UNIVERSITÄT
BAYREUTH

Formalizing Heights in Arithmetic Geometry

Michael Stoll
Universität Bayreuth

SMS Spring Meeting:
Formalization and Proof Assistants
FernUni Schweiz, Brig
March 26, 2026

Motivation

Submissions to:
Number Theory
received from Mon 23 Mar 26 18:00:00 GMT
to Tue 24 Mar 26 18:00:00 GMT

Title: Counting points in thin sets: A survey

Abstract: In the 1980's Serre asked how many points of bounded height can lie in a thin set. . . .

Title: Uniform boundedness of small points on abelian varieties over function fields

Title: Points of bounded height in images of morphisms of weighted projective stacks with applications to counting elliptic curves

Title: A naive $\mathfrak{p}$-adic height on the Jacobians of curves of genus 2

Outline

- Explain what heights are
- Discuss design decisions for the formalization
- Show some Lean code (one slide only)
- What is in Mathlib, what is next, outlook

Heights (1)

Let K be an algebraic number field*.

The height on K measures the arithmetic complexity of an element of K or a K-point on projective space $\mathbb{P}^n$ over K.

Let $x \in \mathbb{P}^n(\mathbb{Q})$.
Then there is a coordinate vector $(x_0, x_1, \ldots, x_n)$ for x
whose entries are coprime integers.
Then we define the multiplicative height as

$$H(x) = \max\{|x_0|, |x_1|, \ldots, |x_n|\}$$

and the logarithmic height as $h(x) = \log H(x)$.

Problem: This does not generalize to K —

There is a notion of integers, but in general no gcd.

* the theory applies also to function fields

Heights (2)

To define a height on K in general, we need

- a family $(v_i)_{i\in I}$ of (weak) absolute values on K

such that

❶ v_i is nonarchimedean* for all but finitely many i

❷ $\forall x \neq 0$: $v_i(x) = 1$ for all but finitely many i

❸ $\forall x \neq 0$: $\prod_i v_i(x) = 1$ (Product Formula)

We then define for $x = (x_0 : x_1 : \ldots : x_n) \in \mathbb{P}^n(K)$

$$H(x) = \prod_i \max\{v_i(x_0), v_i(x_1), \ldots, v_i(x_n)\}, \qquad h(x) = \log H(x).$$

By ❷, this makes sense, and by ❸, it is well-defined.

For $\xi \in K$ we then set $\quad H(\xi) = H\Big((\xi : 1)\Big), \quad h(\xi) = \log H(\xi).$

* $v_i(x+y) \leq \max\{v_i(x), v_i(y)\}$

Heights (3)

For $K = \mathbb{Q}$, we have

- the usual absolute value $v_\infty \colon x \mapsto |x|$
- for each prime p, the p-adic absolute value v_p*

Together, they satisfy the conditions ❶–❸,

and we recover the "naïve" definition of the height.

Example.

$$H(3/2) = H\Big((3/2 : 1)\Big) = \max\{|3/2|, 1\} \cdot \max\{2, 1\} \cdot \max\{1/3, 1\} = 3$$

Number fields satisfy the Northcott property:

The set of elements of K / points in $\mathbb{P}^n(K)$ with bounded height is finite.

* $v_p(x)$ is small when the numerator of x is divisible by a high power of p

Relevance of Heights

Heights allow a precise description of how dense or sparse
K-rational points are on an algebraic variety over K.
(Count points of bounded height as the bound goes to infinity.)

Heights are an indispensable tool in many proofs of important results:

- The Mordell-Weil Theorem
  (the group of K-points on an abelian variety over K is finitely generated)
- The Bombieri-Vojta proof of Faltings's Theorem (Mordell's Conjecture);
  recent uniformity results for K-points on curves
- We need heights for the statement of diophantine approximation results
  over general algebraic number fields.

Conclusion: It is highly desirable to have heights available in Mathlib!

Setup for Formalization

We want to set up heights in the level of generality sketched above.

So we need to formalize the family of absolute values
with the relevant properties.

This is done via a class `AdmissibleAbsValues`; results will be valid
assuming `{K : Type*} [Field K] [AdmissibleAbsValues K]`.

We can then develop the general theory of heights (definitions and API).

We need to provide instances for number fields (and function fields).

There is a class `Northcott` (as of earlier this month)
for which we need to provide an instance for heights on number fields.

Design Decisions

Design Decisions

❶ Weak absolute values* or absolute values with multiplicities?

* $\nu(x+y) \leq C \max\{\nu(x), \nu(y)\}$ for some C

Design Decisions

❶ Weak absolute values* or absolute values with multiplicities?

- ✔ No need for multiplicities: powers of weak abs. values are again weak abs. values
- ✘ weak abs. values not available as a type in Mathlib
  $\rightarrow$ needs API, most of which is essentially duplication
- ✘ Working with weak absolute values
  leads to slightly weaker bounds

* $\nu(x+y) \leq C \max\{\nu(x), \nu(y)\}$ for some C

Design Decisions

❶ Weak absolute values* or absolute values with multiplicities?

$\rightsquigarrow$ use standard absolute values

* $\nu(x+y) \leq C \max\{\nu(x), \nu(y)\}$ for some C

Design Decisions

❶ Weak absolute values* or absolute values with multiplicities?

$\rightsquigarrow$ use standard absolute values

❷ One family of absolute values or two (archimedean/nonarch.)?

* $\nu(x+y) \leq C \max\{\nu(x), \nu(y)\}$ for some C

Design Decisions

❶ Weak absolute values* or absolute values with multiplicities?

$\rightsquigarrow$ use standard absolute values

❷ One family of absolute values or two (archimedean/nonarch.)?

- ✔ Reflects the usual definition
- ✔ Gives simpler expressions
- ✘ Very painful to set up the number field instance
- ✔ In proofs, we frequently need to separate archimedean and nonarchimedean absolute values (plus use that there are only finitely many archimedean ones)
- ✔ Multiplicities are needed only for archimedean absolute values (powers of nonarchimedean ones are again absolute values)

* $v(x+y) \leq C \max\{v(x), v(y)\}$ for some C

Design Decisions

❶ Weak absolute values* or absolute values with multiplicities?

$\rightsquigarrow$ use standard absolute values

❷ One family of absolute values or two (archimedean/nonarch.)?

$\rightsquigarrow$ use two families

* $\nu(x+y) \leq C \max\{\nu(x), \nu(y)\}$ for some C

Design Decisions

❶ Weak absolute values* or absolute values with multiplicities?

$\rightsquigarrow$ use standard absolute values

❷ One family of absolute values or two (archimedean/nonarch.)?

$\rightsquigarrow$ use two families

❸ Indexing types or sets/multisets/finsets of absolute values?

* $v(x+y) \leq C \max\{v(x), v(y)\}$ for some C

Design Decisions

❶ Weak absolute values* or absolute values with multiplicities?

$\rightsquigarrow$ use standard absolute values

❷ One family of absolute values or two (archimedean/nonarch.)?

$\rightsquigarrow$ use two families

❸ Indexing types or sets/multisets/finsets of absolute values?

- ✔ More elegant code (e.g., number field instance)
- ✔ We need to use `finprod`, which is defined for indexing types
- ✘ Types inside classes are unusual in Mathlib
- ✘ Several types must live in the same universe

* $v(x+y) \leq C \max\{v(x), v(y)\}$ for some C

Design Decisions

❶ Weak absolute values* or absolute values with multiplicities?

$\rightsquigarrow$ use standard absolute values

❷ One family of absolute values or two (archimedean/nonarch.)?

$\rightsquigarrow$ use two families

❸ Indexing types or sets/multisets/finsets of absolute values?

$\rightsquigarrow$ use a multiset of archimedean absolute values
(taking care of multiplicities) and a set of nonarchimedean ones

* $\nu(x+y) \leq C \max\{\nu(x), \nu(y)\}$ for some C

Design Decisions

❶ Weak absolute values* or absolute values with multiplicities?

$\rightsquigarrow$ use standard absolute values

❷ One family of absolute values or two (archimedean/nonarch.)?

$\rightsquigarrow$ use two families

❸ Indexing types or sets/multisets/finsets of absolute values?

$\rightsquigarrow$ use a multiset of archimedean absolute values
(taking care of multiplicities) and a set of nonarchimedean ones

❹ Absolute values with values in $\mathbb{R}$ or in $\mathbb{R}_{\geq 0}$?

* $\nu(x+y) \leq C \max\{\nu(x), \nu(y)\}$ for some C

Design Decisions

❶ Weak absolute values* or absolute values with multiplicities?

⇝ use standard absolute values

❷ One family of absolute values or two (archimedean/nonarch.)?

⇝ use two families

❸ Indexing types or sets/multisets/finsets of absolute values?

⇝ use a multiset of archimedean absolute values
(taking care of multiplicities) and a set of nonarchimedean ones

❹ Absolute values with values in $\mathbb{R}$ or in $\mathbb{R}_{\geq 0}$?

✔ Places of number fields are abs. values with values in $\mathbb{R}$

✘ Need to work with a subtype

✔ Nonnegativity side goals are automatic

Design Decisions

❶ Weak absolute values* or absolute values with multiplicities?

$\rightsquigarrow$ use standard absolute values

❷ One family of absolute values or two (archimedean/nonarch.)?

$\rightsquigarrow$ use two families

❸ Indexing types or sets/multisets/finsets of absolute values?

$\rightsquigarrow$ use a multiset of archimedean absolute values
(taking care of multiplicities) and a set of nonarchimedean ones

❹ Absolute values with values in $\mathbb{R}$ or in $\mathbb{R}_{\geq 0}$?

$\rightsquigarrow$ use $v_i \colon K \to \mathbb{R}$

* $v(x+y) \leq C \max\{v(x), v(y)\}$ for some C

The Resulting Definition

```
/-- A type class capturing an admissible family of absolute values. -/
class AdmissibleAbsValues (K : Type*) [Field K] where
  /-- The archimedean absolute values as a multiset of `ℝ`-valued absolute values on `K`. -/
  archAbsVal : Multiset (AbsoluteValue K ℝ)
  /-- The nonarchimedean absolute values as a set of `ℝ`-valued absolute values on `K`. -/
  nonarchAbsVal : Set (AbsoluteValue K ℝ)
  /-- The nonarchimedean absolute values are indeed nonarchimedean. -/
  isNonarchimedean : ∀ v ∈ nonarchAbsVal, IsNonarchimedean v
  /-- Only finitely many (nonarchimedean) absolute values are `≠ 1` for any nonzero `x : K`. -/
  hasFiniteMulSupport {x : K} (_ : x ≠ 0) : (fun v : nonarchAbsVal ↦ v.val x).HasFiniteMulSupport
  /-- The product formula. The archimedean absolute values are taken with their multiplicity. -/
  product_formula {x : K} (_ : x ≠ 0) :
      (archAbsVal.map (· x)).prod * ∏ᶠ v : nonarchAbsVal, v.val x = 1
```

⋮

```
/-- The multiplicative height of a tuple of elements of `K`.
For the zero tuple we take the junk value `1`. -/
def mulHeight (x : ι → K) : ℝ :=
  have : Decidable (x = 0) := Classical.propDecidable _
  if x = 0 then 1 else
    (archAbsVal.map fun v ↦ ⨆ i, v (x i)).prod * ∏ᶠ v : nonarchAbsVal, ⨆ i, v.val (x i)
```

What is in Mathlib?

So far, the following have made it into Mathlib.

- The basic theory (definitions for field elements and tuples, API)
  - `mulHeight`$_1$/`logHeight`$_1$: mult./log. height of field elements
  - `mulHeight`/`logHeight`: mult./log. height of tuples
- Bounds / equalities for powers, products, sums, "Segre embedding", linear maps, polynomial maps (upper and lower bounds)
- The `AdmissibleAbsValues` instance for algebraic number fields

This code can be found in the files under `Mathlib/NumberTheory/Height/`.

Work in Progress

For the following, code exists, and the PR process is ongoing.

- The Northcott property for number fields.

  This involves extending the group theory and ring theory API.

- Results on the naïve height on an elliptic curve

  (this will be needed for Mordell-Weil for elliptic curves)

Future Work

There is a lot that still needs to be done.

- Set up `AdmissibleAbsValues` for function fields
  and prove the Northcott property for function fields over finite fields
- Define absolute heights and establish their properties
- Define canonical heights and establish their properties
- Translate the existing results into the language of algebraic geometry
  - Line bundles / divisors $\rightsquigarrow$ projective embdeddings $\rightsquigarrow$ heights
  - Functoriality, Weil's height machine
- Applications

Thank You!