Annals of Formalized Mathematics
Volume **2** (2026), 55–81
DOI: https://doi.org/10.46298/afm.15738

Formalising the Bruhat–Tits Tree

Judith Ludwig and Christian Merten

Abstract. In this article we describe the formalisation of the Bruhat–Tits tree—an important tool in modern number theory—in the Lean Theorem Prover. Motivated by the goal of connecting to ongoing research, we apply our formalisation to verify a result about harmonic cochains on the tree.

MSC 2020: 68V20, 20E08, 20G25
Keywords: Lean, Bruhat–Tits tree, Cartan decomposition

Contact information:
J. Ludwig: Heidelberg University. ludwig@mathi.uni-heidelberg.de
C. Merten: Utrecht University. c.j.merten@uu.nl

Received: May 23, 2025
Final form: February 18, 2026
Accepted: April 14, 2026

1 Introduction

The Bruhat–Tits tree is a combinatorial object that serves as a powerful tool in number theory and arithmetic geometry. In the simplest setting of the field $\mathbb{Q}_p$ of p-adic numbers, the Bruhat–Tits tree is constructed from lattices in $\mathbb{Q}_p^2$ and is a regular infinite tree, where each vertex has $p+1$ neighbours. Its geometry is characterised by its simplicity. However its connections to the structure theory and the representation theory of the group $\mathrm{GL}_2(\mathbb{Q}_p)$ are deep and intricate.

In this article we report on the formalisation of this useful mathematical concept in the Lean Theorem Prover. Our formalisation adds to the ongoing effort of formalising graduate level concepts in pure mathematics and more specifically in number theory.

For the formalisation of the Bruhat–Tits tree, we pass to a more general setting and replace $\mathbb{Q}_p$ by any discrete valuation field K. Before we describe our project in more detail, let us say a few more words on the applications of the Bruhat–Tits tree. For that it is useful to furthermore assume that K is complete and has finite residue field. In this setting, the Bruhat–Tits tree is a powerful tool for the study of subgroups of $\mathrm{GL}_2(K)$ (as well as the closely related groups $\mathrm{SL}_2(K)$ and $\mathrm{PGL}_2(K)$) and their homology, see e.g. [Ser03, Chapter II]. Prominently, as explained in op. cit. it can be used to show Ihara's theorem, that every torsion free subgroup of $\mathrm{SL}_2(K)$ is free. The Bruhat–Tits tree is important in other parts of number theory, arithmetic geometry and representation theory, e.g., via its connection to Drinfeld's upper half plane Ω (see [DT08]) or through the relation to classical as well as function field modular forms via harmonic cochains (see e.g. [Tei91; Tei90]). The Bruhat–Tits tree has also been explored in mathematical physics (see e.g. [Zab89; CLH21]).

Bruhat–Tits theory puts the Bruhat–Tits tree into a larger number theoretic context. Keeping the setting of a non-archi-medean local field K for simplicity, let us remark that GL_2/K is an example of a connected reductive group. Now much more generally, to any connected reductive group G/K one can associate a so called Bruhat–Tits building, a contractible topological space that carries the structure of a polysimplicial complex equipped with an action of $G(K)$. Bruhat–Tits buildings are a key tool for the structure theory of reductive groups, in spirit analogous to symmetric spaces of real reductive Lie groups. We refer to the introduction of [KP23] for an overview of the theory. The simplest non-trivial example is the building (as in [KP23, Section 4.1]) for $G = \mathrm{GL}_2$, where one recovers the Bruhat–Tits tree. Figure 1 is an illustration of the Bruhat–Tits tree for $\mathbb{Q}_2$, the field of 2-adic numbers.

We review the construction of the Bruhat–Tits tree and describe its formalisation in detail in Section 3. For now let us mention that it is constructed as a simple graph, i.e., a graph such

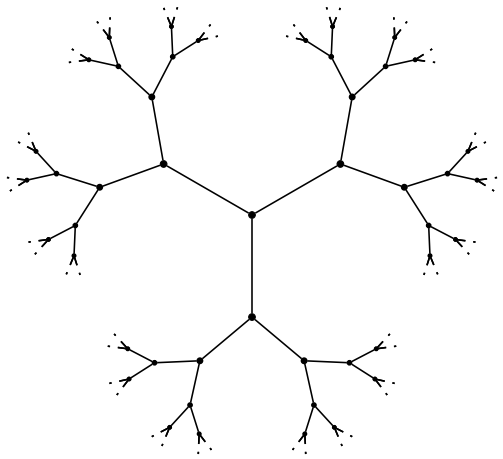


Figure 1: The Bruhat–Tits tree for $\mathbb{Q}_2$

that any two vertices are connected by at most one edge, and such that no edge connects a vertex to itself; it has no loops. Moreover, it is connected and has no cycles making it into a so-called tree. As the edges out of any vertex are in bijection with points in the projective line over the residue field, the graph is regular, i.e., the edges out of any vertex have the same cardinality.

The vertices of the Bruhat–Tits tree correspond to certain equivalence classes of lattices in K^2. To define the edges of this graph and to understand that it is indeed a tree, it is necessary to relate lattices to each other, i.e., to develop a concept of distance between lattices and to understand chains of lattices. This is achieved by studying how bases of lattices can be transformed into standard shapes and for that one uses a decomposition of the group $\mathrm{GL}_2(K)$ called the *Cartan decomposition*.

This decomposition exists for $\mathrm{GL}_n(K)$ (and more generally for connected reductive groups) and is of independent interest. In the context of p-adic fields for example, it has consequences for the theory of smooth representations of the p-adic group $\mathrm{GL}_n(K)$. In Section 2 we give the precise statement, more details on context and describe the formalisation of this decomposition.

Our formalisation adds to the ongoing formalisation effort of modern number theory. It is moreover motivated by connecting to current research. An ongoing research project of one of us (J.L.) prominently features the Bruhat–Tits tree via harmonic cochains. These are certain functions on the edges of the tree connecting to representation theory of GL_2 and to rigid analytic functions on Drinfeld's period domain. As we strive to write up our research findings in a way that is error-free, accessible, understandable and unambiguous, we are intrigued by

the idea that one day formalisation and verification tools will effortlessly support this process and improve on our documentation. This is still *Zukunftsmusik*, particularly when using the word effortless. But in an attempt to turn up the volume we applied our formalisation of the Bruhat–Tits tree to verify a result about harmonic cochains and gained some mathematical insights along the way. This application is explained in Section 4 below. We refer to Section 5.2 for further discussion on the research context.

To our knowledge, the Bruhat–Tits tree has never been formalised before in a proof assistant. The code for this project is available at the public Github repository. We organised our roughly 8000 lines of code into five folders. The folder `Graph` contains the main constructions regarding the Bruhat–Tits tree, with the file `Graph/Tree.lean` containing the main theorem `theorem BTtree`. We refer to the Readme for a detailed overview of the structure of the repository.

We accompany our explanations of the formalisation by some Lean code snippets. We have edited some of them for clarity of exposition. All links to declarations in mathlib or in our repository are pinned to a specific commit and will therefore remain valid.

1.1 Lean and mathlib

This formalisation project was done in the interactive theorem prover Lean 4, which is also a functional programming language and is based on the Calculus of Inductive Constructions. For more background on Lean, we refer to [MU21; Avi+]. We use the stable version `v4.19.0`.

Our project is based on Lean's monolithic mathematical library `mathlib4`, an extensive and fast growing open source library of mathematics [mat20]. At the time of writing mathlib contains roughly 100 000 definitions and 200 000 theorems. The mathlib community strives to build a unified library of mathematics. Our project profited from this, as in particular mathlib contains both the definition of a discrete valuation ring and that of a regular tree. Furthermore we were able to build on the formalisation of foundational undergraduate material from linear algebra and algebra, importantly, the theory of modules over a ring. Working with the graph theory part of the library was quite pleasant, as almost all prerequisites that we needed (such as the notion of a simple graph and a regular tree) were at our disposal.

Merely in the context of formalising the Cartan decomposition our project would have benefited from more existing API on matrices, as this result requires some tedious manipulation of matrices.

We have started to integrate parts of the project into mathlib. We aim to fully integrate the Cartan decomposition as well as the formalisation of the Bruhat–Tits tree. For more details see Section 5.1.

2 The Cartan decomposition of $\mathrm{GL}_n(K)$

Let R be a discrete valuation ring, with uniformiser ϖ and fraction field K. Let val denote the valuation on K. Note that here and in mathlib by valuation we mean the absolute value (i.e., the norm) and not the additive valuation. Consider the group $\mathrm{GL}_n(K)$ of invertible $n \times n$ matrices. Let $T(K)$ be the subgroup of diagonal matrices and let T^- be the subset of $T(K)$ given by

$$T^- = \{\mathrm{diag}(\varpi^{f_1}, \ldots, \varpi^{f_n}) : f_1 \geq \ldots \geq f_n \in \mathbb{Z}\}.$$

Then we have

Theorem 2.1 (Cartan decomposition)**.** *There is a decomposition of* $\mathrm{GL}_n(K)$ *into a disjoint union of double cosets*

$$\mathrm{GL}_n(K) = \bigsqcup_{t \in T^-} \mathrm{GL}_n(R) \cdot t \cdot \mathrm{GL}_n(R). \tag{2.1}$$

The Cartan decomposition (for $\mathrm{GL}_2(K)$) is important for the construction of the Bruhat–Tits tree (see below). There is an analogous but historically earlier decomposition for Lie groups and Lie algebras with the same name (and which we do not formalise).

The proof of Theorem 2.1 that we formalise can be described as a Gaussian type algorithm. Alternatively, the decomposition can be deduced from the theory of the Smith normal form for matrices over a PID. The advantage of the former is that it is simpler and in parts works over general valuation rings.

An analogue of the Cartan decomposition exists more generally for connected reductive groups over non-archimedean local fields K. For a reference in this generality, see e.g. [BT72, Proposition 4.4.3].

When K is a non-archimedean local field, the Cartan decomposition is also an essential result in the theory of smooth representations of $\mathrm{GL}_n(K)$, a theory that is of fundamental importance in the Langlands programme. For instance, the Cartan decomposition implies the countability of the set $\mathrm{GL}_n(K)/\mathrm{GL}_n(R)$ which guarantees that one has Schur's lemma. It is also used to show the commutativity of spherical Hecke algebras and to compute spherical vectors. Note that in these applications, the field is assumed to be complete with respect to the valuation, but for the Cartan decomposition itself this is not necessary.

One may also ask to what extent the Cartan decomposition holds for more general valuation rings. There one still has the decomposition

$$\mathrm{GL}_n(K) = \mathrm{GL}_n(R)T(K)\mathrm{GL}_n(R),$$

where as above $T(K)$ is the abelian subgroup of diagonal matrices. However in the absence of a uniformiser, there is no natural finer decomposition.

The proof of Theorem 2.1 is divided into two parts. One first shows existence of the decomposition, i.e., that one can write any element $g \in \mathrm{GL}_n(K)$ as $g = k_1 t k_2$, for $k_1, k_2 \in \mathrm{GL}_n(R)$ and some $t \in T^-$. Then, in a second step, one shows uniqueness of the decreasing tuple $(f_1, \ldots, f_n)$ of integers that defines the element $t \in T^-$. This implies the disjointness of the double cosets in (2.1). One also shows that the tuple $(f_1, \ldots, f_n)$ does not depend on the choice of the uniformiser.

The proof of existence is an inductive linear algebra argument making use of the fact that one has a valuation to order matrix entries. The strategy for the formalisation is inspired by Sébastien Gouëzel's proof that every matrix can be written as a product of a diagonal matrix and transvections.

To elaborate a bit, note that the group $\mathrm{GL}_n(R)$ contains permutation matrices. So by multiplying a given matrix g from the left and the right by elements in $\mathrm{GL}_n(R)$, we can swap rows and columns and move matrix entries. In particular we can normalise the matrix so that the entry in the lower right corner has maximal valuation.

```
lemma exists_normalization0 {n : ℕ} (g : Matrix (Fin n ⊕ Unit) (Fin n ⊕ Unit) K) :
  ∃ (k₁ k₂ : GL (Fin n ⊕ Unit) R),
   v ((k₁.val * g * k₂.val) (Sum.inr _) (Sum.inr _)) = g.coeffs_sup v := by
/- ... -/
```

The notation `.val` denotes the underlying matrix of an element of `GL (Fin n ⊕ Unit) R` and `Fin n` denotes a type of n elements. Note that we use `Fin n ⊕ Unit` instead of `Fin (n + 1)` to avoid casting between `Fin n` and `Fin (n + 1)` when using this in the induction step. The price we pay is that we need to write `Sum.inr _` for the $(n+1)$-st entry. Moreover `g.coeffs_sup v` denotes the supremum of the valuations `v` of the coefficients of the matrix g.

From there we can proceed to eliminate non-zero elements in the last row and column. The outcome is a block matrix in $\mathrm{GL}_{n-1}(K) \times \mathrm{GL}_1(K)$ and we can keep repeating this process to arrive at a diagonal matrix.

```
lemma exists_trafo_isDiag
 (g : Matrix (Fin n) (Fin n) K) :
 ∃ (k₁ k₂ : GL (Fin n) R),
  IsMonotoneDiag (k₁ * g * k₂) ∧
  (k * g * k₂).coeffs_sup v = g.coeffs_sup v := /- ... -/
```

The predicate `Matrix.IsMonotoneDiag` expresses that an $n \times n$-matrix over K is diagonal and that the valuation of the diagonal entries increases monotonically.

Up until this point, the proof works over an arbitrary valuation ring. The additional assumption that the valuation is discrete is needed for the final step, namely to get the diagonal matrix into the prescribed form, i.e., with entries ordered powers of the uniformiser.

```
theorem cartan_decomposition [IsDiscreteValuationRing R]
 (g : GL (Fin n) K) :
 ∃ (k₁ k₂ : GL (Fin n) R)
  (f : Fin n → ℤ), Antitone f ∧
 k₁ * g * k₂ = cartanDiag ϖ hϖ f := /- ... -/
```

Here `f` takes the role of the integers $f_1, \dots, f_n$ and the condition $f_1 \geq \cdots \geq f_n$ is encoded by `Antitone f`. For a discussion of the definition of `IsDiscreteValuationRing` see [FFC24].[1]

To prove the uniqueness part, one reduces to showing the following: Let $k_1, k_2 \in \mathrm{GL}_n(R)$ and $t, t' \in T(K)$ with entries integral powers ϖ^{f_i} of the uniformiser such that $tk_1t' = k_2 \in \mathrm{GL}_n(R)$. Then there exists a permutation $\sigma \in S_n$ such that $\sigma(t) = t'$, where $\sigma(t)$ is the matrix obtained from t by permuting the entries. For this one argues using the determinant and e.g. the fact that $|\det(x)| = 1$ for any $x \in \mathrm{GL}_n(R)$.

In Lean, the uniqueness statement then reads as follows.

```
theorem cartan_decomposition_unique
 {k₁ k₂ k₁' k₂' : GL (Fin n) R}
 {f f' : Fin n → ℤ} (hf : Antitone f)
 (hf' : Antitone f')
 (h : k₁ * cartanDiag ϖ hϖ f * k₂ = k₁' * cartanDiag ϖ hϖ f' * k₂') :
 f = f' := /- ... -/
```

For completeness, we also formalise a more literal version of Theorem 2.1, but when applying the Cartan decomposition, the two formulations we give above are more useful.

[1] The cited article is written for Lean3 and `IsDiscreteValuationRing` was still called `discrete_valuation_ring`.

In the formalisation of the proof we often have to pass between elements of $\mathrm{GL}_n(R)$ and elements of $\mathrm{GL}_n(K)$. Since these two are distinct types, we use a *coercion* to ease the transition.

```
instance : CoeHead (GL (Fin n) R) (GL (Fin n) K) where
  coe g := GeneralLinearGroup.map R.subtype g
```

The function `GeneralLinearGroup.map R.subtype` is the inclusion $\mathrm{GL}_n(R) \to \mathrm{GL}_n(K)$ and registering this coercion means that Lean automatically inserts this function when it sees an element of $\mathrm{GL}_n(R)$, but expects an element of $\mathrm{GL}_n(K)$.

During the formalisation, we expanded the general purpose API for (invertible) matrices with elementary constructions such as swap matrices and valuations of coefficients.

3 The Bruhat–Tits Tree

We keep the notation from the previous section so that R denotes a discrete valuation ring. In this section we describe the formalisation of the Bruhat–Tits tree $\mathscr{T}$ in detail.

For that let us briefly review its construction. Consider the set $\mathrm{Latt}(R)$ of R-lattices $L \subset K^2$, i.e., of finitely generated R-submodules of K^2 that span K^2 as a K-vector space. Two lattices L, L' are called homothetic if one is a scalar multiple of the other, i.e., if there exists $\alpha \in K^\times$, such that $L = \alpha L'$. Homothety defines an equivalence relation on $\mathrm{Latt}(R)$ and the set of *vertices of* $\mathscr{T}$ is defined as the set of equivalence classes

$$V(\mathscr{T}) := \mathrm{Latt}(R)/\sim.$$

For example the *standard lattice* R^2 gives rise to a vertex v_0, which we refer to as the standard vertex. To define the edges, one associates to each pair of vertices (v, v') a natural number $d(v, v')$, their *distance*. We describe this function along with the formalisation in Section 3.2 below. Its construction requires a normalisation result on bases of pairs of lattices and uses the Cartan decomposition.

The distance function allows us to define two vertices v, v' as neighbours, i.e., they are connected by an edge if $d(v, v') = 1$. An equivalent characterization is as follows: two vertices $v, v' \in \mathscr{T}$ are neighbours if there exist representatives $v = [L]$ and $v' = [L']$ such that

$$\varpi L \subsetneq L' \subsetneq L.$$

The resulting graph is a tree, i.e., connected and acyclic, and is called the *Bruhat–Tits tree*. When the residue field of R is finite, the Bruhat–Tits tree is regular of degree $q + 1$, where q is

the cardinality of the residue field. Here regular of degree $d \in \mathbb{N}$ means that every vertex has exactly d neighbours.

Recall that we are assuming that the field K is discretely valued. As indicated in the introduction, in many applications of the Bruhat–Tits tree the field K is assumed to be a local field and in particular complete (or at least, as e.g. in [KP23], Henselian with perfect residue field). For the construction of the Bruhat–Tits tree itself and for further notions and results formalised in this project completeness is not needed and therefore we do not assume it anywhere.

3.1 Lattices

For the formalisation of the notion of lattices, let us pass to a more general setting. Let K be any field, let $R \subset K$ be a subring and consider R-submodules of K^n. Our main definition is the predicate when such a submodule is a lattice:

```
class IsLattice (M : Submodule R (Fin n → K)) : Prop where
 /-- `M` is finitely generated. -/
 isFG : M.FG
 /-- `M` spans `Fin n → K` -/
 spans : Submodule.span K (M : Set (Fin n → K)) = ⊤
```

This is implemented as a type class, following the general design principle of mathlib. While this definition is useful as it is easy to verify, in practice we want to work with explicit descriptions of lattices via bases. While every free R-submodule of K^n of rank n is clearly a lattice the converse might not be true. But when R is a principal ideal domain, every lattice is indeed free.

```
instance IsLattice.free [IsPrincipalIdealRing R]
   (M : Submodule R (Fin n → K))
   [IsLattice M] : Module.Free R M :=
 /- ... -/
```

The extensible, typeclass based setup of `IsLattice` is well suited for developing the general theory of lattices. In order to construct the Bruhat–Tits tree for a given R though, a *type* of vertices is required that can be equipped with a graph structure. Hence we also need a type of R-lattices, which we define as follows.

```
structure Lattice where
 M : Submodule R (Fin 2 → K)
 isLattice : IsLattice M
```

We restrict to $n = 2$ mostly for simplicity, since we would have to write `Lattice n R` otherwise.[2] Note that this is unrelated to the order-theoretic notion of a lattice, which also carries the name `Lattice` in mathlib. Our `Lattice` declaration is therefore namespaced as `BruhatTits.Lattice`, but we drop the namespace in the following, when there is no risk of confusion.

The vertices of $\mathscr{T}$ are defined as lattices up to homothety and with the above, the type of vertices can then be defined as a quotient of the type `Lattice`.

Note that, when building on top of this, for extensibility as much as possible should be developed in terms of `IsLattice` and only if needed, the constructions should be applied to the type `Lattice` in a last step. An example of this pattern is the `Lattice/Construction.lean` file.

3.2 Distance function

We return to the setting of a discrete valuation ring R and from now on only consider lattices in K^2. An essential tool for defining the Bruhat–Tits tree as a graph and for proving that it is indeed a tree, is the construction of the above mentioned distance function on pairs of vertices. For that, a crucial ingredient is the following proposition:

Proposition 3.1 (). *Let M, L be R-lattices. Then there exists an ordered R-basis (e, f) of M and integers $n \geq m$ such that $(\varpi^n e, \varpi^m f)$ form an R-basis of L. The integers are uniquely determined by M and L and in particular do not depend on the R-basis of M.*

In Lean the existence part of this proposition reads as:

```
lemma exists_normal_basis (M L : Lattice R) :
  ∃ (bM : Basis (Fin 2) R M.M) (bL : Basis (Fin 2) R L.M) (f : Fin 2 → ℤ),
  Antitone f ∧
  ∀ i, (bL i).val = (ϖ ^ f i : K) • (bM i).val
```

Here, for a term `x : M`, the notation `x.val` denotes the underlying element of K^2. This is an invocation of the inclusion map $M \to K^2$.

Proof of Proposition 3.1. We apply the Cartan decomposition: First note that for a lattice M the group $\mathrm{GL}_2(R)$ acts on the set of bases of M on the right. Now let (e, f) and (e', f') be arbitrary R-bases of M and L respectively. Interpreting these as the columns of elements g and h of $\mathrm{GL}_2(K)$, we may apply the Cartan decomposition to the product $g^{-1}h$ to obtain a factorization $g^{-1}h = k_1 d k_2$ where $d = diag(\varpi^n, \varpi^m)$ with $n \geq m$. Then $(e, f) \cdot k_1$ and $(e', f') \cdot k_2^{-1}$ work with the pair (n, m). □

[2] For planned adaptations for the integration into mathlib, we refer to Section 5.1.

We can now define the *distance* of two lattices M and L, denoted by $d(M,L)$ as the difference $n-m$ of the unique pair of integers from Proposition 3.1. One then easily checks that $d(M,L)=d(L,M)$, $d(M,L)=0$ if and only if $M=L$ and that d is invariant under scalar multiplication by $K^\times$. In particular, d also induces a well-defined function on the vertices (called `inv` in the formalisation) and we say two vertices x and y are *neighbours* if $d(x,y)=1$.

By the properties of the distance, we immediately obtain that this endows $V(\mathcal{T})$ with the structure of a simple graph, which in Lean reads as follows:

```
def BTgraph : SimpleGraph (Vertices R) where
 Adj L M := BruhatTits.IsNeighbour L M
 symm L M := /- ... -/
 loopless L := /- ... -/
```

A priori, $\mathcal{T}$ now has two notions of distance: The one defined above and the graph-theoretic one, given by infima over lengths of paths. These two notions agree, as we show in Section 3.4.

In order to show that `BTgraph` is indeed a tree, we need to show that it is connected and acyclic. Connectedness is straightforwardly proven by induction on the distance (`BTgraph_connected`). Acyclicity is more involved and we explain the proof in Section 3.4.

3.3 Perspectives on lattices

When working with the Bruhat–Tits graph in practice and while showing it is a tree, we rely heavily on the fact that any lattice over a PID is free. In particular, many proofs start by choosing appropriate bases for the lattices in play. How we represent such a basis in Lean, depends on our perspective on lattices.

Restricting to the rank 2 case, we can view a lattice as

(1) a free R-submodule of K^2 of rank 2,
(2) the R-span of a K-basis of K^2, or
(3) the R-span of the columns of an element of $\mathrm{GL}_2(K)$ (as above in the proof of Proposition 3.1).

While informally we freely switch between these perspectives, formally all of these are different and this difference shows most clearly when multiple lattices are in play. We focus on comparing the first two viewpoints.

For the comparison, we consider a variant of Proposition 3.1, which is essentially an unfolding of the definition of `dist`:

Lemma 3.2. *Let M, L be lattices. Then there exists an R-basis (e, f) of M and integers $n \geq m$ such that $(\varpi^n e, \varpi^m f)$ is an R-basis of L and $n - m = d(M, L)$.*

This is written from the first perspective, as we work with R-bases of M and L. One possible formalisation of this statement is the following:

```
lemma exists_repr_dist (M L : Lattice R) :
 ∃ (bM : Basis (Fin 2) R M) (bL : Basis (Fin 2) R L) (f : Fin 2 → ℤ),
  Antitone f ∧
  (∀ i, (bL i).val = (ϖ ^ f i : K) • (bM i).val) ∧
  f 0 - f 1 = dist M L :=
 /- ... -/
```

What corresponds to the statement $(\varpi^n e, \varpi^m f)$ *is an R-basis of L* in 3.2 is expressed in the formal version as *there exists a basis (e', f') of L such that $e' = \varpi^n e$ and $f' = \varpi^m f$*. Note though, that the last two equalities do not type-check. Formally, the elements `ϖ ^ n • e` and `ϖ ^ n • f` are of type `M`, while `e'` and `f'` are of type `L`. Hence to state these equalities in Lean, we have to compare `ϖ ^ n • e` and `e'` inside K^2, which explains the appearance of `.val` and the explicit type annotation in the above snippet. In particular this means that we cannot use the induction principle of equality on these equalities.

In order to avoid casting elements of M into elements of K^2 via `.val` while still keeping the first perspective, one can introduce a predicate `IsBasisOn`, which expresses that a given indexed family $(b_i)_{i \in I}$ of elements of a vector space V is a basis for the R-submodule M, i.e., it is linearly independent over R and generates M as an R-module. Concretely, this could be implemented as :

```
class IsBasisOn {R M : Type*} [Ring R] [AddCommGroup V] [Module R V]
  (b : ι → V) (M : Submodule R V) : Prop where
 linearIndependent : LinearIndependent R b
 span_range_eq : Submodule.span R (Set.range b) = M
```

Then instead of working with terms of type `Basis ι R M`, we only work with families `(b : ι → V) (M : Submodule R V) [IsBasisOn b M]`.

Our above lemma could then be formulated as

```
lemma exists_repr_dist₂ (M L : Lattice R) :
 ∃ (b : Fin 2 → (Fin 2 → K))
  [IsBasisOn b M] (f : Fin 2 → ℤ),
  Antitone f ∧
  IsBasisOn (fun i ↦(ϖ ^ f i : K) • b i) L ∧
  f 0 − f 1 = dist M L := /− ... −/
```

The `.val` has disappeared, since for `i : Fin 2`, both `b i` and `(ϖ ^ f i : K) • b i` are terms of type `Fin 2 → K`. As a side effect, the statement is very close to the informal formulation of Lemma 3.2.

Let us now give a formal statement of Lemma 3.2 from the second perspective. This means we don't start from R-submodules of K^2 and their R-bases, but instead from K-bases of K^2 and consider the submodules they span. The key definition is the following, which associates to a K-basis (b_0, b_1) of K^2 the R-lattice spanned over R by (b_0, b_1).

```
def Basis.toLattice (b : Basis (Fin 2) K (Fin 2 → K)) : Lattice R where
 M := Submodule.span R (Set.range b)
 isLattice := /− ... −/
```

Since in our context, every lattice is free and the underlying elements of an R-basis of a lattice form a K-basis of K^2, the function `Basis.toLattice` has a section, assigning an arbitrarily chosen basis to a lattice M (and). In particular, every lattice is of the form `Basis.toLattice b` for some basis b.

To formulate Lemma 3.2 in this language, it remains to add the twist of a basis by a tuple of integers: Given a basis $b = (b_0, b_1)$ the twist `b.twist f` by a pair of integers $f = (f_0, f_1)$ is the basis $(\varpi^{f_0} b_0, \varpi^{f_1} b_1)$. With these definitions, the third version of our lemma is:

```
lemma exists_repr_dist₃ (M L : Lattice R) :
 ∃ (b : Basis (Fin 2) K (Fin 2 → K)) (f : Fin 2 → ℤ),
  Antitone f ∧
  M = b.toLattice ∧
  L = (b.twist f).toLattice ∧
  f 0 − f 1 = dist M L := /− ... −/
```

Note that instead of comparing two bases (or indexed families), we have two equalities of lattices: `M = b.toLattice` and `L = (b.twist f).toLattice`. As in the second formulation from the first perspective, this eliminates the need for invocations of `.val` and allows to use the induction principle of equality. As an example of the application of Lemma 3.2 in this

formulation, consider the proof that the distance function is invariant under the action of $\mathrm{GL}_2(K)$:

```
lemma dist_smul_GL_eq_dist (M L : Lattice R) (g : GL (Fin 2) K) :
  dist (g • M) (g • L) = dist M L := by
 obtain ⟨ϖ, hϖ, b, f, hf, hM, hL, hdiff⟩ := exists_repr_dist₃ M L
 subst hM hL
 rw [← Basis.smulGL_toLattice, ← Basis.smulGL_toLattice, Basis.smulGL_twist]
 /- ... -/
```

Here in the first line, we invoke `exists_repr_dist₃` to obtain equalities `hM` and `hL` on which we apply the `subst` tactic to replace all occurrences of `M` (resp. `L`) by `g • b.toLattice` (resp. `g • (b.twist f).toLattice`). As the name suggests, the `subst` tactic is Lean's equivalent of performing substitutions. This is done by applying the induction principle of equality. Then we can use rewrite lemmas to interchange the action of $\mathrm{GL}_2(K)$ with various operations on bases and lattices.[3]

What makes the second perspective work well is that we may assume that every lattice is of the form `b.toLattice` for some basis `b`. To then do calculations with lattices, we make use of API lemmas like the ones used in the proof of `dist_smul_GL_eq_dist` of which the following is an example,

```
lemma smulGL_toLattice (g : GL (Fin 2) K) (b : Basis (Fin 2) K (Fin 2 → K)) :
  (g • b).toLattice = g • b.toLattice := /- ... -/
```

It says that the actions of $\mathrm{GL}_2(K)$ on bases of K^2 and on R-lattices commute with the induced lattice construction.

The common pattern in the variants `exists_repr_dist₂` and `exists_repr_dist₃` is the avoidance of calculating in types that depend on the specific lattices in play (such as `Basis (Fin 2) R M` and `M` itself). Instead the types of all data carrying terms only depend on `R` and `K` and this is what enables the induction principle of equality.

The difference of the two variants is that the former uses equalities in `Fin 2 → K`, while the latter uses equalities in `Lattice R`. Since in the context of this project we work with vertices of the tree, equality of lattices is typically more useful. This is why the latter approach is what is now broadly used in the project.

[3] See `dist_smul_GL_eq_dist` for the full proof.

3.4 Acyclicity

The reason why showing connectedness is straightforward is that it only involves two vertices, which we can immediately bring into a compatible normal form using the lemma `exists_repr_dist₃`. The same trick does not work, when three or more lattices are in play. However, in good enough cases we may transform into a normal form, using the natural $\mathrm{GL}_2(K)$-action on $\mathscr{T}$.

By normal form we mean the following: Let $(v_0, \dots, v_n)$ be a walk in $\mathscr{T}$, i.e., a sequence of adjacent vertices v_i. Following [Cas19, Section 4], we call a walk $(v_0, \dots, v_n)$ a *standard chain* if there exists a basis (e, f) of a representative of v_0 such that $(\varpi^k e, f)$ is a basis of a representative of v_k for every $0 \leq k \leq n$. Note that by Proposition 3.1, any walk of length one is a standard chain. By construction of the distance, if $(v_0, \dots, v_n)$ is a standard chain, then $d(v_0, v_n) = n$.
The key tool now is:

Lemma 3.3 (). *Let p be a trail in $\mathscr{T}$, i.e. a sequence of adjacent vertices $p = (v_0, \dots, v_n)$ without repeating edges. Then there exists $g \in \mathrm{GL}_2(K)$ such that $(gv_0, \dots, gv_n)$ is a standard chain.*

The proof of Lemma 3.3 follows closely [Cas19, Proposition 4.1] and is fairly technical as it involves going back and forth between submodules of the quotient $L/\varpi L$ and submodules of K^2 sitting between ϖL and L. The formalisation of the proof is contained in the file `Lattice/Quotient.lean`. This is the slowest of our files but we expect the performance to improve when replacing subrings by algebras (see Section 3.7).

Since $\mathrm{GL}_2(K)$ acts by graph isomorphisms on $\mathscr{T}$, Lemma 3.3 implies that the length of any trail in $\mathscr{T}$ agrees with the distance of its endpoints. So indeed our distance function and the graph theoretical distance agree .
Moreover, in the special case of distance zero, any trail starting and ending at the same point has length zero, so $\mathscr{T}$ is acyclic. Putting all the pieces together yields:

```
theorem BTtree : SimpleGraph.IsTree (BTgraph R) where
 isConnected := BTconnected
 IsAcyclic := BTacyclic
```

3.5 Regularity

We briefly comment on regularity. Let $\mathfrak{m}$ denote the maximal ideal of R and let k denote the residue field $R/\mathfrak{m}$. For every lattice L, the neighbours of the vertex associated to L are in

one-to-one correspondence to one dimensional subspaces of the two-dimensional k-vector space $L/\mathfrak{m}L$, i.e., to the projectivization $\mathbb{P}_k(L/\mathfrak{m}L)$.

To see this it is convenient to introduce the notion of standard neighbours[4]: We say a lattice M is a *standard neighbour* of the lattice L if $\varpi L \subsetneq M \subsetneq L$. Note that this condition is independent of the choice of ϖ, since $\varpi L = \mathfrak{m}L$.

```
structure IsStandardNeighbour (M L : Lattice R) : Prop where
 lt : M.M < L.M
 ϖlt : ∀ (ϖ : R), Irreducible ϖ → ϖ • L.M < M.M
```

To avoid a dependence on an additional parameter ϖ in `IsStandardNeighbour`, we have the condition $\varpi L \subsetneq M$ for all uniformisers ϖ.

If M and L are standard neighbours, the associated vertices of M and L are neighbours and conversely, if a vertex v is a neighbour of the associated vertex of L, there exists an unique representative M of v such that M is a standard neighbour of L. Hence, the neighbours of the vertex associated to M are in one-to-one correspondence to standard neighbours of M. But the latter are exactly the one-dimensional subspaces of $L/\varpi L = L/\mathfrak{m}L$.

In particular, if k is finite of cardinality q, the Bruhat–Tits tree is regular of degree $\#(\mathbb{P}_k(L/\mathfrak{m}L)) = q + 1$.

3.6 Group actions

An important feature of the Bruhat–Tits tree is that it comes with a well-understood transitive action of $\mathrm{GL}_2(K)$. To formalise this result we first have to introduce the notion of a group action on a graph. We define this as an action on vertices that preserves the adjacency relation.

```
class GraphAction (G : Type*) [Group G] [MulAction G V] (X : SimpleGraph V) where
 smul_adj_smul (g : G) (x y : V)
 (h : X.Adj x y) : X.Adj (g • x) (g • y)
```

Now the group $\mathrm{GL}_2(K)$ acts on lattices in a way that preserves similarity and distance. As a consequence we get a graph action on $\mathscr{T}$.

```
instance : GraphAction (GL (Fin 2) K) (BTgraph R) where
 smul_adj_smul g x y := (adj_smul_smul_iff_adj g x y).mpr
```

This action is transitive. The stabilizer of the standard vertex is $K^\times \mathrm{GL}_2(R)$. Let (e_0, e_1) denote the standard basis of the standard lattice R^2 and consider the lattice L_1 with basis

[4] In fact we use this notion already for the proof of Lemma 3.3, and it is introduced in the file *Graph/Edges.lean*.

$(e_0, \varpi e_1)$. Then $v_0 = [R^2]$ and $v_1 := [L_1]$ are neighbours and the pointwise stabilizer of the edge e connecting v_0 to v_1 is given by $K^\times I_1$, where I_1 is the subgroup of $\mathrm{GL}_2(R)$ that is upper triangular modulo ϖ.

For the application in Section 4, we need to use the action of $\mathrm{SL}_2(K)$ on $\mathcal{T}$, which just acts as a subgroup of $\mathrm{GL}_2(K)$. Contrary to the case of $\mathrm{GL}_2(K)$ this action is no longer transitive. One way to see this is to note that $\mathcal{T}$ admits a natural orientation coming from a partition of the vertices into even and odd; which can be interpreted as a direction on the edges. A vertex is called even (resp. odd) if the determinant of the matrix spanned by a basis has even (resp. odd) additive valuation.

```
def IsEven (L : Lattice R) : Prop := Even L.valuation
```

Equivalently v is even (resp. odd) if its distance from the standard vertex v_0 is even (resp. odd). The action of $\mathrm{SL}_2(K)$ preserves this orientation, in the sense that it respects even and odd vertices.

```
lemma isEven_specialLinearGroup_smul_iff {x : Vertices R}
  (g : SL (Fin 2) K) : IsEven (g • x) ↔ IsEven x := /- ... -/
```

3.7 Subrings or algebras

Informally, when thinking about a domain R and its fraction field K, we view R as a subring of K. Alternatively, we may view K as an R-algebra, where the induced ring homomorphism $R \to K$ is injective. In Lean, introducing a variable R that is a subring of an already declared field `K` is as simple as writing `(R : Subring K)`. In contrast, the second point of view is spelled as `(R : Type) [CommRing R] [Algebra R K] [FaithfulSMul R K]`. The assumption `FaithfulSMul R K` expresses that the canonical ring homomorphism $R \to K$ is injective.

When starting the project, we chose the naive `Subring` approach, mostly for two reasons: The first is simplicity. Not only does the initial introduction of `R` require only one variable instead of four, also every K-module is automatically an R-module. This removes the need for additional `Module R V` and `IsScalarTower R K V` assumptions, that obfuscate definition and theorem statements.[5] Secondly, the field K is carried in the type of R, which means the type `IsLattice M` is unambiguous. In the algebra setting, we need an additional type parameter for K. Having the type of K in M also means that we may register instances such as `Module. Finite R M` for any submodule satisfying `IsLattice M`. [6]

[5] Compare the upstreamed definition `Submodule.IsLattice` with our original one.

[6] In the `Algebra R K` approach, there are technical ways to still register this instance using an `outParam` for `K`.

One disadvantage of the naive approach is the loss of modularity. While going from `Subring K` to `Algebra R K` is automatic by typeclass inference, results formulated using `Subring K` can not be easily used when starting from `Algebra R K`. This issue appears less though, when only one field and one subring are in play, which is the case in this project. Another disadvantage of `Subring K` is the worse performance, which is related to slow typeclass synthesization due to more information in the type.[7]

We eventually chose to use the first approach for the sake of a simpler presentation, but plan to adapt it to the `Algebra R K` setting for integration into mathlib.

4 Harmonic cochains

We apply our formalisation of the Bruhat–Tits tree to verify a result involving harmonic cochains. These form an important tool in the theory of automorphic forms over function fields and in Section 5.2 below, we give more context as to why we chose this application.

To introduce harmonic cochains recall that $V(\mathscr{T})$ denotes the set of vertices of the Bruhat–Tits tree $\mathscr{T}$. We denote the edges of $\mathscr{T}$ by $E(\mathscr{T})$. By assumption an edge $e \in E(\mathscr{T})$ is an unordered pair $\{v, w\}$ of adjacent vertices (or neighbours, as we called them above) $v, w \in V(\mathscr{T})$. For $v \in V(\mathscr{T})$ and $e \in E(\mathscr{T})$ we write $v \in e$ when v is a vertex of the edge e.

Recall from Section 3.6, that we have a partition of the vertices into even and odd ones. Consider the map $w_{\mathrm{par}} : V(\mathscr{T}) \to \{\pm 1\}$ that sends even vertices to $+1$ and odd vertices to -1.

Let A be a commutative ring (with 1) and M be an A-module. For any set X, we let $\mathrm{Maps}(X, M)$ be the A-module of maps from X to M, with pointwise addition and scalar multiplication.

Definition 4.1. *(1) Define the* Bruhat–Tits Laplacian Δ *as the A-module homomorphism*

$$\begin{aligned} \Delta : \mathrm{Maps}(E(\mathscr{T}), M) &\to \mathrm{Maps}(V(\mathscr{T}), M) \\ f &\mapsto \left(v \mapsto w_{\mathrm{par}}(v) \sum_{\substack{e \in E(\mathscr{T}) \\ v \in e}} f(e) \right). \end{aligned}$$

(2) Define the A-module of M-valued harmonic cochains *as the kernel of Δ and denote it by* $\mathrm{Har}(\mathscr{T}, M)$.

[7] An example of this effect can be seen in this pull request to mathlib.

In number theoretic applications the setting of $A = M = \mathbb{Z}$ and K a local field is of particular interest. As the action of $\mathrm{SL}_2(K)$ on $V(\mathcal{T})$ respects the parity of the vertices, the map Δ is $\mathrm{SL}_2(K)$-equivariant. As we will show below, it is also surjective.

Therefore, we have a short exact sequence of $\mathrm{SL}_2(K)$-equivariant maps

$$0 \to \mathrm{Har}(\mathcal{T}, \mathbb{Z}) \to \mathrm{Maps}(E(\mathcal{T}), \mathbb{Z}) \xrightarrow{\Delta} \mathrm{Maps}(V(\mathcal{T}), \mathbb{Z}) \to 0. \tag{4.1}$$

This short exact sequence is well known (see, for example, [Put83, Proof of Prop. 1.7]). In forthcoming work of one of us (J.L.) with Gebhard Böckle and Oğuz Gezmiş we explore it in the context of function field automorphic forms. There the sequence serves as a starting point and a key tool to understand the size and structure of the cohomology group $H^1(\Gamma, \mathrm{Har}(\mathcal{T}, \mathbb{Z}))$ where Γ is a function field analogue of the Ihara group $\mathrm{SL}_2(\mathbb{Z}[1/p])$.

It was in our interest to formalise a direct elementary proof of surjectivity. As we will explain in the next sections, we can define and show surjectivity of the Laplacian in a more general setting. Alongside these details we also explain the formal verification of our proof.

4.1 Surjectivity of the Laplacian

Let $\mathcal{T}$ be a simple graph that is locally finite, i.e., every vertex has only finitely many adjacent vertices. As for the Bruhat–Tits tree we denote the vertices of $\mathcal{T}$ by $V(\mathcal{T})$ and the edges of $\mathcal{T}$ by $E(\mathcal{T})$. For a vertex $v \in V(\mathcal{T})$, we let

$$E_v := \{e \in E(\mathcal{T}) \mid v \in e\}$$

denote the set of edges containing v. For $v \in V(\mathcal{T})$ denote by $\deg(v) = \deg_{\mathcal{T}}(v)$ the (finite) number of neighbours of v in $\mathcal{T}$.

As before let A be a commutative ring and fix a map $w\colon V(\mathcal{T}) \to A^\times$. We call w a *weight function* on $\mathcal{T}$. Let M be an A-module.

Definition 4.2. *Let $f : E(\mathcal{T}) \to M$ be a map. Then we define*

$$\Delta_w(f)\colon V(\mathcal{T}) \to M, \quad v \mapsto w(v) \sum_{e \in E_v} f(e).$$

This defines a map

$$\Delta_w\colon \mathrm{Maps}(E(\mathcal{T}), M) \longrightarrow \mathrm{Maps}(V(\mathcal{T}), M),$$

which we refer to as the Laplacian of weight w *of the graph* $\mathscr{T}$. *If there is no risk of confusion, we also write* Δ *for* Δ_w.

Note that as $\mathscr{T}$ is locally finite, the Laplacian is well-defined. In Lean, the definition reads as:

```
def laplace (w : V → A^×) (f : X.edgeSet → M) (v : V) : M :=
 w v • Σ e ∈ X.incidenceFinset' v, f e
```

Before we state and prove the surjectivity, we need some preparations. Assume that $\mathscr{T}$ is a tree. Then the set $V(\mathscr{T})$ of vertices is non-empty and for every pair of vertices $v, w \in V(\mathscr{T})$ there is a unique path from v to w.
The length of this path is called the *distance*[8] of v and w in $\mathscr{T}$ and is denoted by $d(v,w) = d_{\mathscr{T}}(v,w)$. Let us fix a vertex v_0 of $\mathscr{T}$. We consider v_0 as the root vertex of $\mathscr{T}$. For a vertex v of $\mathscr{T}$, denote by $|v| = |v|_{v_0}$ the distance $d(v, v_0)$ of v to the root vertex v_0.

As $\mathscr{T}$ is a tree, for any edge e of $\mathscr{T}$ there exist unique vertices $s(e)$, the *source* of e, and $t(e)$, the *target* of e, such that $e = \{s(e), t(e)\}$ and $|s(e)| < |t(e)|$.

Note that for a vertex v of $\mathscr{T}$ with $|v| > 0$, there exists a unique edge o_v with $t(e) = v$. In other words, o_v is the unique adjacent edge of v pointing towards v_0. Let us assume that $\deg(v) \geq 2$ for all v. Then there exists at least one adjacent edge e of v with $e \neq o_v$, and then $s(e) = v$ and e is pointing away from v_0. Note that this does not require $|v| > 0$ so it also holds for $v = v_0$.

Definition 4.3. *For a vertex* v *we denote by* Out_v *the subset of* E_v *containing the edges* e *with* $s(e) = v$ *and call it the* outward edge cone *of* v.

We have

$$E_v = \begin{cases} \mathrm{Out}_v \cup \{o_v\} & v \neq v_0 \\ \mathrm{Out}_v & v = v_0 \end{cases}.$$

Note that for every $v \in V(\mathscr{T})$ the outward edge cone $\mathrm{Out}_v \neq \emptyset$ is non-empty, since $\deg(v) \geq 2$. Now choose for every $v \in V(\mathscr{T})$ a distinguished edge $d_v \in \mathrm{Out}_v$.

Proposition 4.4 (). *Suppose* $\mathscr{T}$ *is a tree such that* $2 \leq deg(v) < \infty$ *for all* $v \in V(\mathscr{T})$. *Let* w *be a weight function on* $\mathscr{T}$. *Then the Laplacian* Δ_w *is surjective.*

[8] As explained above, in the case of the Bruhat–Tits tree this agrees with the distance function from Section 3.2.

```
lemma laplace_surjective
 {X : SimpleGraph V} (htree : IsTree X)
 [∀ v, Fintype (X.neighborSet v)]
 (w : V → A ˣ) (hmindeg : ∀ v, 2 ≤ X.degree v) :
 Function.Surjective (X.laplace M w) := /- ... -/
```

Proof. We prove this by constructing for any function $f : V(\mathscr{T}) \to M$ a preimage under Δ_w. So let f be such a function. For $n \in \mathbb{N}_0$ denote by B_n the ball of radius n in $E(\mathscr{T})$, by which we mean

$$B_n = \{e \in E(\mathscr{T}) \mid |t(e)| \leq n\}.$$

We now inductively construct a function $h_n \colon B_n \to M$ for $n \geq 0$ such that

(i) $h_{n+1}|_{B_n} = h_n$,
(ii) $f(v) = w(v) \sum_{e \in E_v} h_{n+1}(e)$ for every $v \in V(\mathscr{T})$ with $|v| \leq n$.

Note that the second condition is well-defined, since for $v \in V(\mathscr{T})$ with $|v| \leq n$ every edge e with $v \in e$ satisfies $|t(e)| \leq n+1$.

For $n = 0$, the set B_0 is empty and there is nothing to do. For $n = 1$, every $e \in B_1$ satisfies $s(e) = v_0$. Hence $B_1 = E_{v_0} = \mathrm{Out}_{v_0}$. Now define

$$h_1 \colon B_1 \to M,\ h_1(e) = \begin{cases} w(v_0)^{-1} f(v_0) & e = d_{v_0} \\ 0 & e \neq d_{v_0} \end{cases}.$$

Condition (i) is trivially satisfied since $B_0 = \emptyset$. Moreover if $v \in V(\mathscr{T})$ satisfies $|v| \leq 0$, we have $v = v_0$ and

$$\begin{aligned} w(v_0) \sum_{e \in E_{v_0}} h_1(e) &= w(v_0) h_1(d_{v_0}) \\ &= w(v_0) w(v_0)^{-1} f(v_0) \\ &= f(v_0). \end{aligned}$$

Now suppose $n \geq 2$ and suppose h_n is already constructed. For $e \in B_n$ let $h_{n+1}(e) = h_n(e)$. For $e \in E(\mathscr{T})$ with $|t(e)| = n+1$ let $v = s(e)$. Note that $e \in \mathrm{Out}_v$. Define

$$h_{n+1}(e) = \begin{cases} w(v)^{-1} f(v) - h_n(o_v) & e = d_v \\ 0 & e \neq d_v. \end{cases} \tag{4.2}$$

By construction, condition (i) is satisfied. For condition (ii), let $v \in V(\mathscr{T})$ with $|v| \leq n$. For $|v| \leq n-1$ the induction hypothesis applies, thus suppose $|v| = n$. Recall that $E_v = \{o_v\} \cup \mathrm{Out}_v$. Hence

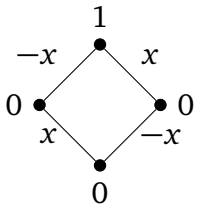
$$\begin{aligned} w(v)\sum_{e\in E_v} h_{n+1}(e) &= w(v)\left(h_{n+1}(o_v) + \sum_{e\in \mathrm{Out}_v} h_{n+1}(e)\right) \\ &= w(v)\left(h_{n+1}(o_v) + h_{n+1}(d_v)\right) \\ &= w(v)\left(h_n(o_v) + w(v)^{-1}f(v) - h_n(o_v)\right) \\ &= f(v). \end{aligned}$$

Now define $h\colon E(\mathscr{T}) \to M$ by $e \mapsto h_{|t(e)|}(e)$. Using conditions (i) and (ii) one now checks easily that $\Delta_w(h) = f$.

□

Remark 4.5.

- Note that our assumption that $deg(v) \geq 2$ is needed (at least for all but one vertex) as the simple example of a graph with two vertices and one connecting edge shows.
- Furthermore surjectivity breaks as soon as we allow loops with an even number of edges, as the following example illustrates.



  Let v denote the top vertex and let f be the function on the vertices that is encoded by the number written next to them, so that e.g. $f(v) = 1$. Then if the edge from v to the middle right vertex is assigned the value x, the values at all other edges are predetermined by the zeroes, but then it doesn't work at v.
- In the proof of Proposition 4.4, one may assume the weight function w to be trivial, i.e., identically 1. Indeed, Δ_w can be recovered from the Laplacian for the trivial weight function by pre-composing with the automorphism of $\mathrm{Maps}(V(\mathscr{T}), M)$ given by $f \mapsto (v \mapsto w(v)f(v))$.

4.2 Verification of surjectivity proof

The formal verification of the proof of Proposition 4.4 went smoothly and follows closely the proof outlined above. Remarkably, the LaTeX to Lean code ratio of this proof is significantly lower than in the rest of the project: Less than 750 lines of code (including documentation and all needed general purpose graph theory lemmas that were missing in mathlib) for 140 lines of LaTeX. Let us note here a few aspects of the formal proof.

Let `V` be a type of vertices and `X : SimpleGraph V`, such that every vertex has finitely many neighbours.

Firstly, the construction of the pre-image relies strongly on the arbitrary, but globally fixed choice of the distinguished edges in the, by assumption nonempty, outward edge cones. While informally, this might appear as a subtlety, in Lean this is straightforward:

```
def distinguishedEdge (w : V)
  (hw : (X.outwardEdgeCone v₀ w).Nonempty) : X.edgeSet :=
 hw.choose
```

Secondly, to keep track of the inductive construction of the pre-image and to check the required equations hold in each step, Lean served its role as a proof assistant very well. In particular, the construction of h_n from the informal proof can be expressed in a clear way by a recursive function `aux`:

```
def aux (n : ℕ) : X.edgeSet → M := match n with
 /- arbitrary value in step zero. -/
 | 0 => fun _ ↦ 0
 | n + 1 => fun e ↦
   /- if the norm of the edge is `> n + 1`, we give a junk value -/
   if h₁ : n + 1 < X.dist v₀ (X.target v₀ e) then 0
   else
    /- if the norm of the edge is `≤ n`, we use the induction hypothesis -/
    if h₂ : X.dist v₀ (X.target v₀ e) ≤ n then aux n e
    else
     /- if the norm of the edge is `n + 1`, it is on the current border, so we compute -/
     auxBorder w f v₀ n (aux n) e
```

The `auxBorder` function called in the induction step case is defined exactly as explained above in the informal proof, i.e., it is given by the formula (4.2). Note that, instead of defining `aux n` as a function on the ball of radius n, we define it as a function on all edges of `X`. On edges e with distance $|e| > n$, we instead pick a *junk value* of 0. This follows the recurring

pattern in formalisation to prefer unconditionally defined functions and instead only pass the necessary assumptions in proofs.

Finally, the formal counterpart for the definition of the preimage h is given by

```
def preimage (e : X.edgeSet) : M :=
 aux (X.dist v₀ (X.target v₀ e)) e
```

and from there it is only a few more lines to check that this is indeed a preimage and complete the verification of Proposition 4.4.

5 Concluding remarks

5.1 Integration into mathlib

Our aim is to fully integrate the formalisation of the Cartan decomposition and of the Bruhat–Tits tree into mathlib4. We have started to integrate some background and some preparations for the main files (a list of merged pull-requests can be found here). Notably we have added the notion of R-lattices and results about the cardinality of the projectivisation of a finite vector space over a finite field. We do not currently plan to integrate our results on harmonic cochains as they are rather specialised.

For integration into mathlib, besides the above mentioned replacement of `Subring K` by `Algebra R K`, the API for lattices, which has been mostly developed for submodules of K^2, needs to be generalised to submodules of an arbitrary finite dimensional K-vector space V. We do not expect any major obstactles here.

Regarding the integration of the Cartan decomposition, we are confident that we have worked at the right level of generality. For example, we work with general valuation rings for as long as possible. While in our original formalisation we adapt the strategy used in mathlib's development of transvections, for mathlib we want to generalise as much as possible from this file to a relative setting of an R-algebra K, where K is not necessarily a field, to avoid duplication of the core ideas.

5.2 Formalising ongoing research

One motivation for this project was to see how much of an ongoing research work can be formalised in the Lean Theorem Prover. The aim of that work is to determine the structure of a group $H^1(\Gamma, \mathscr{A}_\nu^\times/\mathbb{C}_\nu^\times)$ of so called rigid analytic theta cocycles. Here $\mathscr{A}_\nu^\times$ denotes the group of invertible rigid analytic functions on the Drinfeld upper half plane Ω_ν attached to a local function field K, and $\mathbb{C}_\nu^\times$ denotes the subgroup of constant functions. Moreover, Γ is a function

field analogue of the group $\mathrm{SL}_2(\mathbb{Z}[1/p])$. There is an identification of $\mathscr{A}_v^\times/\mathbb{C}_v^\times$ with the group of harmonic cochains $\mathscr{H}(\mathscr{T},\mathbb{Z})$ on the Bruhat–Tits tree $\mathscr{T}$ due to Van der Put [Put92, Theorem 2.1]. The strategy to prove our structural result is to start from the sequence (4.1) and analyze the long exact sequence in group cohomology $H^i(\Gamma,-)$ crucially invoking Shapiro's lemma. This leads in particular to a map $H^1(\Gamma,\mathscr{A}_v^\times/\mathbb{C}_v^\times)\to H^1(\Gamma_0(v),\mathbb{Z})$, where $\Gamma_0(v)$ is a classical congruence subgroup, and provides a link to function field automorphic forms.

As described in Section 4, in the context of this formalisation project we wrote down a direct proof that the sequence (4.1) is short exact and verified it. One may/should ask whether the argument sketched above can be verified. The answer is: Not without investing significant effort in formalising further theoretical background. For example, to the best of our knowledge, rigid analytic geometry in general and the Drinfeld upper half plane in particular have not been formalised. We also crucially need some background from group cohomology. Here the state of the art is different. By work of Amelia Livingston ([Liv23]) the categorical abstract approach is available in mathlib. Furthermore there is work on group cohomology in low degrees. Nevertheless, for our concrete calculations we would have to formalise Shapiro's lemma and develop more API to manipulate our cohomology groups. Given these, it would then be feasible to verify a significant amount of the argument above.

Let us end by remarking that we ended up using Lean to generalise some initial hypotheses. The reader might have noticed that the spaces of maps in the sequence (4.1) all have target $\mathbb{Z}$. When we were thinking about a direct proof of surjectivity and about formalising it, we were motivated to generalise the hypothesis to arbitrary commutative rings A. After the formalisation of `laplace` and the verification of `laplace_surjective`, as a map between A-valued functions it occurred to us, that instead of A-valued maps we should be working more generally with M-valued maps, where M is an arbitrary A-module. Thanks to Lean, this was as easy as one could hope for; essentially all we had to do was to replace the occurrences of A by M in the code.

Acknowledgments We are grateful to the Lean and mathlib community for their work on building a unified digital library of mathematics. Without mathlib, this project would not exist. J.L. would like to thank Gebhard Böckle and Oğuz Gezmiş for their support in taking our research on rigid analytic theta cocycles for function fields down the formalisation lane. We would like to thank Gebhard Böckle, Kevin Buzzard, Johan Commelin, Oğuz Gezmiş, Peter Schneider and Junyan Xu for helpful comments on an earlier draft of this paper and the anonymous referee for their comments and suggestions.

References

[Avi+] J. Avigad, L. de Moura, S. Kong, and S. Ullrich. *Theorem Proving in Lean 4*. URL: https://leanprover.github.io/theorem_proving_in_lean4/.

[BT72] F. Bruhat and J. Tits. "Reductive groups over a local field". French. In: *Publ. Math., Inst. Hautes Étud. Sci.* 41 (1972), pages 5–251. ISSN: 0073-8301. DOI: 10.1007/BF02715544. URL: https://eudml.org/doc/103918.

[Cas19] B. Casselman. *The Bruhat-Tits tree of* SL(2). https://api.semanticscholar.org/CorpusID:1614277. 2019.

[CLH21] L. Chen, X. Liu, and L.-Y. Hung. "Bending the Bruhat-Tits tree. Part II. The p-adic BTZ black hole and local diffeomorphism on the Bruhat-Tits tree". In: *JHEP* 09 (2021), page 097. DOI: 10.1007/JHEP09(2021)097. arXiv: 2102.12024 [hep-th].

[DT08] S. Dasgupta and J. Teitelbaum. "The p-adic upper half plane". English. In: *p-adic geometry. Lectures from the 2007 10th Arizona winter school, Tucson, AZ, USA, March 10–14, 2007*. Providence, RI: American Mathematical Society (AMS), 2008, pages 65–121. ISBN: 978-0-8218-4468-7.

[FFC24] M. I. de Frutos-Fernández and F. A. E. Nuccio Mortarino Majno di Capriglio. "A Formalization of Complete Discrete Valuation Rings and Local Fields". In: *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP '24. ACM, Jan. 2024, 190–204. DOI: 10.1145/3636501.3636942. URL: http://dx.doi.org/10.1145/3636501.3636942.

[KP23] T. Kaletha and G. Prasad. *Bruhat-Tits theory. A new approach*. English. Volume 44. New Math. Monogr. Cambridge: Cambridge University Press, 2023. ISBN: 978-1-108-83196-3; 978-1-108-93304-9. DOI: 10.1017/9781108933049.

[Liv23] A. Livingston. "Group cohomology in the Lean community library". English. In: *14th international conference on interactive theorem proving, ITP 2023, Białystok, Poland, July 31 – August 4, 2023*. Id/No 22. Wadern: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, page 17. ISBN: 978-3-95977-284-6. DOI: 10.4230/LIPIcs.ITP.2023.22.

[mat20] The mathlib Community. “The Lean mathematical library”. In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2020. New Orleans, LA, USA: Association for Computing Machinery, 2020, 367–381. ISBN: 9781450370974. DOI: 10.1145/3372885.3373824. URL: https://doi.org/10.1145/3372885.3373824.

[MU21] L. d. Moura and S. Ullrich. “The Lean 4 Theorem Prover and Programming Language”. In: *Automated Deduction – CADE 28*. Edited by A. Platzer and G. Sutcliffe. Cham: Springer International Publishing, 2021, pages 625–635. ISBN: 978-3-030-79876-5. DOI: 10.1007/978-3-030-79876-5_37.

[Put92] M. van der Put. “Discrete groups, Mumford curves and theta functions”. English. In: *Ann. Fac. Sci. Toulouse, Math. (6)* 1.3 (1992), pages 399–438. ISSN: 0240-2963. DOI: 10.5802/afst.754. URL: https://eudml.org/doc/73309.

[Put83] M. van der Put. *Les fonctions theta d’une courbe de Mumford*. French. Groupe Etude Anal. Ultrametrique, 9e Annee: 1981/82, No. 1, Expose No. 10, 12 p. (1983). 1983. URL: https://eudml.org/doc/91869.

[Ser03] J.-P. Serre. *Trees. Transl. from the French by John Stillwell.* English. Corrected 2nd printing of the 1980 original. Springer Monogr. Math. Berlin: Springer, 2003. ISBN: 3-540-44237-5. DOI: 10.1007/978-3-642-61856-7.

[Tei91] J. T. Teitelbaum. “The Poisson kernel for Drinfeld modular curves”. English. In: *J. Am. Math. Soc.* 4.3 (1991), pages 491–511. ISSN: 0894-0347. DOI: 10.2307/2939266.

[Tei90] J. T. Teitelbaum. “Values of p-adic L-functions and a p-adic Poisson kernel”. English. In: *Invent. Math.* 101.2 (1990), pages 395–410. ISSN: 0020-9910. DOI: 10.1007/BF01231508. URL: https://eudml.org/doc/143809.

[Zab89] A. V. Zabrodin. “Non-Archimedean strings and Bruhat-Tits trees”. English. In: *Commun. Math. Phys.* 123.3 (1989), pages 463–483. ISSN: 0010-3616. DOI: 10.1007/BF01238811.