Definition and Elementary Properties of Ultrametric Spaces

Benoît Ballenghien[*] Benjamin Puyobro[†] Burkhart Wolff[‡]

February 7, 2026

Abstract

An ultrametric space is a metric space in which the triangle inequality is strengthened by using the maximum instead of the sum. More formally, such a space is equipped with a real-valued application *dist*, called distance, verifying the four following conditions.

$$dist\ x\ y \geq 0$$
$$dist\ x\ y = dist\ y\ x$$
$$dist\ x\ y = 0 \longleftrightarrow x = y$$
$$dist\ x\ z \leq max\ (dist\ x\ y)\ (dist\ y\ z)$$

In this entry, we present an elementary formalization of these spaces relying on axiomatic type classes. The connection with standard metric spaces is obtained through a subclass relationship, and fundamental properties of ultrametric spaces are formally established.

---

[*] Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF
https://orcid.org/0009-0000-4941-187X

[†] Université Paris-Saclay, IRT SystemX, CNRS, ENS Paris-Saclay, LMF
https://orcid.org/0009-0006-0294-9043

This work has been supported by the French government under the “France 2030” program, as part of the SystemX Technological Research Institute within the CVH project.

[‡] Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF
https://orcid.org/0000-0002-9648-7663

Contents

1 Definition 2

2 Properties on Balls 2
- 2.1 Balls are centered everywhere 3
- 2.2 Balls are “clopen” 3
- 2.3 Balls are disjoint or contained 4
- 2.4 Distance to a Ball 6

3 Additional Properties 7
- 3.1 Cauchy Sequences 7
- 3.2 Isosceles Triangle Principle 9
- 3.3 Distance to a convergent Sequence 9
- 3.4 Diameter 9
- 3.5 Totally disconnected 10

1 Definition

```
setup ‹Sign.add-const-constraint (const-name ‹dist›, NONE)›
  — To be able to use dist out of the metric-space class.

class ultrametric-space = uniformity-dist + open-uniformity +
  assumes dist-eq-0-iff [simp]: ‹dist x y = 0 ⟷ x = y›
    and ultrametric-dist-triangle2: ‹dist x y ≤ max (dist x z) (dist y
z)›
begin

subclass metric-space
proof (unfold-locales)
  show ‹dist x y = 0 ⟷ x = y› for x y by simp
next
  show ‹dist x y ≤ dist x z + dist y z› for x y z
    by (rule order-trans[OF ultrametric-dist-triangle2, of - z], simp)
      (metis local.dist-eq-0-iff ultrametric-dist-triangle2 max.idem)
qed

end

setup ‹Sign.add-const-constraint (const-name ‹dist›, SOME typ ‹'a
:: metric-space ⇒ 'a ⇒ real›)›
  — Back to normal.


class complete-ultrametric-space = ultrametric-space +
  assumes Cauchy-convergent: ‹Cauchy X ⟹ convergent X›
```

```
begin

subclass complete-space by unfold-locales (fact Cauchy-convergent)

end
```

2 Properties on Balls

In ultrametric space, balls satisfy very strong properties.

```
context ultrametric-space begin

lemma ultrametric-dist-triangle: ‹dist x z ≤ max (dist x y) (dist y z)›
 using ultrametric-dist-triangle2 [of x z y] by (simp add: dist-commute)

lemma ultrametric-dist-triangle3: ‹dist x y ≤ max (dist a x) (dist a y)›
 using ultrametric-dist-triangle2 [of x y a] by (simp add: dist-commute)

end
```

2.1 Balls are centered everywhere

```
context fixes x :: ‹'a :: ultrametric-space› begin
```

The best way to do this would be to work in the context *ultrametric-space*. Unfortunately, *ball*, *cball*, etc. are not defined inside the context *metric-space* but through a sort constraint.

```
lemma ultrametric-every-point-of-ball-is-centre :
  ‹ball y r = ball x r› if ‹y ∈ ball x r›
proof (unfold set-eq-iff, rule allI)
  from ‹y ∈ ball x r› have * : ‹dist x y < r› by simp
  show ‹z ∈ ball y r ⟷ z ∈ ball x r› for z
    using ultrametric-dist-triangle[of x z y]
      * ultrametric-dist-triangle[of y z x]
    by (intro iffI) (simp-all add: dist-commute)
qed


lemma ultrametric-every-point-of-cball-is-centre :
  ‹cball y r = cball x r› if ‹y ∈ cball x r›
proof (unfold set-eq-iff, rule allI)
  from ‹y ∈ cball x r› have * : ‹dist x y ≤ r› by simp
  show ‹z ∈ cball y r ⟷ z ∈ cball x r› for z
    using ultrametric-dist-triangle[of x z y]
      * ultrametric-dist-triangle[of y z x]
    by (intro iffI) (simp-all add: dist-commute)
qed
```

```
end
```

2.2 Balls are “clopen”

Balls are both open and closed.

```
context fixes x :: ‹'a :: ultrametric-space› begin

lemma ultrametric-open-cball [intro, simp] : ‹open (cball x r)› if ‹0
< r›
proof (rule openI)
  fix y assume ‹y ∈ cball x r›
  hence ‹cball y r = cball x r›
    by (rule ultrametric-every-point-of-cball-is-centre)
  hence ‹ball y (r / 2) ⊆ cball x r›
   by (metis ball-subset-cball cball-divide-subset-numeral subset-trans)
  moreover have ‹0 < r / 2› by (simp add: ‹0 < r›)
  ultimately show ‹∃ e>0. ball y e ⊆ cball x r› by blast
qed

lemma ‹closed (cball y r)› by (fact closed-cball)


lemma ultrametric-closed-ball [intro, simp]: ‹closed (ball x r)› if ‹0
≤ r›
proof (cases ‹r = 0›)
  show ‹r = 0 ⟹ closed (ball x r)› by simp
next
  assume ‹r ≠ 0›
  with ‹0 ≤ r› have ‹0 < r› by simp
  show ‹closed (ball x r)›
  proof (unfold closed-def)
    have ‹− ball x r = (⋃ z ∈ − ball x r. ball z r)›
    proof (intro subset-antisym subsetI)
     from ‹0 < r› show ‹z ∈ − ball x r ⟹ z ∈ (⋃ z∈− ball x r. ball
z r)› for z
        by (meson UN-iff centre-in-ball)
    next
      show ‹z ∈ (⋃ z ∈ − ball x r. ball z r) ⟹ z ∈ − ball x r› for z
        by simp (metis ComplD dist-commute mem-ball
            ultrametric-every-point-of-ball-is-centre)
    qed
   show ‹open (− ball x r)› by (subst ‹?this›, rule open-Union, simp)
  qed
qed

lemma ultrametric-open-sphere [intro, simp] : ‹0 < r ⟹ open (sphere
x r)›
  by (fold cball-diff-eq-sphere) (simp add: open-Diff order-le-less)
```

lemma *closed-sphere* [*intro*, *simp*] : ‹*closed* (*sphere y r*)›
by (*metis open-ball cball-diff-eq-sphere closed-Diff closed-cball*)

end

2.3 Balls are disjoint or contained

context fixes *x* :: ‹′*a* :: *ultrametric-space*› **begin**

lemma *ultrametric-ball-ball-disjoint-or-subset*:
‹*ball x r* ∩ *ball y s* = {} ∨ *ball x r* ⊆ *ball y s* ∨
ball y s ⊆ *ball x r*›
proof (*unfold disj-imp*, *intro impI*)
assume ‹*ball x r* ∩ *ball y s* ≠ {}› ‹¬ *ball x r* ⊆ *ball y s*›
from ‹*ball x r* ∩ *ball y s* ≠ {}›
obtain *z* **where** ‹*z* ∈ *ball x r*› **and** ‹*z* ∈ *ball y s*› **by** *blast*
with *ultrametric-every-point-of-ball-is-centre*
have ‹*ball x r* = *ball z r*› **and** ‹*ball y s* = *ball z s*› **by** *auto*
with ‹¬ *ball x r* ⊆ *ball y s*› **have** ‹*s* < *r*› **by** *auto*
with ‹*ball x r* = *ball z r*› **and** ‹*ball y s* = *ball z s*›
show ‹*ball y s* ⊆ *ball x r*› **by** *auto*
qed

lemma *ultrametric-ball-cball-disjoint-or-subset*:
‹*ball x r* ∩ *cball y s* = {} ∨ *ball x r* ⊆ *cball y s* ∨
cball y s ⊆ *ball x r*›
proof (*unfold disj-imp*, *intro impI*)
assume ‹*ball x r* ∩ *cball y s* ≠ {}› ‹¬ *ball x r* ⊆ *cball y s*›
from ‹*ball x r* ∩ *cball y s* ≠ {}›
obtain *z* **where** ‹*z* ∈ *ball x r*› **and** ‹*z* ∈ *cball y s*› **by** *blast*
with *ultrametric-every-point-of-ball-is-centre*
ultrametric-every-point-of-cball-is-centre
have ‹*ball x r* = *ball z r*› ‹*cball y s* = *cball z s*› **by** *blast+*
with ‹¬ *ball x r* ⊆ *cball y s*› **have** ‹*s* < *r*› **by** *auto*
with ‹*ball x r* = *ball z r*› **and** ‹*cball y s* = *cball z s*›
show ‹*cball y s* ⊆ *ball x r*› **by** *auto*
qed

corollary *ultrametric-cball-ball-disjoint-or-subset*:
‹*cball x r* ∩ *ball y s* = {} ∨ *cball x r* ⊆ *ball y s* ∨
ball y s ⊆ *cball x r*›
using *Elementary-Ultrametric-Spaces.ultrametric-ball-cball-disjoint-or-subset*
by *blast*

lemma *ultrametric-cball-cball-disjoint-or-subset*:
‹*cball x r* ∩ *cball y s* = {} ∨ *cball x r* ⊆ *cball y s* ∨
cball y s ⊆ *cball x r*›
proof (*unfold disj-imp*, *intro impI*)
assume ‹*cball x r* ∩ *cball y s* ≠ {}› ‹¬ *cball x r* ⊆ *cball y s*›

```
  from ‹cball x r ∩ cball y s ≠ {}›
  obtain z where ‹z ∈ cball x r› and ‹z ∈ cball y s› by blast
  with ultrametric-every-point-of-cball-is-centre
  have ‹cball x r = cball z r› ‹cball y s = cball z s› by auto
  with ‹¬ cball x r ⊆ cball y s› have ‹s < r› by auto
  with ‹cball x r = cball z r› and ‹cball y s = cball z s›
  show ‹cball y s ⊆ cball x r› by auto
qed

end
```

2.4 Distance to a Ball

```
context fixes a :: ‹'a :: ultrametric-space› begin

lemma ultrametric-equal-distance-to-ball:
  ‹dist a y = dist a z› if ‹a ∉ ball x r› ‹y ∈ ball x r› ‹z ∈ ball x r›
proof (rule order-antisym)
  show ‹dist a y ≤ dist a z›
    by (rule order-trans[OF ultrametric-dist-triangle[of a y z]], simp)
      (metis dist-commute dual-order.strict-trans2 linorder-linear
        mem-ball that ultrametric-every-point-of-ball-is-centre)
next
  show ‹dist a z ≤ dist a y›
    by (rule order-trans[OF ultrametric-dist-triangle[of a z y]], simp)
      (metis dist-commute dual-order.strict-trans2 linorder-linear
        mem-ball that ultrametric-every-point-of-ball-is-centre)
qed


lemma ultrametric-equal-distance-to-cball:
  ‹dist a y = dist a z› if ‹a ∉ cball x r› ‹y ∈ cball x r› ‹z ∈ cball x r›
proof (rule order-antisym)
  show ‹dist a y ≤ dist a z›
    by (rule order-trans[OF ultrametric-dist-triangle[of a y z]], simp)
      (metis dist-commute dual-order.trans linorder-linear mem-cball
        that ultrametric-every-point-of-cball-is-centre)
next
  show ‹dist a z ≤ dist a y›
    by (rule order-trans[OF ultrametric-dist-triangle[of a z y]], simp)
      (metis dist-commute dual-order.trans linorder-linear mem-cball
        that ultrametric-every-point-of-cball-is-centre)
qed

end


context fixes x :: ‹'a :: ultrametric-space› begin
```

```
lemma ultrametric-equal-distance-between-ball-ball:
  ‹ball x r ∩ ball y s = {} ⟹
   ∃ d. ∀ a ∈ ball x r. ∀ b ∈ ball y s. dist a b = d›
 by (metis disjoint-iff dist-commute ultrametric-equal-distance-to-ball)


lemma ultrametric-equal-distance-between-ball-cball:
  ‹ball x r ∩ cball y s = {} ⟹
   ∃ d. ∀ a ∈ ball x r. ∀ b ∈ cball y s. dist a b = d›
  by (metis disjoint-iff dist-commute ultrametric-equal-distance-to-ball
      ultrametric-equal-distance-to-cball)

lemma ultrametric-equal-distance-between-cball-ball:
  ‹cball x r ∩ ball y s = {} ⟹
   ∃ d. ∀ a ∈ cball x r. ∀ b ∈ ball y s. dist a b = d›
 by (metis disjoint-iff-not-equal dist-commute ultrametric-equal-distance-to-ball
      ultrametric-equal-distance-to-cball)

lemma ultrametric-equal-distance-between-cball-cball:
  ‹cball x r ∩ cball y s = {} ⟹
   ∃ d. ∀ a ∈ cball x r. ∀ b ∈ cball y s. dist a b = d›
 by (metis disjoint-iff dist-commute ultrametric-equal-distance-to-cball)

end
```

3 Additional Properties

Here are a few other interesting properties.

3.1 Cauchy Sequences

```
lemma (in ultrametric-space) ultrametric-dist-triangle-generalized:
  ‹n < m ⟹ dist (σ n) (σ m) ≤ (MAX l ∈ {n..m − 1}. dist (σ l)
(σ (Suc l)))›
proof (induct m)
  show ‹n < 0 ⟹ dist (σ n) (σ 0) ≤ (MAX l∈{n..0 − 1}. dist (σ
l) (σ (Suc l)))› by simp
next
  case (Suc m)
  show ‹dist (σ n) (σ (Suc m)) ≤ (MAX l∈{n..Suc m − 1}. dist (σ
l) (σ (Suc l)))›
  proof (cases ‹n = m›)
    show ‹n = m ⟹ dist (σ n) (σ (Suc m)) ≤ (MAX l∈{n..Suc m
− 1}. dist (σ l) (σ (Suc l)))›
      by simp
  next
    assume ‹n ≠ m›
    with ‹n < Suc m› obtain m' where ‹m = Suc m'› ‹n ≤ m'›
```

by (*metis le-add1 less-Suc-eq less-imp-Suc-add*)
have ‹{*n*..*Suc m'*} = {*n*..*m*−*1*} ∪ {*m*}›
by (*simp add*: ‹*m* = *Suc m'*› ‹*n* ≤ *m'*› *atLeastAtMostSuc-conv le-Suc-eq*)
have ‹*dist* (σ *n*) (σ (*Suc m*)) ≤ *max* (*dist* (σ *n*) (σ *m*)) (*dist* (σ *m*) (σ (*Suc m*)))›
by (*simp add*: *ultrametric-dist-triangle*)
also have ‹. . . ≤ *max* ((*MAX l*∈{*n*..*m* − *1*}. *dist* (σ *l*) (σ (*Suc l*)))) (*dist* (σ *m*) (σ (*Suc m*)))›
using *Suc.hyps Suc.prems* ‹*n* ≠ *m*› **by** *linarith*
also have ‹. . . = (*MAX l*∈{*n*..*Suc m* − *1*}. *dist* (σ *l*) (σ (*Suc l*)))›
by (*subst Max-Un*[*of* - ‹((λ*l*. *dist* (σ *l*) (σ (*Suc l*))) ‘ {*m*})›, *simplified*, *symmetric*])
(*simp-all add*: ‹*m* = *Suc m'*› ‹*n* ≤ *m'*› ‹{*n*..*Suc m'*} = {*n*..*m*−*1*} ∪ {*m*}›)
finally show ‹*dist* (σ *n*) (σ (*Suc m*)) ≤ (*MAX l*∈{*n*..*Suc m* − *1*}. *dist* (σ *l*) (σ (*Suc l*)))› **.**
qed
qed

lemma (**in** *ultrametric-space*) *ultrametric-Cauchy-iff*:
‹*Cauchy* σ ⟷ (λ*n*. *dist* (σ (*Suc n*)) (σ *n*)) ⟶ *0*›
proof (*rule iffI*)
assume ‹*Cauchy* σ›
show ‹(λ*n*. *dist* (σ (*Suc n*)) (σ *n*)) ⟶ *0*›
proof (*unfold lim-sequentially*, *intro allI impI*)
fix ε :: *real*
assume ‹*0* < ε›
from ‹*Cauchy* σ›[*unfolded Cauchy-altdef*, *rule-format*, *OF* ‹*0* < ε›]
show ‹∃ *N*. ∀ *n*≥*N*. *dist* (*dist* (σ (*Suc n*)) (σ *n*)) *0* < ε›
by (*auto simp add*: *dist-commute*)
qed
next
assume *convergent* : ‹(λ*n*. *dist* (σ (*Suc n*)) (σ *n*)) ⟶ *0*›
show ‹*Cauchy* σ›
proof (*unfold Cauchy-altdef2*, *intro allI impI*)
fix ε :: *real*
assume ‹*0* < ε›
from *convergent*[*unfolded lim-sequentially*, *rule-format*, *OF* ‹*0* < ε›]
obtain *N* **where** ∗ : ‹*N* ≤ *n* ⟹ *dist* (σ (*Suc n*)) (σ *n*) < ε› **for** *n*
by (*simp add*: *dist-real-def*) *blast*

have ‹*N* < *n* ⟹ *dist* (σ *n*) (σ *N*) < ε› **for** *n*
proof (*subst dist-commute*, *rule le-less-trans*)
show ‹*N* < *n* ⟹ *dist* (σ *N*) (σ *n*) ≤ (*MAX l*∈{*N*..*n* − *1*}. *dist*

```
(σ l) (σ (Suc l)))›
        by (fact ultrametric-dist-triangle-generalized)
    next
      show ‹N < n ⟹ (MAX l∈{N..n − 1}. dist (σ l) (σ (Suc l)))
< ε›
        by simp (metis ∗ atLeastAtMost-iff dist-commute)
    qed
    with ‹0 < ε› have ‹N ≤ n ⟹ dist (σ n) (σ N) < ε› for n
      by (cases ‹N = n›) simp-all
    thus ‹∃N. ∀n≥N. dist (σ n) (σ N) < ε› by blast
  qed
qed
```

3.2 Isosceles Triangle Principle

```
lemma (in ultrametric-space) ultrametric-isosceles-triangle-principle
:
  ‹dist x z = max (dist x y) (dist y z)› if ‹dist x y ≠ dist y z›
proof (rule order-antisym)
  show ‹dist x z ≤ max (dist x y) (dist y z)›
    by (fact ultrametric-dist-triangle)
next
  from ‹dist x y ≠ dist y z› linorder-less-linear
  have ‹dist x y < dist y z ∨ dist y z < dist x y› by blast
  with ultrametric-dist-triangle[of y z x]
    ultrametric-dist-triangle[of x y z]
  show ‹max (dist x y) (dist y z) ≤ dist x z›
    by (elim disjE) (simp-all add: dist-commute)
qed
```

3.3 Distance to a convergent Sequence

```
lemma ultrametric-dist-to-convergent-sequence-is-eventually-const :
  fixes σ :: ‹nat ⇒ 'a :: ultrametric-space›
  assumes ‹σ ⟶ Σ› and ‹x ≠ Σ›
  shows ‹∃N. ∀n≥N. dist (σ n) x = dist Σ x›
proof −
  from ‹x ≠ Σ› have ‹0 < dist x Σ› by simp
  then obtain ε where ‹0 < ε› ‹ball x ε ∩ ball Σ ε = {}›
    by (metis centre-in-ball disjoint-iff mem-ball order-less-le
        ultrametric-every-point-of-ball-is-centre)

  from ‹σ ⟶ Σ› ‹0 < ε› obtain N where ‹N ≤ n ⟹ σ n ∈
ball Σ ε› for n
    by (auto simp add: dist-commute lim-sequentially)
  with ‹0 < ε› ‹ball x ε ∩ ball Σ ε = {}› have ‹N ≤ n ⟹ dist (σ
n) x = dist Σ x› for n
   by (metis centre-in-ball dist-commute ultrametric-equal-distance-between-ball-ball)
  thus ‹∃N. ∀n≥N. dist (σ n) x = dist Σ x› by blast
qed
```

3.4 Diameter

```
lemma ultrametric-diameter : ‹diameter S = (SUP y ∈ S. dist x y)›
  if ‹bounded S› and ‹x ∈ S› for x :: ‹'a :: ultrametric-space›
proof −
  from ‹x ∈ S› have ‹S ≠ {}› by blast
  show ‹diameter S = (SUP y ∈ S. dist x y)›
  proof (rule order-antisym)
    from diameter-bounded-bound[OF ‹bounded S› ‹x ∈ S›]
    have ‹y ∈ S ⟹ dist x y ≤ diameter S› for y .
    thus ‹(SUP y ∈ S. dist x y) ≤ diameter S›
      by (rule cSUP-least[OF ‹S ≠ {}›])
  next
    have ‹bdd-above (dist x ‘ S)›
      by (meson bdd-above.I2 bounded-any-center ‹bounded S›)
    have ‹y ∈ S ⟹ z ∈ S ⟹ dist y z ≤ max (dist x y) (dist x z)›
for y z
      by (metis dist-commute ultrametric-dist-triangle)
    also have ‹y ∈ S ⟹ z ∈ S ⟹
               max (dist x y) (dist x z) ≤ (SUP y ∈ S. dist x y)› for y z
      by (cases ‹dist x y ≤ dist x z›)
        (simp-all add: cSUP-upper2[OF ‹bdd-above (dist x ‘ S)›])
    finally have ∗ : ‹y ∈ S ⟹ z ∈ S ⟹ dist y z ≤ (SUP y ∈ S.
dist x y)› for y z .
    have ‹(SUP (y, z) ∈ S × S. dist y z) ≤ (SUP y ∈ S. dist x y)›
      by (rule cSUP-least) (use ∗ in ‹auto simp add: ‹S ≠ {}››)
    thus ‹diameter S ≤ Sup (dist x ‘ S)›
      by (simp add: diameter-def ‹S ≠ {}›)
  qed
qed
```

3.5 Totally disconnected

```
lemma ultrametric-totally-disconnected :
  ‹∃x. S = {x}› if ‹S ≠ {}› ‹connected S›
for S :: ‹'a :: ultrametric-space set›
proof −
  from ‹S ≠ {}› obtain x where ‹x ∈ S› by blast
  have ‹ball x r ∩ S = {} ∨ − ball x r ∩ S = {}› if ‹0 < r› for r
   by (rule ‹connected S›[unfolded connected-def, simplified, rule-format])
       (simp-all, use order-less-imp-le that ultrametric-closed-ball in
blast)
  with ‹x ∈ S› have ‹0 < r ⟹ − ball x r ∩ S = {}› for r
    by (metis centre-in-ball disjoint-iff)
  hence ‹0 < r ⟹ y ∈ S ⟹ dist x y < r› for r y
    by (auto simp add: disjoint-iff)
  hence ‹y ∈ S ⟹ dist x y = 0› for y
    by (metis dist-self order-less-irrefl zero-less-dist-iff)
  hence ‹y ∈ S ⟹ y = x› for y by simp
  with ‹x ∈ S› show ‹∃x. S = {x}› by blast
```

qed