : ℕ} [fact (0 < n)]
  (χ : dirichlet_character R n) : ∃ M : ℝ,
  ‖(⟨asso_dirichlet_character χ, χ.asso_dirichlet_character_continuous⟩ :
```

```
  C(zmod n, R))‖ < M
```

In the last lemma, the norm is the sup norm on $C(\mathbb{Z}/n\mathbb{Z}, R)$.
These are all the ingredients we needed. Let us now define a special Dirichlet character, the Teichmüller character.

2.2.3 Teichmüller character

The initial effort was to formalise the definition of the Teichmüller character directly. However, it was discovered that Witt vectors, and in particular Teichmüller lifts had previously been added to `mathlib`. This was used and was very helpful. It 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. 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 that

```
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`.

Note that given a multiplicative homomorphism `R →* S` for monoids `R` and `S`, one can obtain a multiplicative homomorphism `units R →* units S`. This translation is done by `units.map`. Combining this with the previous two definitions, we obtain our definition of the Teichmüller character from $(\mathbb{Z}/p\mathbb{Z})^\times \to \mathbb{Z}_p$:

```
/-- The Teichmuller character defined on units ℤ/pℤ. -/
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))
```

Often we shall view this as taking values in a $\mathbb{Q}_p$-algebra R, by composing it with the algebra map `algebra_map ℤ_[p] R`, which identifies elements of $\mathbb{Q}_p$ in R.

One of the important properties of Teichmüller characters is that for odd primes p, the character is primitive :

```
lemma is_primitive_teichmuller_character_zmod_p (p : ℕ) [fact (nat.prime p)]
  (hp : 2 < p) : (teichmuller_character_mod_p p).is_primitive
```

The proof is as follows : The conductor must divide the level p, hence it must be 1 or p. Thus it suffices to prove that for $2 < p$, the conductor is not 1. The result then follows from :

```
lemma conductor_eq_one_iff {n : ℕ} (χ : dirichlet_character R n) (hn : 0 < n) :
  χ = 1 ↔ χ.conductor = 1
lemma teichmuller_character_mod_p_ne_one (p : ℕ) [fact (nat.prime p)] (hp : 2 < p) :
  teichmuller_character_mod_p p ≠ 1
```

The proof of the latter uses the fact that the Teichmüller character is injective (follows from properties of the Teichmüller lift), while 1 is not. For $p = 2$, we know the Teichmüller character is 1 :

```
lemma teichmuller_character_mod_p_two : teichmuller_character_mod_p 2 = 1
```

2.3 Bernoulli polynomials and the generalized Bernoulli number

The Bernoulli polynomials, an important number theoretic object, are a generalization of Bernoulli numbers. They occur as special values of p-adic L-functions. The Bernoulli numbers B_n are generating functions given by :

$$\sum B_n \frac{t^n}{n!} = \frac{t}{e^t - 1}$$

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 $B_1 = -\frac{1}{2}$. On using the Taylor series expansion for e^t and equating coefficients, one gets,

$$B_n = 1 - \sum_{k=0}^{n-1} \binom{n}{k} \frac{B_k}{n-k+1}$$

In `mathlib`, B_n was already defined (by Johan Commelin) as :

```
bernoulli' n = 1 - Σ k : fin n, n.choose k / (n - k + 1) * bernoulli' k
```

However, we needed B'_n, which was 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 :

$$\sum_{n=0}^{\infty} B_n(X) \frac{t^n}{n!} = \frac{te^{tX}}{e^t - 1}$$

A calculation similar to the one above gives :

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

We now define the Bernoulli polynomials as :

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

Here, `polynomial.monomial n a` translates to aX^n. 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.

2.3.1 Properties

The following properties of Bernoulli polynomials were proved :

1. $B_0(X) = 1$:

```
    lemma polynomial.bernoulli_zero : bernoulli 0 = 1
```

For a polynomial f over a ring R, `f.eval x` translates to $f(x)$.

2. $B_n(0) = B_n$:

```
    lemma polynomial.bernoulli_eval_zero (n : ℕ) :
    (bernoulli n).eval 0 = bernoulli n
```

3. $B_n(1) = B'_n$:

```
    lemma polynomial.bernoulli_eval_one (n : ℕ) :
    (bernoulli n).eval 1 = bernoulli' n
```

The theorem `power_series.mk` (abbreviated as `mk`) defines a formal power series in terms of its coefficients, that is, $\sum_{n=0}^{\infty} a_n X^n$ translates to `mk (λ n, a_n)`.

4. For a commutative $\mathbb{Q}$-algebra A and $t \in A$,

$$\left(\sum_{n=0}^{\infty} B_n(t)\frac{X^n}{n!}\right)(e^X - 1) = Xe^{tX} :$$

```
  theorem polynomial.bernoulli_generating_function (t : A) :
  mk (λ n, aeval t ((1 / n! : ℚ) · bernoulli n)) * (exp A - 1) =
    X * rescale t (exp A)
```

The symbol · represents scalar multiplication of $\mathbb{Q}$ on $\mathbb{Q}$-polynomials. The exponential function e^x, which is defined as a Taylor series expansion, takes as input A, but not x. In order to define e^{tx}, we need to use (the ring homomorphism) `rescale y`, which, for an element y of a commutative semiring B, takes a formal power series over B, say $f(X)$, to $f(yX)$.

The proof of the last theorem involves equating the n^{th} coefficients of the RHS and the LHS. After differentiating between n zero and nonzero, one requires the following lemma to complete the nonzero case :

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

The proof of this theorem follows from the following property of Bernoulli numbers :

```
theorem sum_bernoulli (n : ℕ):
Σ k in range n, (n.choose k : ℚ) * bernoulli k = if n = 1 then 1 else 0
```

This follows from the analogous theorem `sum_bernoulli'`, whose proof follows from rearranging sums and the definition of `bernoulli'`.

In order to prove properties of generalized Bernoulli numbers, we needed the following theorem :

```
/-- Bernoulli polynomials multiplication theorem :
For k ≥ 1, B_m(k*x) = Σ i in range k, B_m (x + i / k).  -/
theorem bernoulli_eval_mul' (m : ℕ) {k : ℕ} (hk : k ≠ 0) (x : ℚ) :
  (bernoulli m).eval ((k : ℚ) * x) =
  k^(m - 1 : ℤ) * Σ i in range k, (bernoulli m).eval (x + i / k)
```

There were several different approaches to the proof. Induction on any of the 3 variables did not work out either. We did end up with a proof using Faulhaber's theorem (the proof has been formalised in Lean) for `x:ℕ`, which could not be easily generalized to `x:ℚ`. Finally, we used the generating function equality, which makes the proof calculation heavy. That is, it suffices to show that

$$\sum_n \sum_{i=0}^{k-1} \frac{k^{n-1}}{n!} B_n\left(x+\frac{i}{k}\right) = (e^{kx}-1)\sum_n \frac{B_n(kx)}{n!}$$

We can now define the generalized Bernoulli numbers, the special values p-adic L-functions take at negative integers.

2.3.2 Generalized Bernoulli numbers

Given a primitive Dirichlet character χ of conductor f, let us now define the generalized Bernoulli numbers (section 4.1, [?]) :

$$\sum_{n=0}^{\infty} B_{n,\chi}\frac{t^n}{n!} = \sum_{a=1}^{f} \frac{\chi(a)te^{at}}{e^{ft}-1}$$

For any multiple F of f, Proposition 4.1 of [?] 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, modulo a dependency on the argument F. Taking $F = f$, we chose to formalise this to be our definition in Lean :

```
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 ((polynomial.bernoulli m).eval (a.succ / ψ.conductor : ℚ)))
```

This definition is for all characters, contrary to the typical definition, which is only for primitive characters. Note that the sum ranges from 0 to f. This does not make a difference since the values of the summand at 0 and f are the same (zero). Also note that the Dirichlet character ψ takes values in a general commutative semiring (explain semiring) which is a $\mathbb{Q}$-algebra, and (the ring homomorphism) `algebra_map ℚ S` identifies elements of $\mathbb{Q}$ in S. 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). Automatically, the power changes from `nat.pow` to `zpow`. Finally, note that the primitive Dirichlet character associated to ψ was chosen; we would have obtained incorrect values if we chose `ψ.asso_dirichlet_character` instead.

The following are some important properties of generalized Bernoulli numbers :

```
/-- B_{n,1} = B_n, where 1 is the trivial Dirichlet character of level 1. -/
lemma general_bernoulli_number_one_eval {n : ℕ} :
  general_bernoulli_number (1 : dirichlet_character S 1) n =
  algebra_map ℚ S (bernoulli' n)
/-- Showing that the definition of general_bernoulli_number is independent of F,
  where F is a multiple of the conductor. -/
lemma eq_sum_bernoulli_of_conductor_dvd {F : ℕ} [hF : fact (0 < F)] (m : ℕ)
  (h : ψ.conductor | F) : general_bernoulli_number ψ m =
  (algebra_map ℚ S) (F^(m - 1 : ℤ)) * (Σ a in finset.range F,
  asso_dirichlet_character ψ.asso_primitive_character a.succ *
  algebra_map ℚ S ((polynomial.bernoulli m).eval (a.succ / F : ℚ)))
```

The latter lemma proves Proposition 4.1 of [?]. An important ingredient of the proof is :

```
/-- ‘Σ_{a = 0}^{m*n - 1} f a = Σ_{i = 0}^{n - 1} (Σ_{a = m*i}^{m*(i + 1)} fa)‘. -/
lemma finset.sum_range_mul_eq_sum_Ico {m n : ℕ} (f : ℕ → S) :
  Σ a in finset.range (m * n), f a =
  Σ i in finset.range n, (Σ a in finset.Ico (m * i) (m * i.succ), f a)
```

The proof is as follows : since F is a multiple of f, we use the above lemma to unfold the RHS, and `eval_bernoulli_mul’` (discussed in the previous section) to expand the LHS. The hypothesis that $F > 0$ is needed so we can cancel the f^{m-1} terms from both sides. Moreover, this theorem is false for $F = 0$, since there is no dependency of the LHS on F.

An important property of these numbers is (from Proposition 7.2 of [?]) :

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

This is used heavily in the proof of the special values of the p-adic L-functions theorem (cite). We shall discuss its proof in Section 4 (link).

2.4 Filters and convergence

None of the mathematical proofs require filters on paper, however, we find that working with them makes formalising these proofs significantly less cumbersome. We shall not delve into the details of what a filter is, but instead explain how we use them to formalise convergence and limits.

Often, we have expressions of the form $\lim_{n\to\infty} f_n(x) = a$ for a sequence of functions $(f_n)_n$. This is represented in Lean as :

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

Here, `filter.at_top` (for the naturals) is the filter on ℕ generated by the collection of up-sets $\{b|a \le b\}$ for all $a \in \mathbb{N}$. There is a large library of lemmas regarding `filter.tendsto` in `mathlib`. Some important properties that we use frequently include :

```
  lemma filter.tendsto.const_mul {M : Type} {α : Type u_1} {β : Type}
  [topological_space α] [has_mul M α] [has_continuous_const_mul M α]
  {f : β → α} {l : filter β} {a : α} (hf : tendsto f l (nhds a)) (c : M) :
  tendsto (λ x, c * f x) l (nhds (c * a))
  lemma filter.tendsto.add {α : Type} {M : Type u_1} [topological_space M]
  [has_add M] [has_continuous_add M] {f g : α → M} {x : filter α} {a b : M},
  (hf : tendsto f x (nhds a)) (hg : tendsto g x (nhds b)) :
  tendsto (λ (x : α), f x + g x) x (nhds (a + b))
  lemma filter.tendsto_congr {α : Type} {β : Type u_1} {f₁ f₂ : α → β}
  {l₁ : filter α} {l₂ : filter β} (h : ∀ (x : α), f₁ x = f₂ x) :
  tendsto f₁ l₁ l₂ ↔ tendsto f₂ l₁ l₂
  lemma tendsto_zero_of_tendsto_const_smul_zero [algebra ℚ_[p] R] {f : ℕ → R}
  {l : filter ℕ} {c : ℚ_[p]} (hc : c ≠ 0) (hf : tendsto (λ y, c · f y) l (nhds 0)):
  tendsto (λ x, f x) l (nhds 0)
  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
```

The last lemma is particularly useful. it shows that sequences that are eventually equal (the same after finitely many elements) have the same limit. In particular, given `f₁ f₂ : ℕ → R`, `f₁ =ᶠ[at_top] f₂` is equivalent to saying that `∃ (a : ℕ), ∀ (b : ℕ), b ≥ a, f₁ b = f₂ b`.

We aim to use these lemmas as much as possible in order to avoid messy calculations with inequalities on norms. The only places they are not used are when it is necessary to deal with the inequalities, specifically when the non-Archimedean condition on the ring needs to be used.

3 Construction of the p-adic L-function

3.1 Profinite spaces

3.1.1 Density of locally constant functions

In Proposition 12.1 of [?], Washington defines the p-adic integral on $C(X, \mathbb{C}_p)$, the Banach space of continuous functions from a profinite space X to $\mathbb{C}_p$. This is an extension of a function defined on a dense subset of $C(X, \mathbb{C}_p)$, the locally constant functions from X to $\mathbb{C}_p$. In fact, for any compact Hausdorff totally disconnected space X and a commutative normed ring A, $LC(X, A)$ is a dense subset of $C(X, A)$.

The mathematical proof is the following : For $f \in C(X, A)$, we want to prove that, for any $\epsilon > 0$, we have $R = \bigcup_{x \in R} B(x, \epsilon)$. Since X is compact, one can find finitely many open sets $U_1, \dots, U_n$ such that $U_i = f^{-1}(B(x_i, \epsilon))$ for $x_1, \dots, x_n$ in R. One needs the fact that compact Hausdorff totally disconnected spaces have a clopen basis. We then find a finite set of disjoint clopen sets $C_1, \dots, C_m$ which form a basis, such that each C_j is contained in some U_i. Then, we pick elements $a_1, \dots, a_m$ in $C_1, \dots, C_m$, and construct the locally constant function $g(x) := \sum_{j=1}^{m} f(a_j)\chi_{C_j}(x)$, where χ_U is the characteristic (locally constant) function taking value 1 on every element of U and 0 otherwise. It then follows that $||f - g|| = sup_{x \in X}||f(x) - g(x)|| < \epsilon$, as required.

formalising this took about 500 lines of code. Let us show that locally compact Hausdorff totally disconnected spaces have a clopen basis :

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

The mathematical proof is : We want to show that for every $x \in H$ and open set U such that $x \in U$, there exists a clopen set C such that $x \in C$ and $C \subseteq U$. Since H is a locally compact space, we can find a compact set s such that $x \in$ (interior s) and $s \subseteq U$. The following lemma states that, every member of an open set in a compact Hausdorff totally disconnected space is contained in a clopen set contained in the open set. :

```
lemma compact_exists_clopen_in_open {x : α} {U : set α} (is_open : is_open U)
  (memU : x ∈ U) : ∃ (V : set α) (hV : is_clopen V), x ∈ V ∧ V ⊆ U
```

This implies that we can find a clopen set $V \subseteq$ (interior s) of s, with $x \in V$. Since V is closed in s(compact, hence closed), V is closed in H. Since $V \subseteq$ (interior s) is open in s, hence in (interior s), V is open in H, thus we are done.

This turned out to be harder to formalise than expected. Lean gives a subset V of s (`compact_space s ⟷ is_compact (s:set H)`) the type `V:set s`; however, Lean does not recognize V as a subset of H. As a result, I 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.

3.1.2 Clopen sets of the p-adic integers

As mentioned before, $\mathbb{Z}_p$ is a profinite space. Since it is the inverse limit of finite discrete topological spaces $\mathbb{Z}/p^n\mathbb{Z}$ for all n, it 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 now want to show that `clopen_basis` forms a topological basis and that every element is clopen :

```
theorem clopen_basis_clopen :
  topological_space.is_topological_basis (clopen_basis p) ∧
  ∀ 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} :

```
lemma preimage_to_zmod_pow_eq_ball (n : ℕ) (x : zmod (p^n)) :
(to_zmod_pow n) ⁻¹' {(x : zmod (p^n))} =
  metric.ball (x : ℤ_[p]) ((p : ℝ) ^ (1 - (n : ℤ)))
```

Notice that, in the RHS, we must specify `n:ℤ`. If that is not done, Lean interprets `1 - n:ℕ`. This is not the same as `1 - n:ℤ`, since subtraction is defined differently for the naturals.
Proving this lemma was fairly straightforward using the expansion of a p-adic integer, that is, every $a \in \mathbb{Z}_p$ can be written as $\sum_{n=0}^{\infty} a_n p^n$, with $a_n \in \mathbb{Z}/p^n\mathbb{Z}$. Approximating a by $\sum_{n=0}^{m} a_n p^n$ is done by the function `appr:ℤ_p → ℕ → ℕ`. Note that `appr` returns a natural number, with a_n being the smallest natural number in the $\mathbb{Z}/p^n\mathbb{Z}$ equivalence class. This turned out to be very useful, along with the following lemmas :

```
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])
```

In the latter lemma, the LHS is a coercion of `appr x n`, which has type ℕ, to `ℤ_p`. The RHS 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 $proj_n$ descried above. We call this set `clopen_basis' p d`. For the sake of simplicity, we shall discuss the case $d = 1$ in this article.

3.2 p-adic 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 $i \geq j$. We use G to denote an abelian group, A for a commutative normed ring, R for a commutative complete normed ring having a coercion from $\mathbb{Z}_p$, 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 $gcd(d, p) = 1$.

We begin by defining distributions on profinite sets. In Section 12.1 of [?], Washington gives 3 equivalent definitions of a distribution :

1. A system of maps $\phi_i : X_i \to G$ such that $\forall i \geq j$,
$$\phi_j(x) = \sum_{\pi_{ij}(y)=x} \phi_i(y)$$
2. A G-linear function $\phi : LC(X, G) \to G$.
3. A finitely additive function from the compact open sets of X to G.

Since switching between definitions is cumbersome, and(at that point of time), `mathlib` had no notion of thinking about profinite sets as inverse limits of finite sets, we chose to work with the second definition. However, this is already a Type, hence there is no need to redefine it.

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 simply used the existing lemmas along with the equivalence `bounded_continuous_function.equiv_bounded_of_compa`

While showing that, $\forall f \in C(X, A)$, $||f|| = 0 \implies f = 0$, it suffices to show $||f|| \leq 0 \implies \forall x \in X, ||f(x)|| \leq 0$. We then use the following lemma, which requires a nonempty assumption on X :

```
theorem cSup_le_iff {α : Type*} [conditionally_complete_lattice α]
{s : set α} {a : α} (hb : bdd_above s) (ne : nonempty s) :
Sup s ≤ a ↔ (∀b ∈ s, b ≤ a)
```

The case that X is empty is separately (and trivially) solved.

We can now define p-adic measures. Measures are bounded distributions. Note that the p-adic measures are not to be confused with measures arising from measure theory. The key difference lies in the fact that the clopen sets of a profinite space do not, in general, form a σ-algebra.

```
def measures [nonempty X] :=
{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 boundedness of the distribution is needed to make the measure continuous :

```
lemma integral_cont (φ : measures X A) : continuous ⇑φ
```

The proof is straightforward : for $b \in LC(X, A)$, given $\epsilon > 0$, there exists a $\delta > 0$ such that for all $a \in LC(X, A)$ with $||b - a|| < \delta$, $||\phi(a) - \phi(b)|| < \epsilon$. Since ϕ is a measure, it suffices to prove that $K * ||inclusion(a - b)|| < \epsilon$. Choosing $\delta = \epsilon/K$ gives the desired result.

The Bernoulli measure is an essential p-adic 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 $x_n \in (\mathbb{Z}/dp^{n+1}\mathbb{Z})^\times$, the (first) Bernoulli measure is defined by

$$E_{c,n}(x_n) = B_1\left(\left\{\frac{x_n}{dp^{n+1}}\right\}\right) - cB_1\left(\left\{\frac{c^{-1}x_n}{dp^{n+1}}\right\}\right)$$

The system $(E_{c,n})_{n \in \mathbb{N}}$ forms a distribution according to the first definition given above. We want to get an equivalent reformulation in terms of the second definition. We know that, since X is compact, every locally constant function can be written in terms of a finite sum of a characteristic function of a basis element multiplied by a constant. Since X is profinite, from the previous section, we know that there exists a clopen basis of the form `set.preimage (padic_int.to_zmod_pow n) a` for $a \in \mathbb{Z}/dp^n\mathbb{Z}$. For the sake of simplicity, let us assume $d = 1$. Thus, for a clopen set $U_{a,n}$:= `set.preimage (padic_int.to_zmod_pow n) a`, we define

$$E_c(\chi_{U_{a,n}}) = E_{c,n}(a)$$

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

```
def E_c (hc : gcd d p = 1) :=
  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
```

3.3 The Bernoulli measure

Throughout this section, we assume that R is a normed commutative ring which is a $\mathbb{Q}_p$-algebra, $\mathbb{Q}$-algebra, and has a nonarchimedean norm, that is, for any finite set of elements $(r_i)_{i=1}^n$ of R, $\sum_{i=1}^n \parallel r_i \parallel \leq \sup \parallel r_i \parallel$.

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 (is_clopen_clopen_from p d n a)) =
   (algebra_map ℚ R) (E_c p d hc n a) }
```

and to show that it is nonempty. However, this is quite a roundabout way, since one then has to use `classical.some` to extract the Bernoulli measure. We use an elegant way to tackle the problem.

First, we define `eventually_constant_seq` to be the type of sequences satifying :

```
/-- 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
```

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

$$g(n) = \sum_{a \in \mathbb{Z}/(d*p^n)\mathbb{Z}} f(a) E_{c,n}(a)$$

for all natural numbers n. We shall look into the proof of g being eventually constant later. We now define the Bernoulli distribution to be the limit of this sequence g.

The proof of this distribution being closed under addition and scalar multiplication follows easily from these definitions and lemmas :

```
/-- The smallest number ‘m‘ for the sequence ‘a‘ such that ‘a n = a (n + 1)‘ for all ‘n ≥ m‘. -/
noncomputable def sequence_limit_index' {α : Type*} (a : @eventually_constant_seq α) : ℕ :=
  Inf { n | ∀ m, n ≤ m → a.to_seq m.succ = a.to_seq m }

/-- The limit of an ‘eventually_constant_seq‘. -/
noncomputable def sequence_limit {α : Type*} (a : @eventually_constant_seq α) :=
  a.to_seq (sequence_limit_index' a)

lemma sequence_limit_eq {α : Type*} (a : @eventually_constant_seq α) (m : ℕ)
  (hm : sequence_limit_index' a ≤ m) : sequence_limit a = a.to_seq m
```

Now, given a locally constant function `f : locally_constant (units (zmod d) × units ℤ_[p]) R`, the `bernoulli_meas` is given by :

```
  bernoulli_distribution 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. We shall skip the proof that this function is locally constant, because it is long and cumbersome. For every open set s, one must look at the case when 0 is contained in s separately. We (prove and) use the fact that `coe : units (zmod d) × units ℤ_[p] → (zmod d) × ℤ_[p]` is an open embedding heavily.

We must now prove that `bernoulli_measure` is indeed a measure, that is, it is bounded. The bound we choose is $1 + \parallel c \parallel + \parallel \frac{c-1}{2} \parallel$. The proof is as follows : let BD denote the Bernoulli distribution, and ϕ denote `loc_const_ind_fn`. We want to show that :

$$\parallel BD(\phi(f)) \parallel \leq K \parallel inclusion(f) \parallel$$

where K is the constant given above. We know that, one can find an n such that

$$\phi(f) = \sum_{a \in \mathbb{Z}/d\mathbb{Z} \times \mathbb{Z}/p^n\mathbb{Z}} \phi(f)(a)\dot{} \chi_{n,a}$$

Since BD is a linear map, it suffices to show that $\parallel BD(\phi(f)(a)\chi_{n,a}) \parallel \leq K \parallel inclusionf \parallel$ for some a (the supremum is achieved since the set is finite). If a is not a unit, $\phi(f)(a) = 0$; by using the linearity of BD, we are done. In the case that a is a unit, it suffices to prove that $\parallel BD(\chi_{n,a}) \parallel \leq K$, and $\parallel \phi(f)(a) \parallel \leq \parallel inclusionf \parallel$. Both of these are easily verified, so we are done.

The implementation is similar, and is heavily dependent on the following lemma :

```
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 (is_clopen_clopen_from p d n a)
```

The machinery used in this proof is similar to the one used to prove that g is eventually constant. Let us have a look at the latter first.

We must first take a look at 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 that $\forall x, y \in X$, $x \sim_A y \implies x \sim_B y$.

Any locally constant function induces a discrete quotient, since each of its fibers is clopen :

```
def discrete_quotient : discrete_quotient X :=
{ rel := λ a b, f b = f a,
  equiv := ⟨by tauto, by tauto, λ a b c h1 h2, by rw [h2, h1]⟩,
  clopen := λ x, f.is_locally_constant.is_clopen_fiber _ }
```

We now define a function :

```
def F : 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, _, _⟩
```

In other words, for $a = (a_1, a_2)$ and $b = (b_1, b_2)$ in $\mathbb{Z}/d\mathbb{Z} \times \mathbb{Z}_p$, $F(n)$ represents the relation

$$a \sim b \iff a_2(\text{mod } p^n) = b_2(\text{mod } p^n)a_1 = b_1$$

Then, given a locally constant function f on $\mathbb{Z}/d\mathbb{Z} \times \mathbb{Z}_p$, we have :

```
lemma factor_F (hd : d.gcd p = 1) (f : locally_constant (zmod d × ℤ_[p]) R) :
  ∃ N : ℕ, F N ≤ discrete_quotient f
```

`factor_F` states that, for N large enough, the fibers of f mod p^N are contained in the basic clopen sets of p^N. Here is the proof of `factor_F` : Since X is compact, there exists a finite set t in R such that $X \subseteq \cup_{i \in t} f^{-1}(i)$. Given an open set U of $\mathbb{Z}/d\mathbb{Z} \times \mathbb{Z}_p$, we now define the `bound_set` of U to be $\{n \in \mathbb{N} \mid \forall a \in U, U_{n,(a\ :\ \text{zmod}\ p^n)} \subseteq U\}$. The `bound` of U is then defined to be the infimum of the `bound_set U`. Let n be the supremum of `bound` of $f^{-1}(i)$ for all $i \in t$. This is the required `N`.

The proofs of both g being eventually constant now follows. We want to show :

$$\exists N, \forall m \geq N, \sum_{a \in \mathbb{Z}/d*p^{m+1}\mathbb{Z}} f(a)E_{c,m+1}(a) = \sum_{a \in \mathbb{Z}/d*p^m\mathbb{Z}} f(a)E_{c,m}(a)$$

The N we choose is `classical.some (factor_F f) + 1`. We also define the following :

```
/-- Given `a ∈ zmod (d * p^n)`, and `n < m`, the set of all `b ∈ zmod (d * p^m)` such that `b = a
    mod (d * p^n)`. -/
def equi_class (n m : ℕ) (h : n ≤ m) (a : zmod (d * p^n)) :=
 {b : zmod (d * p^m) | (b : zmod (d * p^n)) = a}
```

Then, we have the following lemma :

```
lemma succ_eq_bUnion_equi_class : 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 $b \in \mathbb{Z}/dp^m\mathbb{Z}$. Notice that we use `zmod'` instead of `zmod`, since that has type `finset`, that is, the property of `zmod` being finite is encoded in it. This is needed since we are working with finite sums, hence Lean demands elements of type `finset` instead of `set`.

The proof is now complete with the following lemma :

```
lemma E_c_sum_equi_class (x : zmod (d * p^m)) :
  Σ (y : zmod (d * p ^ (m + 1))) in (λ a : zmod (d * p ^ m),
  set.to_finset ((equi_class m (m + 1)) a)) x, (E_c (m + 1) y) = E_c m x
```

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

Finally, we turn to the proof of `loc_const_eq_sum_char_fn`. We want to show that, for any locally constant function f from $\mathbb{Z}/d\mathbb{Z} \times \mathbb{Z}_p$ to R,

$$\exists n, f = \sum_{a \in \mathbb{Z}/dp^n\mathbb{Z}} f(a)\chi_{n,a}$$

We choose n to be `classical.some factor_F f`. The proof now follows easily, since each element belongs to exactly one clopen set of level n. One must go between elements of $\mathbb{Z}/d\mathbb{Z} \times \mathbb{Z}_p$ and $\mathbb{Z}/dp^n\mathbb{Z}$. This requires use of multiple coercions, which makes the proof lengthy.

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.

3.4 p-adic integrals and the p-adic L-function

3.4.1 p-adic Integrals

The last piece in the puzzle is the p-adic 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 [?], 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 *inclusion* $: LC(X, R) \to C(X, R)$ is `dense_inducing`, that is, it has dense range and the topology on $LC(X, R)$ is the one induced by `inclusion` from the topology on $C(X, R)$.

For the linearity of the map, we use `dense_range.induction_on₂`, which states that, given a map $e : \alpha \to \beta$ which has dense range, and a property p (taking two elements as input), in order for any two elements of β to satisfy p, it is sufficient to show that any two elements in the range of e satisfy p; and, the set of elements satifying p is closed with respect to the topology of β :

```
lemma dense_range.induction_on₂ {α β : Type*} [topological_space β] {e : α → β}
{p : β → β → Prop} (he : dense_range e) (hp : is_closed {q:β×β | p q.1 q.2})
(h : ∀a₁ a₂, p (e a₁) (e a₂)) (b₁ b₂ : β) : p b₁ b₂
```

In particular, for every continuous function f, we can take the property p to be preservation of addition (and scalar multiplication respectively) under $\mu(f)$, and `inclusion` to be the map with dense range. The first condition is satisfied due to the following lemma :

```
lemma is_closed_eq [t2_space α] {f g : β → α} (hf : continuous f)
(hg : continuous g) : is_closed {x:β | f x = g x}
```

The second condition follows easily due to the linearity of the measure and the map `inclusion`.

The continuity of the extension of the integral follows from the fact that every measure μ is uniformly continuous. Uniform continuity is a product of the boundedness of the measure : We want to show that, for any measure ϕ, given an $\epsilon > 0$, $\exists \delta > 0$ such that for any $a, b \in LC(X, R)$, with $||a - b|| < \delta$, $||\phi(a) - \phi(b)|| < \epsilon$. Assuming that the constant in the definition of ϕ is K, it is clear that ϵ/K is the required δ. This is the proof of the following lemma :

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

3.4.2 Construction

There are several possible definitions for the p-adic L-functions (fixing an embedding of $\bar{\mathbb{Q}}$ into $\mathbb{C}_p$), including :

1. (Theorem 5.11, [?]) The p-adic meromorphic function $L_p(s, \chi)$ on $\{s \in \mathbb{C}_p || s| < p\}$ 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 $n \geq 1$.
2. (Proof of Theorem 5.11, [?]) $L_p(s, \chi) = \sum_{a=1, p\nmid a}^{F} \chi(a)H_p(s, a, F)$, where $H_p(s, a, F)$ is a meromorphic function satisfying $H_p(1-n, a, F) = -\frac{F^{n-1}\omega^{-n}(a)}{n}B_n\left(\frac{a}{F}\right)$ for all natural numbers n.
3. (Theorem 12.2, [?]) 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 $gcd(c, dp) = 1$:
$$(1 - \chi(c) < c >^{s+1})L_p(-s, \chi) = \int_{(\mathbb{Z}/d\mathbb{Z})^\times \times \mathbb{Z}_p^\times} \chi\omega^{-1}(a) < a >^s \, dE_c$$
where $< a >= \omega^{-1}(a)a$, and $b^s = exp(log_p(b))$ (the exponential and logarithm are defined in terms of power series expansions).

It is beyond the scope of this article to explain all the notation in the points above. I chose to define the p-adic L-function as a reformulation of (3). This is because it is the most optimal definition that helps with stating the Iwasawa Main Conjecture.

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. We replace $< a >^s$ with `w:weight_space A`. The advantage is that our p-adic L-function can now be defined over a more general space.

Given a primitive Dirichlet character χ of character dp^m with $gcd(d, p) = 1$ and $m \geq 0$, we now define the p-adic L-function to be :
$$L_p(w, \chi) := \int_{(\mathbb{Z}/d\mathbb{Z})^\times \times \mathbb{Z}_p^\times} \chi\omega^{p-2}(a)w \, dE_c$$

```
  def p_adic_L_function (hc : gcd c p = 1) :=
  integral (units (zmod d) × units ℤ_[p]) R _
  (bernoulli_measure_of_measure p d R hc)
⟨(λ (a : (units (zmod d) × units ℤ_[p])), ((pri_dir_char_extend p d R) a) *
  (inj (üTeichmller_character p a.snd))^(p - 2) * (w.to_fun a : R)),
    cont_paLf p d R inj w ⟩
```

Note that we have absorbed the constant term given in (3). 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. However, this factor appears as it is in the Iwasawa Main Conjecture, and can be easily removed if one assumes R to have division.

Lean (implicitly) interprets the placeholder _ as a proof that $(\mathbb{Z}/d\mathbb{Z})^\times \times \mathbb{Z}_p^\times$ = `units (zmod d) × units ℤ_[p]` is nonempty. Note that `pri_dir_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.

By construction, the last term given as input to `integral` has type `C((units (zmod d) × units ℤ_[p]),R)`. This is the term in angular brackets `⟨_, _⟩`, the former is the integrand, and the latter is a proof that the integrand is continuous :

```
lemma cont_paLf : continuous (λ (a : (units (zmod d) × units ℤ_[p])),
((pri_dir_char_extend p d R) a) * (inj (üTeichmller_character p (a.snd)))^(p - 2)
* (w.to_fun a : R))
```

What remains to be proved is that f is invariant with respect to c. Once these are proved, we can formalise properties of p-adic L-functions. One of the most important properties is, for an integer n with $n \geq 1$,

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

Recall that the function corresponding to $1-n$ is $< a >^{1-n}$. One must show that $< a >^{1-n}$ is an element of the weight space. formalising the first definition would have made showing this result a lot tougher, since showing the analytic continuation of these functions is nontrivial, even on paper.

4 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 $n > 1$,

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

For this section, we assume that R is a normed commutative $\mathbb{Q}_p$-algebra and $\mathbb{Q}$-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 the condition `is_scalar_tower ℚ ℚ_[p] R` (see [?]). It is also non-archimedean, which are encapsulated by the following hypotheses:

```
na : ∀ (n : ℕ) (f : ℕ → R),
  ‖Σ (i : ℕ) in finset.range n, f ‖i ≤ ⨆ (i : zmod n), ‖f i.‖val
na' : ∀ (n : ℕ) (f : (zmod n)ˣ → R),
  ‖Σ i : (zmod n)ˣ, f ‖i ≤ ⨆ (i : (zmod n)ˣ), ‖f ‖i
```

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.

4.1 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 mentioned in the previous section, 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 Dirichlet character ψ of level k_1k_2, one can find primitive Dirichlet characters ψ_1 and ψ_2 of levels k_1 and k_2 respectively such that $\psi = \psi_1\psi_2$:

```
lemma dirichlet_character.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 Dirichlet characters χ_1 and χ_2 with conductors z_1 and z_2 such that $z_1|d$ and $z_2|p^m$ 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 $\chi\omega^{-k}(y) = 0$.

4.2 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 very calculation intensive. We shall instead highlight the key concepts that are used.

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 [?].

Using the fact that the space of locally constant functions 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 :

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

$$= \lim_{j\to\infty} \left(\sum_{a\in(\mathbb{Z}/dp^j\mathbb{Z})^\times} \chi\omega^{-n}a^{n-1}\left\{\frac{a}{dp^j}\right\} - \sum_{a\in(\mathbb{Z}/dp^j\mathbb{Z})^\times} \chi\omega^{-n}a^{n-1}\left(c\left\{\frac{c^{-1}a}{dp^j}\right\}\right) + \left(\frac{c-1}{2}\right) \sum_{a\in(\mathbb{Z}/dp^j\mathbb{Z})^\times} \chi\omega^{-n}a^{n-1}\right)$$

Going from the first equation to the second took about 600 lines of code, which can be found in `try.lean` (add link). 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. Some other difficulties include several different ways to write 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 in `UVW.lean`. This is dependent on the following lemma :

$$\lim_{j\to\infty} \frac{1}{dp^j} \sum_{i\in\mathbb{Z}/dp^j\mathbb{Z}} \chi\omega^{-n}(i)i^n = B_{n,\chi\omega^{-n}}$$

for $n > 1$. There were two options for the `finset` used in `finset.sum` : `finset.range` or `zmod (d * p^j)`. We chose the former since it was easier to deal with, as some required lemmas regarding it had been previously formalised. This lemma is formulated in Lean as :

```
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
(nhds (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 [?]. It is very calculation intensive, since there are multiple coercions to be dealt with. Also, terms needs to be moved around to be multiplied, deleted and cancelled.

Unfortunately, there is no tactic that takes care of these elementary but lengthy calculations.

This is similar to what we need to compute the first sum in (4.2) :

```
tendsto (λ (j : ℕ), Σ (x : (zmod (d * p ^ j))ˣ),
((asso_dirichlet_character (χ.mul (teichmuller_character_mod_p' p R ^ n))) ↑x *
↑(↑x.val) ^ (n - 1)) · (algebra_map ℚ R) (int.fract (↑x / (↑d * ↑p ^ j))))
at_top
(nhds ((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))
```

In order to convert this sum into a sum over `finset.range`, we show that :

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

This is equivalent to saying :

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

The condition that d divides the conductor is then used to show that the associated Dirichlet character is 0 everywhere except $(\mathbb{Z}/dp^k\mathbb{Z})^\times$.

Evaluating the middle sum 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.

Finally, the last sum is 0. This follows by substituting a in the summand with $-a$. This is where one uses that χ is even.

There were two ways to do calculations with respect to inequalities on the norm : working with lemmas regarding `filter.tendsto`, or using the following 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 (nhds a) ↔ ∀ (ε : ℝ), ε > 0 →
(∃ (N : β), ∀ (n : β), n ≥ N → dist (u n) a < ε)
```

We would like to point out that working with `filter.tendsto` instead of `metric.tendsto_at_top` really simplified calculations. This is because, often, the ε we would choose would be complicated, making our calculations more complicated. As an example, suppose we want to prove :

```
(h : filter.tendsto (λ x, f x) at_top (nhds 0)) → filter.tendsto (λ x, c * f x) at_top (nhds 0)
```

This is a one-line proof using `filter.tendsto_const_mul`. However, if done using `metric.tendsto_at_top`, given $\varepsilon > 0$, we must pick an N such that $\|fx\| < \varepsilon/c$, and use N to complete the proof. Most of such issues can be dealt with using the lemma `filter.tendsto_congr'`.

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 needs to be used. Hence, this is a good indicator of where the non-Archimedean condition is needed.

5 Conclusion

5.1 Analysis

We list some of the observations that arose while working on this paper.
Throughout this paper, `abbreviation` has played an important part. They help to reduce the number of `def` for a prticular type. A technical difficulty is that one cannot use tactic `rw` to unfold the underlying definition. One must either use `delta`, which often slows compilation time, or make a lemma unfolding the `abbreviation`.

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; the only tactics one can use inside `conv` mode are `rw`, `apply_congr` (similar to `apply`), `simp` and `norm_cast`. 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 problem that was recurring 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. This is problematic because 7 is already a large number of hypotheses. 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`. Moreover, one cannot get out of `conv` mode unless all these goals are solved. This is difficult since `apply_instance` does not work in `conv` mode. The other alternative is to explicitly provide terms using `@`, however this leads to very large expressions.

Working with Dirichlet characters was challenging. This is because, given $a, b \in \mathbb{N}$ such that $a = b$, Lean does not identify `dirichlet_character R a = dirichlet_character R b`. Tactics such as `subst` also fail. A workaround was putting `h:a = b` as a local hypothesis and then using `congr'`. We then end up with the goal `dirichlet_character R a == dirichlet_character R b`. The API for `heq` suggests that this should be avoided as much as possible. Another workaround was to define and use `dirichlet_character R a` $\simeq$ `dirichlet_character R b`. However, this would slow down the compilation a lot, sometimes also leading to deterministic timeouts.

5.2 Statistics

When initially completed, this project consisted of around 18,000 lines of code. A major refactor was then done, keeping in mind several of the points mentioned above, specifically using properties of filters instead of metric space calculations wherever possible. All properties regarding particular types (such as Dirichlet characters) were also compiled together in separate files wherever possible. Keeping in mind the spirit of `mathlib`, these properties have been made in the greatest generality as much as possible. After the refactor, there are about 15000 lines of code, put into 10 files (check!).

While most properties regarding Bernoulli numbers and polynomials 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.

5.3 Related and future work

There are several projects that require Dirichlet characters and properties of the p-adic integers. These include the project on the formalisation of Fermat's last theorem (link). There is also an effort by Prof David Loeffler which involves formalisation of the classical Dirichlet L-function, that is somewhat dependent on this work.

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 formalise more properties of Bernoulli numbers, that are a fundamental component of number theory.

6 Typesetting instructions – Summary

LIPIcs is a series of open access high-quality conference proceedings across all fields in informatics established in cooperation with Schloss Dagstuhl. In order to do justice to the high scientific quality of the conferences that publish their proceedings in the LIPIcs series, which is ensured by the thorough review process of the respective events, we believe that LIPIcs proceedings must have an attractive and consistent layout matching the standard of the series. Moreover, the quality of the metadata, the typesetting and the layout must also meet the requirements of other external parties such as indexing service, DOI registry, funding agencies, among others. The guidelines contained in this document serve as the baseline for the authors, editors, and the publisher to create documents that meet as many different requirements as possible.

Please comply with the following instructions when preparing your article for a LIPIcs proceedings volume.

Minimum requirements

- Use pdflatex and an up-to-date LaTeX system.
- Use further LaTeX packages and custom made macros carefully and only if required.
- Use the provided sectioning macros: `\section`, `\subsection`, `\subsubsection`, `\paragraph`, `\paragraph*`, and `\subparagraph*`.
- Provide suitable graphics of at least 300dpi (preferably in PDF format).
- Use BibTeX and keep the standard style (`plainurl`) for the bibliography.
- Please try to keep the warnings log as small as possible. Avoid overfull `\hboxes` and any kind of warnings/errors with the referenced BibTeX entries.
- Use a spellchecker to correct typos.

Mandatory metadata macros

Please set the values of the metadata macros carefully since the information parsed from these macros will be passed to publication servers, catalogues and search engines. Avoid placing macros inside the metadata macros. The following metadata macros/environments are mandatory:

- `\title` and, in case of long titles, `\titlerunning`.
- `\author`, one for each author, even if two or more authors have the same affiliation.
- `\authorrunning` and `\Copyright` (concatenated author names)
  The `\author` macros and the `\Copyright` macro should contain full author names (especially with regard to the first name), while `\authorrunning` should contain abbreviated first names.
- `\ccsdesc` (ACM classification, see https://www.acm.org/publications/class-2012).
- `\keywords` (a comma-separated list of keywords).
- `\relatedversion` (if there is a related version, typically the "full version"); please make sure to provide a persistent URL, e. g., at arXiv.
- `\begin{abstract}...\end{abstract}` .

Please do not ...

Generally speaking, please do not override the `lipics-v2021`-style defaults. To be more specific, a short checklist also used by Dagstuhl Publishing during the final typesetting is given below. In case of **non-compliance** with these rules Dagstuhl Publishing will remove the corresponding parts of LaTeX code and **replace it with the `lipics-v2021` defaults**. In serious cases, we may reject the LaTeX-source and expect the corresponding author to revise the relevant parts.

- Do not use a different main font. (For example, the `times` package is forbidden.)
- Do not alter the spacing of the `lipics-v2021.cls` style file.
- Do not use `enumitem` and `paralist`. (The `enumerate` package is preloaded, so you can use `\begin{enumerate}[(a)]` or the like.)
- Do not use "self-made" sectioning commands (e. g., `\noindent{\bf My Paragraph}`).
- Do not hide large text blocks using comments or `\iffalse` … `\fi` constructions.
- Do not use conditional structures to include/exclude content. Instead, please provide only the content that should be published – in one file – and nothing else.
- Do not wrap figures and tables with text. In particular, the package `wrapfig` is not supported.
- Do not change the bibliography style. In particular, do not use author-year citations. (The `natbib` package is not supported.)

This is only a summary containing the most relevant details. Please read the complete document "LIPIcs: Instructions for Authors and the `lipics-v2021` Class" for all details and don't hesitate to contact Dagstuhl Publishing (mailto:publishing@dagstuhl.de) in case of questions or comments: `http://drops.dagstuhl.de/styles/lipics-v2021/lipics-v2021-authors/lipics-v2021-authors-guidelines.pdf`

7 Lorem ipsum dolor sit amet

Lorem ipsum dolor sit amet, consectetur adipiscing elit [?]. Praesent convallis orci arcu, eu mollis dolor. Aliquam eleifend suscipit lacinia. Maecenas quam mi, porta ut lacinia sed, convallis ac dui. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse potenti. Donec eget odio et magna ullamcorper vehicula ut vitae libero. Maecenas lectus nulla, auctor nec varius ac, ultricies et turpis. Pellentesque id ante erat. In hac habitasse platea dictumst. Curabitur a scelerisque odio. Pellentesque elit risus, posuere quis elementum at, pellentesque ut diam. Quisque aliquam libero id mi imperdiet quis convallis turpis eleifend.

▶ **Lemma 1** (Lorem ipsum). *Vestibulum sodales dolor et dui cursus iaculis. Nullam ullamcorper purus vel turpis lobortis eu tempus lorem semper. Proin facilisis gravida rutrum. Etiam sed sollicitudin lorem. Proin pellentesque risus at elit hendrerit pharetra. Integer at turpis varius libero rhoncus fermentum vitae vitae metus.*

Proof. Cras purus lorem, pulvinar et fermentum sagittis, suscipit quis magna.

Just some paragraph within the proof. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

▷ Claim 2. content...

Proof. content...

1. abc abc abc ◁

◀

▶ **Corollary 3** (Curabitur pulvinar, [?]). *Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.*

▶ **Proposition 4.** *This is a proposition*

Proposition 4 and Proposition 4 …

Listing 1 Useless code.

```
for i:=maxint to 0 do
begin
    j:=square(root(i));
end;
```

7.1 Curabitur dictum felis id sapien

Curabitur dictum Corollary 3 felis id sapien Corollary 3 mollis ut venenatis tortor feugiat. Curabitur sed velit diam. Integer aliquam, nunc ac egestas lacinia, nibh est vehicula nibh, ac auctor velit tellus non arcu. Vestibulum lacinia ipsum vitae nisi ultrices eget gravida turpis laoreet. Duis rutrum dapibus ornare. Nulla vehicula vulputate iaculis. Proin a consequat neque. Donec ut rutrum urna. Morbi scelerisque turpis sed elit sagittis eu scelerisque quam condimentum. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Aenean nec faucibus leo. Cras ut nisl odio, non tincidunt lorem. Integer purus ligula, venenatis et convallis lacinia, scelerisque at erat. Fusce risus libero, convallis at fermentum in, dignissim sed sem. Ut dapibus orci vitae nisl viverra nec adipiscing tortor condimentum [?]. Donec non suscipit lorem. Nam sit amet enim vitae nisl accumsan pretium.

7.2 Proin ac fermentum augue

Proin ac fermentum augue. Nullam bibendum enim sollicitudin tellus egestas lacinia euismod orci mollis. Nulla facilisi. Vivamus volutpat venenatis sapien, vitae feugiat arcu fringilla ac. Mauris sapien tortor, sagittis eget auctor at, vulputate pharetra magna. Sed congue, dui nec vulputate convallis, sem nunc adipiscing dui, vel venenatis mauris sem in dui. Praesent a pretium quam. Mauris non mauris sit amet eros rutrum aliquam id ut sapien. Nulla aliquet fringilla sagittis. Pellentesque eu metus posuere nunc tincidunt dignissim in tempor dolor. Nulla cursus aliquet enim. Cras sapien risus, accumsan eu cursus ut, commodo vel velit. Praesent aliquet consectetur ligula, vitae iaculis ligula interdum vel. Integer faucibus faucibus felis.

- Ut vitae diam augue.
- Integer lacus ante, pellentesque sed sollicitudin et, pulvinar adipiscing sem.
- Maecenas facilisis, leo quis tincidunt egestas, magna ipsum condimentum orci, vitae facilisis nibh turpis et elit.

▶ Remark 5. content...

8 Pellentesque quis tortor

Nec urna malesuada sollicitudin. Nulla facilisi. Vivamus aliquam tempus ligula eget ornare. Praesent eget magna ut turpis mattis cursus. Aliquam vel condimentum orci. Nunc congue, libero in gravida convallis [?], orci nibh sodales quam, id egestas felis mi nec nisi. Suspendisse tincidunt, est ac vestibulum posuere, justo odio bibendum urna, rutrum bibendum dolor sem nec tellus.

▶ **Lemma 6** (Quisque blandit tempus nunc)**.** *Sed interdum nisl pretium non. Mauris sodales consequat risus vel consectetur. Aliquam erat volutpat. Nunc sed sapien ligula. Proin faucibus sapien luctus nisl feugiat convallis faucibus elit cursus. Nunc vestibulum nunc ac massa pretium pharetra. Nulla facilisis turpis id augue venenatis blandit. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.*

Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.

9 Morbi eros magna

Morbi eros magna, vestibulum non posuere non, porta eu quam. Maecenas vitae orci risus, eget imperdiet mauris. Donec massa mauris, pellentesque vel lobortis eu, molestie ac turpis. Sed condimentum convallis dolor, a dignissim est ultrices eu. Donec consectetur volutpat eros, et ornare dui ultricies id. Vivamus eu augue eget dolor euismod ultrices et sit amet nisi. Vivamus malesuada leo ac leo ullamcorper tempor. Donec justo mi, tempor vitae aliquet non, faucibus eu lacus. Donec dictum gravida neque, non porta turpis imperdiet eget. Curabitur quis euismod ligula.

A Styles of lists, enumerations, and descriptions

List of different predefined enumeration styles:

- `\begin{itemize}...\end{itemize}`
- ...
- ...

1. `\begin{enumerate}...\end{enumerate}`
2. ...
3. ...

(a) `\begin{alphaenumerate}...\end{alphaenumerate}`
(b) ...
(c) ...

(i) `\begin{romanenumerate}...\end{romanenumerate}`
(ii) ...
(iii) ...

(1) `\begin{bracketenumerate}...\end{bracketenumerate}`
(2) ...
(3) ...

Description 1 `\begin{description} \item[Description 1]  ...\end{description}`
Description 2 Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.
Description 3 ...

Proposition 10 and Proposition 10 ...

B Theorem-like environments

List of different predefined enumeration styles:

▶ **Theorem 7.** *Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.*

▶ **Lemma 8.** *Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.*

▶ **Corollary 9.** *Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.*

▶ **Proposition 10.** *Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.*

▶ **Conjecture 11.** *Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.*

▶ **Observation 12.** *Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.*

▶ **Exercise 13.** *Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.*

▶ **Definition 14.** *Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.*

▶ **Example 15.** Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.

▶ Note 16. Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.

▶ Note. Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.

▶ Remark 17. Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.

▶ Remark. Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.

▷ Claim 18. Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.

▷ Claim. Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque.

Proof. Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque. ◀

Proof. Fusce eu leo nisi. Cras eget orci neque, eleifend dapibus felis. Duis et leo dui. Nam vulputate, velit et laoreet porttitor, quam arcu facilisis dui, sed malesuada risus massa sit amet neque. ◁