Providing restriction Spaces with an ultrametric Structure

Benoît Ballenghien Benjamin Puyobro Burkhart Wolff

February 6, 2026

Abstract

We investigate the relationship between restriction spaces and classical metric structures by instantiating the former as ultrametric spaces. This is classically captured by defining the distance as

$$\text{dist } x\ y = \inf_{x{\downarrow}n\ =\ y{\downarrow}n} \left(\frac{1}{2}\right)^n$$

but we actually generalize this perspective by introducing a hierarchy of increasingly refined type classes to systematically relate ultrametric and restriction-based notions. This layered approach enables a precise comparison of structural and topological properties. In the end, our main result establishes that completeness in the sense of restriction spaces coincides with standard metric completeness, thus bridging the gap between `Restriction_Spaces` and Banach's fixed-point theorem established in `HOL-Analysis`.

Contents

1 Definitions on Functions of Metric Space

In this theory, we define the notion of lipschitz map, non-expanding map and contraction map. We also establish correspondences.

1.1 Definitions

1.1.1 Lipschitz Map

This notion is a generalization of contraction map and non-expanding map.

definition *lipschitz-with-on* :: ‹['*a* :: *metric-space* ⇒ '*b* :: *metric-space*, *real*, '*a set*] ⇒ *bool*›
where ‹*lipschitz-with-on f α A* ≡ *0* ≤ *α* ∧ (∀ *x*∈*A*. ∀ *y*∈*A*. *dist* (*f x*) (*f y*) ≤ *α* ∗ *dist x y*)›

abbreviation *lipschitz-with* :: ‹['*a* :: *metric-space* ⇒ '*b* :: *metric-space*, *real*] ⇒ *bool*›
where ‹*lipschitz-with f α* ≡ *lipschitz-with-on f α UNIV*›

lemma *lipschitz-with-onI* :
‹⟦*0* ≤ *α*; ⋀*x y*. ⟦*x* ∈ *A*; *y* ∈ *A*; *x* ≠ *y*; *f x* ≠ *f y*⟧ ⟹ *dist* (*f x*) (*f y*) ≤ *α* ∗ *dist x y*⟧ ⟹
lipschitz-with-on f α A›
⟨*proof*⟩

lemma *lipschitz-withI* :
‹⟦*0* ≤ *α*; ⋀*x y*. *x* ≠ *y* ⟹ *f x* ≠ *f y* ⟹ *dist* (*f x*) (*f y*) ≤ *α* ∗ *dist x y*⟧ ⟹
lipschitz-with f α›
⟨*proof*⟩

lemma *lipschitz-with-onD1* : ‹*lipschitz-with-on f α A* ⟹ *0* ≤ *α*›
⟨*proof*⟩

lemma *lipschitz-withD1* : ‹*lipschitz-with f α* ⟹ *0* ≤ *α*›
⟨*proof*⟩

lemma *lipschitz-with-onD2* :
‹*lipschitz-with-on f α A* ⟹ *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *dist* (*f x*) (*f y*) ≤ *α* ∗ *dist x y*›
⟨*proof*⟩

lemma *lipschitz-withD2* :
‹*lipschitz-with f α* ⟹ *dist* (*f x*) (*f y*) ≤ *α* ∗ *dist x y*›
⟨*proof*⟩

lemma *lipschitz-with-imp-lipschitz-with-on*: ‹*lipschitz-with f α* ⟹ *lipschitz-with-on f α A*›
⟨*proof*⟩

lemma *lipschitz-with-on-imp-lipschitz-with-on-ge* : ‹*lipschitz-with-on f β A*›
if ‹*α* ≤ *β*› **and** ‹*lipschitz-with-on f α A*›
⟨*proof*⟩

theorem *lipschitz-with-on-comp-lipschitz-with-on* :
‹*lipschitz-with-on* (λ*x*. *g* (*f x*)) (*β* ∗ *α*) *A*›
if ‹*f* ‘ *A* ⊆ *B*› ‹*lipschitz-with-on g β B*› ‹*lipschitz-with-on f α A*›
⟨*proof*⟩

corollary *lipschitz-with-comp-lipschitz-with* :
‹⟦*lipschitz-with g β*; *lipschitz-with f α*⟧ ⟹
lipschitz-with (λ*x*. *g* (*f x*)) (*β* ∗ *α*)›
⟨*proof*⟩

1.1.2 Non-expanding Map

definition *non-expanding-on* :: ‹[′*a* :: *metric-space* ⇒ ′*b* :: *metric-space*, ′*a set*] ⇒ *bool*›
where ‹*non-expanding-on f A* ≡ *lipschitz-with-on f 1 A*›

abbreviation *non-expanding* :: ‹[′*a* :: *metric-space* ⇒ ′*b* :: *metric-space*] ⇒ *bool*›
where ‹*non-expanding f* ≡ *non-expanding-on f UNIV*›

lemma *non-expanding-onI* :
‹⟦⋀*x y*. ⟦*x* ∈ *A*; *y* ∈ *A*; *x* ≠ *y*; *f x* ≠ *f y*⟧ ⟹ *dist* (*f x*) (*f y*) ≤ *dist*

$x\ y\rrbracket \Longrightarrow$
non-expanding-on f A›
⟨*proof*⟩

lemma *non-expandingI* :
‹⟦⋀*x y*. *x* ≠ *y* ⟹ *f x* ≠ *f y* ⟹ *dist* (*f x*) (*f y*) ≤ *dist x y*⟧ ⟹ *non-expanding f*›
⟨*proof*⟩

lemma *non-expanding-onD* :
‹*non-expanding-on f A* ⟹ *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *dist* (*f x*) (*f y*) ≤ *dist x y*›
⟨*proof*⟩

lemma *non-expandingD* : ‹*non-expanding f* ⟹ *dist* (*f x*) (*f y*) ≤ *dist x y*›
⟨*proof*⟩

lemma *non-expanding-imp-non-expanding-on*: ‹*non-expanding f* ⟹ *non-expanding-on f A*›
⟨*proof*⟩

lemma *non-expanding-on-comp-non-expanding-on* :
‹⟦*f* ‘ *A* ⊆ *B*; *non-expanding-on g B*; *non-expanding-on f A*⟧ ⟹
non-expanding-on (λ*x*. *g* (*f x*)) *A*›
⟨*proof*⟩

corollary *non-expanding-comp-non-expanding* :
‹⟦*non-expanding g*; *non-expanding f*⟧ ⟹ *non-expanding* (λ*x*. *g* (*f x*))›
⟨*proof*⟩

1.1.3 Contraction Map

definition *contraction-with-on* :: ‹[′*a* :: *metric-space* ⇒ ′*b* :: *metric-space*, *real*, ′*a set*] ⇒ *bool*›
where ‹*contraction-with-on f* α *A* ≡ α < *1* ∧ *lipschitz-with-on f* α *A*›

abbreviation *contraction-with* :: ‹[′*a* :: *metric-space* ⇒ ′*b* :: *metric-space*, *real*] ⇒ *bool*›
where ‹*contraction-with f* α ≡ *contraction-with-on f* α *UNIV*›

definition *contraction-on* :: ‹[′*a* :: *metric-space* ⇒ ′*b* :: *metric-space*, ′*a set*] ⇒ *bool*›
where ‹*contraction-on f A* ≡ ∃ α. *contraction-with-on f* α *A*›

abbreviation *contraction* :: ‹[′*a* :: *metric-space* ⇒ ′*b* :: *metric-space*]

$\Rightarrow$ *bool*›
where ‹*contraction f* $\equiv$ *contraction-on f UNIV*›

lemma *contraction-with-onI* :
‹⟦*0* $\leq$ α; α < *1*; $\bigwedge$*x y*. ⟦*x* $\in$ *A*; *y* $\in$ *A*; *x* $\neq$ *y*; *f x* $\neq$ *f y*⟧ $\Longrightarrow$ *dist* (*f x*) (*f y*) $\leq$ α * *dist x y*⟧
$\Longrightarrow$ *contraction-with-on f* α *A*›
⟨*proof*⟩

lemma *contraction-withI* :
‹⟦*0* $\leq$ α; α < *1*; $\bigwedge$*x y*. *x* $\neq$ *y* $\Longrightarrow$ *f x* $\neq$ *f y* $\Longrightarrow$ *dist* (*f x*) (*f y*) $\leq$ α * *dist x y*⟧ $\Longrightarrow$
contraction-with f α›
⟨*proof*⟩

lemma *contraction-with-onD1* : ‹*contraction-with-on f* α *A* $\Longrightarrow$ *0* $\leq$ α›
⟨*proof*⟩

lemma *contraction-withD1* : ‹*contraction-with f* α $\Longrightarrow$ *0* $\leq$ α›
⟨*proof*⟩

lemma *contraction-with-onD2* : ‹*contraction-with-on f* α *A* $\Longrightarrow$ α < *1*›
⟨*proof*⟩

lemma *contraction-withD2* : ‹*contraction-with f* α $\Longrightarrow$ α < *1*›
⟨*proof*⟩

lemma *contraction-with-onD3* :
‹*contraction-with-on f* α *A* $\Longrightarrow$ *x* $\in$ *A* $\Longrightarrow$ *y* $\in$ *A* $\Longrightarrow$ *dist* (*f x*) (*f y*) $\leq$ α * *dist x y*›
⟨*proof*⟩

lemma *contraction-withD3* : ‹*contraction-with f* α $\Longrightarrow$ *dist* (*f x*) (*f y*) $\leq$ α * *dist x y*›
⟨*proof*⟩

lemma *contraction-with-imp-contraction-with-on*:
‹*contraction-with f* α $\Longrightarrow$ *contraction-with-on f* α *A*›
⟨*proof*⟩

lemma *contraction-imp-contraction-on*: ‹*contraction f* $\Longrightarrow$ *contraction-on f A*›
⟨*proof*⟩

lemma *contraction-with-on-imp-contraction-on* :

‹*contraction-with-on f α A* ⟹ *contraction-on f A*›
⟨*proof*⟩

lemma *contraction-with-imp-contraction*: ‹*contraction-with f α* ⟹ *contraction f*›
⟨*proof*⟩

lemma *contraction-onE*:
‹⟦*contraction-on f A*; ⋀*α*. *contraction-with-on f α A* ⟹ *thesis*⟧ ⟹ *thesis*›
⟨*proof*⟩

lemma *contractionE*:
‹⟦*contraction f*; ⋀*α*. *contraction-with f α* ⟹ *thesis*⟧ ⟹ *thesis*›
⟨*proof*⟩

lemma *contraction-with-on-imp-contraction-with-on-ge* :
‹⟦*α* ≤ *β*; *β* < *1*; *contraction-with-on f α A*⟧ ⟹ *contraction-with-on f β A*›
⟨*proof*⟩

1.2 Properties

lemma *contraction-with-on-imp-lipschitz-with-on*[*simp*] :
‹*contraction-with-on f α A* ⟹ *lipschitz-with-on f α A*›
⟨*proof*⟩

lemma *non-expanding-on-imp-lipschitz-with-one-on*[*simp*] :
‹*non-expanding-on f A* ⟹ *lipschitz-with-on f 1 A*›
⟨*proof*⟩

lemma *contraction-on-imp-non-expanding-on*[*simp*] :
‹*contraction-on f A* ⟹ *non-expanding-on f A*›
⟨*proof*⟩

lemma *contraction-with-on-comp-contraction-with-on* :
‹*contraction-with-on* (*λx*. *g* (*f x*)) (*β* ∗ *α*) *A*›
if ‹*f* ‘ *A* ⊆ *B*› ‹*contraction-with-on g β B*› ‹*contraction-with-on f α A*›
⟨*proof*⟩

corollary *contraction-with-comp-contraction-with* :
‹⟦*contraction-with g β*; *contraction-with f α*⟧ ⟹ *contraction-with* (*λx*. *g* (*f x*)) (*β* ∗ *α*)›
⟨*proof*⟩

corollary *contraction-on-comp-contraction-on* :
‹⟦*f* ‘ *A* ⊆ *B*; *contraction-on g B*; *contraction-on f A*⟧ ⟹ *contraction-on* (λ*x*. *g* (*f x*)) *A*›
⟨*proof*⟩

corollary *contraction-comp-contraction* :
‹⟦*contraction g*; *contraction f*⟧ ⟹ *contraction* (λ*x*. *g* (*f x*))›
⟨*proof*⟩

lemma *contraction-with-on-comp-non-expanding-on* :
‹*contraction-with-on* (λ*x*. *g* (*f x*)) *β A*›
if ‹*f* ‘ *A* ⊆ *B*› ‹*contraction-with-on g β B*› ‹*non-expanding-on f A*›
⟨*proof*⟩

corollary *contraction-with-comp-non-expanding* :
‹⟦*contraction-with g β*; *non-expanding f*⟧ ⟹ *contraction-with* (λ*x*. *g* (*f x*)) *β*›
⟨*proof*⟩

corollary *contraction-on-comp-non-expanding-on* :
‹⟦*f* ‘ *A* ⊆ *B*; *contraction-on g B*; *non-expanding-on f A*⟧ ⟹ *contraction-on* (λ*x*. *g* (*f x*)) *A*›
⟨*proof*⟩

corollary *contraction-comp-non-expanding* :
‹⟦*contraction g*; *non-expanding f*⟧ ⟹ *contraction* (λ*x*. *g* (*f x*))›
⟨*proof*⟩

lemma *non-expanding-on-comp-contraction-with-on* :
‹*contraction-with-on* (λ*x*. *g* (*f x*)) *α A*›
if ‹*f* ‘ *A* ⊆ *B*› ‹*non-expanding-on g B*› ‹*contraction-with-on f α A*›
⟨*proof*⟩

corollary *non-expanding-comp-contraction-with* :
‹⟦*non-expanding g*; *contraction-with f α*⟧ ⟹ *contraction-with* (λ*x*. *g* (*f x*)) *α*›
⟨*proof*⟩

corollary *non-expanding-on-comp-contraction-on* :
‹⟦*f* ‘ *A* ⊆ *B*; *non-expanding-on g B*; *contraction-on f A*⟧ ⟹ *contraction-on* (λ*x*. *g* (*f x*)) *A*›
⟨*proof*⟩

corollary *non-expanding-comp-contraction* :
‹⟦*non-expanding g*; *contraction f*⟧ ⟹ *contraction* (λ*x*. *g* (*f x*))›
⟨*proof*⟩

1.3 Banach's fixed-point Theorems

We rewrite the Banach's fixed-point theorems with our new definition.

theorem *Banach-fix-type* : ‹*contraction f* $\Longrightarrow \exists!x.\ f\ x = x$›
for $f :: ‹'a :: \textit{complete-space} \Rightarrow 'a›$
⟨*proof*⟩

theorem *Banach-fix*:
‹*contraction-on f s* $\Longrightarrow \exists!x.\ x \in s \wedge f\ x = x$› **if** ‹*complete s*› ‹$s \neq \{\}$› ‹$f\ `\ s \subseteq s$›
⟨*proof*⟩

2 Locales factorizing the proof Work

2.1 Preliminaries on strictly decreasing Sequences

abbreviation *strict-decseq* :: ‹$(\textit{nat} \Rightarrow 'a :: \textit{order}) \Rightarrow \textit{bool}$›
where ‹*strict-decseq* $\equiv$ *monotone* $(<)$ $(\lambda x\ y.\ y < x)$›

lemma *strict-decseq-def* : ‹*strict-decseq X* $\longleftrightarrow$ $(\forall\, m\ n.\ m < n \longrightarrow X\ n < X\ m)$›
⟨*proof*⟩

lemma *strict-decseqI* : ‹*strict-decseq X*› **if** ‹$\bigwedge n.\ X\ (\textit{Suc}\ n) < X\ n$›
⟨*proof*⟩

lemma *strict-decseqD* : ‹*strict-decseq X* $\Longrightarrow m < n \Longrightarrow X\ n < X\ m$›
⟨*proof*⟩

lemma *strict-decseq-def-bis* : ‹*strict-decseq X* $\longleftrightarrow$ $(\forall\, m\ n.\ X\ n < X\ m \longleftrightarrow m < n)$›
⟨*proof*⟩

lemma *strict-decseq-def-ter* : ‹*strict-decseq X* $\longleftrightarrow$ $(\forall\, m\ n.\ X\ n \leq X\ m \longleftrightarrow m \leq n)$›
⟨*proof*⟩

lemma *strict-decseq-imp-decseq* : ‹*strict-decseq* $\sigma \Longrightarrow$ *decseq* σ›
⟨*proof*⟩

lemma *strict-decseq-SucI* : ‹$(\bigwedge n.\ X\ (\textit{Suc}\ n) < X\ n) \Longrightarrow$ *strict-decseq X*›
⟨*proof*⟩

lemma *strict-decseq-SucD* : ‹*strict-decseq A* $\Longrightarrow A\ (\textit{Suc}\ i) < A\ i$›
⟨*proof*⟩

Classically, a restriction space is given the structure of a metric space by defining $dist\ x\ y \equiv Inf\ \{(1\ /\ 2)\ \widehat{}\ n|\ n.\ x \downarrow n = y \downarrow n\}$. This obviously also works if we replace $1\ /\ 2$ by any real δ such that $0 < \delta$ and $\delta < 1$. But more generally, this still works if we set $dist\ x\ y \equiv Inf\ \{\sigma\ n|\ n.\ x \downarrow n = y \downarrow n\}$ where σ is a sequence of *real* verifying $\forall n.\ 0 < \sigma\ n$ and $\sigma \longrightarrow 0$. As you would expect, the more structure you have, the more powerful theorems you get. We explore all these variants in the theory below.

2.2 The Construction with Locales

Our formalization will extend the class *metric-space*. But some proofs are redundant, especially when it comes to the product type. So first we will be working with locales, and interpret them with the classes.

locale *NonDecseqRestrictionSpace* = *PreorderRestrictionSpace* +
— Factorization of the proof work.
fixes M :: ‹$'a$ *set*› **and** *restriction-σ* :: ‹*nat* ⇒ *real*› (‹$\sigma_\downarrow$›)
and *restriction-dist* :: ‹$'a$ ⇒ $'a$ ⇒ *real*› (‹$dist_\downarrow$›)
assumes *restriction-σ-tendsto-zero* : ‹*restriction-σ* ⟶ *0*›
and *zero-less-restriction-σ* [*simp*] : ‹*0* < *restriction-σ n*›
and *dist-restriction-is* :
‹$dist_\downarrow\ x\ y$ = (*INF n* ∈ *restriction-related-set x y*. *restriction-σ n*)›
begin

lemma *zero-le-restriction-σ* [*simp*] : ‹*0* ≤ $\sigma_\downarrow$ *n*›
⟨*proof*⟩

lemma *restriction-σ-neq-zero* [*simp*] : ‹$\sigma_\downarrow$ *n* ≠ *0*›
⟨*proof*⟩

lemma *bounded-range-restriction-σ*: ‹*bounded* (*range* $\sigma_\downarrow$)›
⟨*proof*⟩

abbreviation *restriction-σ-related-set* :: ‹$'a$ ⇒ $'a$ ⇒ *real set*›
where ‹*restriction-σ-related-set x y* ≡ $\sigma_\downarrow$ ‘ *restriction-related-set x y*›

abbreviation *restriction-σ-not-related-set* :: ‹$'a$ ⇒ $'a$ ⇒ *real set*›
where ‹*restriction-σ-not-related-set x y* ≡ $\sigma_\downarrow$ ‘ *restriction-not-related-set x y*›

lemma *nonempty-restriction-σ-related-set* :
‹*restriction-σ-related-set x y* ≠ {}› ⟨*proof*⟩

lemma *restriction-σ-related-set-Un-restriction-σ-not-related-set* :
‹*restriction-σ-related-set x y* ∪ *restriction-σ-not-related-set x y* = *range* $\sigma_\downarrow$›
⟨*proof*⟩

lemma ‹*bdd-above* (*restriction-σ-related-set x y*)›
⟨*proof*⟩

lemma ‹*bdd-above* (*restriction-σ-not-related-set x y*)›
⟨*proof*⟩

lemma *bounded-restriction-σ-related-set*: ‹*bounded* (*restriction-σ-related-set x y*)›
⟨*proof*⟩

lemma *bounded-restriction-σ-not-related-set*: ‹*bounded* (*restriction-σ-not-related-set x y*)›
⟨*proof*⟩

corollary *restriction-space-Inf-properties*:
‹*a* ∈ *restriction-σ-related-set x y* ⟹ $dist_\downarrow$ *x y* ≤ *a*›
‹⟦⋀*a*. *a* ∈ *restriction-σ-related-set x y* ⟹ *b* ≤ *a*⟧ ⟹ *b* ≤ $dist_\downarrow$ *x y*›
⟨*proof*⟩

lemma *restriction-σ-related-set-alt* :
‹*restriction-σ-related-set x y* = {$\sigma_\downarrow$ *n*| *n*. *n* ∈ *restriction-related-set x y*}›
⟨*proof*⟩

lemma *exists-less-restriction-σ* : ‹∃ *n*. *m* < *n* ∧ $\sigma_\downarrow$ *n* < $\sigma_\downarrow$ *m*›
⟨*proof*⟩

lemma ‹$dist_\downarrow$ *x y* = *Inf* (*restriction-σ-related-set x y*)›
⟨*proof*⟩

lemma *not-related-imp-dist-restriction-is-some-restriction-σ* :
‹∃ *n*. $dist_\downarrow$ *x y* = $\sigma_\downarrow$ *n* ∧ (∀ *m*≤*n*. *x* ↓ *m* ≲≈ *y* ↓ *m*) › **if** ‹¬ *x* ≲≈ *y*›
⟨*proof*⟩

lemma *not-related-imp-dist-restriction-le-some-restriction-σ* :
‹¬ *x* ≲≈ *y* ⟹ ∃ *n*. $dist_\downarrow$ *x y* ≤ $\sigma_\downarrow$ *n* ∧ (¬ *x* ↓ *Suc n* ≲≈ *y* ↓ *Suc n*) ∧
(∀ *m*≤*n*. *x* ↓ *m* ≲≈ *y* ↓ *m*)›

⟨*proof*⟩

lemma *restriction-dist-eq-0-iff-related* : ‹$dist_{\downarrow}\ x\ y = 0 \longleftrightarrow x \lessapprox y$›
⟨*proof*⟩

end

locale *DecseqRestrictionSpace* = *NonDecseqRestrictionSpace* +
  assumes *decseq-restriction-σ* : ‹*decseq* $\sigma_{\downarrow}$›
begin

lemma *dist-restriction-is-bis* :
  ‹$dist_{\downarrow}\ x\ y$ = (*if* $x \lessapprox y$ *then* 0 *else* $\sigma_{\downarrow}$ (*Sup* (*restriction-related-set x y*)))›
  if ‹$x \in M$› **and** ‹$y \in M$›
⟨*proof*⟩

lemma *not-eq-imp-dist-restriction-is-restriction-σ-Sup-restriction-eq* :
  ‹$dist_{\downarrow}\ x\ y = \sigma_{\downarrow}$ (*Sup* (*restriction-related-set x y*))›
  ‹$\forall\, m{\leq}$*Sup* (*restriction-related-set x y*). $x \downarrow m \lessapprox y \downarrow m$›
  ‹$\forall\, m{>}$*Sup* (*restriction-related-set x y*). $\neg\ x \downarrow m \lessapprox y \downarrow m$›
  if ‹$\neg\ x \lessapprox y$› **and** ‹$x \in M$› **and** ‹$y \in M$›
  ⟨*proof*⟩

theorem *restriction-dist-tendsto-zero-independent-of-restriction-σ* :
  — Very powerful theorem: the convergence of the distance to *0* is actually independent from the restriction sequence chosen.
  — This is the theorem for which we had to work with locales first.

assumes ‹*DecseqRestrictionSpace* (↓) ($\lessapprox$) *restriction-σ′ restriction-dist′*›
  and ‹$\Sigma \in M$› **and** ‹*range* $\sigma \subseteq M$›
shows ‹$(\lambda n.\ dist_{\downarrow}\ (\sigma\ n)\ \Sigma) \longrightarrow 0 \longleftrightarrow (\lambda n.\ restriction\text{-}dist'\ (\sigma\ n)\ \Sigma) \longrightarrow 0$›
⟨*proof*⟩

end

3 Ultrametric Structure of restriction Spaces

This has only be proven with the sort constraint, not inside the context of the class *metric-space* ...

context *metric-space* **begin**

lemma *LIMSEQ-def* : ‹$X \longrightarrow L \longleftrightarrow (\forall r{>}0.\ \exists no.\ \forall n{\geq}no.\ dist\ (X\ n)\ L < r)$›
⟨*proof*⟩

lemma *LIMSEQ-iff-nz*: ‹$X \longrightarrow L \longleftrightarrow (\forall r{>}0.\ \exists no{>}0.\ \forall n{\geq}no.\ dist\ (X\ n)\ L < r)$›
⟨*proof*⟩

lemma *metric-LIMSEQ-I*: ‹$(\bigwedge r.\ 0 < r \Longrightarrow \exists no.\ \forall n{\geq}no.\ dist\ (X\ n)\ L < r) \Longrightarrow X \longrightarrow L$›
⟨*proof*⟩

lemma *metric-LIMSEQ-D*: ‹$X \longrightarrow L \Longrightarrow 0 < r \Longrightarrow \exists no.\ \forall n{\geq}no.\ dist\ (X\ n)\ L < r$›
⟨*proof*⟩

lemma *LIMSEQ-dist-iff*:
‹$f \longrightarrow l \longleftrightarrow (\lambda x.\ dist\ (f\ x)\ l) \longrightarrow 0$›
⟨*proof*⟩

lemma *Cauchy-converges-subseq*:
fixes u::$nat \Rightarrow 'a$
assumes *Cauchy* u
strict-mono r
$(u \circ r) \longrightarrow l$
shows $u \longrightarrow l$
⟨*proof*⟩

end

3.1 The Construction with Classes

class *restriction-*σ = *restriction-space* +
fixes *restriction-*σ :: ‹$'a$ *itself* $\Rightarrow$ *nat* $\Rightarrow$ *real*› (‹$\sigma_{\downarrow}$›)

⟨*ML*⟩

class *non-decseq-restriction-space* =
uniformity-dist + *open-uniformity* + *restriction-σ* +
— We do not assume the restriction sequence to be *decseq* yet.
assumes *restriction-σ-tendsto-zero'* : ‹$\sigma_\downarrow$ *TYPE*('*a*) $\longrightarrow$ *0*›
and *zero-less-restriction-σ'* [*simp*] : ‹*0* < $\sigma_\downarrow$ *TYPE*('*a*) *n*›

and *dist-restriction-is'* : ‹*dist x y* = (*INF n* ∈ {*n*. *x* ↓ *n* = *y* ↓ *n*}.
$\sigma_\downarrow$ *TYPE*('*a*) *n*)›
begin

sublocale *NonDecseqRestrictionSpace* ‹(↓)› ‹(=)› ‹*UNIV*› ‹$\sigma_\downarrow$ *TYPE*
('*a*)› *dist*
⟨*proof*⟩

end

⟨*ML*⟩

We hide duplicated facts $\sigma_\downarrow$ *TYPE*(*?'a*) $\longrightarrow$ *0*
0 < $\sigma_\downarrow$ *TYPE*(*?'a*) *?n*
dist ?x ?y = *Inf* (*restriction-σ-related-set ?x ?y*).
hide-fact *restriction-σ-tendsto-zero' zero-less-restriction-σ' dist-restriction-is'*

context *non-decseq-restriction-space* **begin**

subclass *ultrametric-space*
⟨*proof*⟩

end

context *non-decseq-restriction-space* **begin**

lemma *restriction-tendsto-self*: ‹(λ*n*. *x* ↓ *n*) $\longrightarrow$ *x*›
⟨*proof*⟩

end

class *decseq-restriction-space* = *non-decseq-restriction-space* +
assumes *decseq-restriction-σ'* : ‹*decseq* ($\sigma_\downarrow$ *TYPE*('*a*))›
begin

sublocale *DecseqRestrictionSpace* ‹(↓)› ‹(=)› ‹*UNIV* :: '*a set*›
‹*restriction-σ TYPE*('*a*)› *dist*

⟨*proof*⟩
lemmas *dist-restriction-is-bis-simplified* = *dist-restriction-is-bis*[*simplified*]
and *not-eq-imp-dist-restriction-is-restriction-σ-Sup-restriction-eq-simplified*
=
not-eq-imp-dist-restriction-is-restriction-σ-Sup-restriction-eq[*simplified*]

end

We hide duplicated fact *antimono-on UNIV* ($\sigma_\downarrow$ *TYPE*(*?'a*)).

hide-fact *decseq-restriction-σ'*

class *strict-decseq-restriction-space* = *non-decseq-restriction-space* +
assumes *strict-decseq-restriction-σ* : ‹*strict-decseq* ($\sigma_\downarrow$ *TYPE*('*a*))›
begin

subclass *decseq-restriction-space*
⟨*proof*⟩

end

Generic Properties

lemma (**in** *metric-space*) *dist-sequences-tendsto-zero-imp-tendsto-iff* :
‹($\lambda n.$ *dist* (σ *n*) (ψ *n*)) $\longrightarrow$ *0* $\Longrightarrow$ $\sigma \longrightarrow \Sigma \longleftrightarrow \psi \longrightarrow \Sigma$›
⟨*proof*⟩

lemma (**in** *non-decseq-restriction-space*) *restricted-sequence-tendsto-iff*
:
‹($\lambda n.$ σ *n* $\downarrow$ *n*) $\longrightarrow \Sigma \longleftrightarrow \sigma \longrightarrow \Sigma$›
⟨*proof*⟩

lemma (**in** *non-decseq-restriction-space*) *Cauchy-restriction-chain* :
‹*Cauchy* σ› **if** ‹*chain*$_\downarrow$ σ›
⟨*proof*⟩

lemma (**in** *non-decseq-restriction-space*) *restriction-tendsto-imp-tendsto*
:
‹$\sigma \longrightarrow \Sigma$› **if** ‹$\sigma$ $-\downarrow\rightarrow$ Σ›
⟨*proof*⟩

In Decseq Restriction Space

context *decseq-restriction-space* **begin**

lemma *le-dist-to-restriction-eqE* :
obtains k **where** ‹$n \leq k$› ‹$\bigwedge x\ y :: 'a.\ dist\ x\ y \leq \sigma_{\downarrow}\ TYPE('a)\ k \Longrightarrow x \downarrow n = y \downarrow n$›
⟨*proof*⟩

theorem *tendsto-iff-restriction-tendsto* : ‹$\sigma \longrightarrow \Sigma \longleftrightarrow \sigma\ -\downarrow\rightarrow \Sigma$›
⟨*proof*⟩

corollary *convergent-iff-restriction-convergent* : ‹*convergent* $\sigma \longleftrightarrow$ *convergent*$_{\downarrow}\ \sigma$›
⟨*proof*⟩

theorem *complete-iff-restriction-complete* :
‹$(\forall \sigma.$ *Cauchy* $\sigma \longrightarrow$ *convergent* $\sigma) \longleftrightarrow (\forall \sigma.$ *chain*$_{\downarrow}\ \sigma \longrightarrow$ *convergent*$_{\downarrow}\ \sigma)$›
— The following result shows that we have not lost anything with our definitions of convergence, completeness, etc. specific to restriction spaces.
⟨*proof*⟩

end

The following classes will be useful later.

class *complete-decseq-restriction-space* = *decseq-restriction-space* +
assumes *restriction-chain-imp-restriction-convergent'* : ‹*chain*$_{\downarrow}\ \sigma \Longrightarrow$ *convergent*$_{\downarrow}\ \sigma$›
begin

subclass *complete-restriction-space*
⟨*proof*⟩

subclass *complete-ultrametric-space*
⟨*proof*⟩

end

We hide duplicated fact *chain*$_{\downarrow}$ *?σ* $\Longrightarrow$ *convergent*$_{\downarrow}$ *?σ*.

hide-fact *restriction-chain-imp-restriction-convergent'*

class *complete-strict-decseq-restriction-space* = *strict-decseq-restriction-space* +
assumes *restriction-chain-imp-restriction-convergent'* : ‹*chain*$_{\downarrow}\ \sigma \Longrightarrow$ *convergent*$_{\downarrow}\ \sigma$›

begin

subclass *complete-decseq-restriction-space*
⟨*proof*⟩

end

We hide duplicated fact $chain_\downarrow\ ?\sigma \Longrightarrow convergent_\downarrow\ ?\sigma$.

hide-fact *restriction-chain-imp-restriction-convergent'*

class *restriction-δ* = *restriction-σ* +
fixes *restriction-δ* :: ‹'*a itself* ⇒ *real*› (‹$\delta_\downarrow$›)
assumes *zero-less-restriction-δ* [*simp*] : ‹$0 < \delta_\downarrow\ TYPE('a)$›
and *restriction-δ-less-one* [*simp*] : ‹$\delta_\downarrow\ TYPE('a) < 1$›
begin

lemma *zero-le-restriction-δ* [*simp*] : ‹$0 \leq \delta_\downarrow\ TYPE('a)$›
and *restriction-δ-le-one* [*simp*] : ‹$\delta_\downarrow\ TYPE('a) \leq 1$›
and *zero-le-pow-restriction-δ* [*simp*] : ‹$0 \leq \delta_\downarrow\ TYPE('a)\ \hat{}\ n$›
⟨*proof*⟩

lemma *pow-restriction-δ-le-one* [*simp*] : ‹$\delta_\downarrow\ TYPE('a)\ \hat{}\ n \leq 1$›
⟨*proof*⟩

lemma *pow-restriction-δ-less-one* [*simp*] : ‹$n \neq 0 \Longrightarrow \delta_\downarrow\ TYPE('a)\ \hat{}\ n < 1$›
⟨*proof*⟩

end

⟨*ML*⟩

class *at-least-geometric-restriction-space* =
uniformity-dist + *open-uniformity* + *restriction-δ* +
assumes *zero-less-restriction-σ'* : ‹$0 < \sigma_\downarrow\ TYPE('a)\ n$›
and *restriction-σ-le* :
‹$\sigma_\downarrow\ TYPE('a)\ (Suc\ n) \leq \delta_\downarrow\ TYPE('a) * \sigma_\downarrow\ TYPE('a)\ n$›
and *dist-restriction-is'* :
‹$dist\ x\ y = (INF\ n \in \{n.\ x \downarrow n = y \downarrow n\}.\ \sigma_\downarrow\ TYPE('a)\ n)$›

⟨*ML*⟩

context *at-least-geometric-restriction-space* **begin**

lemma *restriction-σ-le-restriction-σ-times-pow-restriction-δ* :
‹$\sigma_\downarrow\ TYPE('a)\ (n + k) \leq \sigma_\downarrow\ TYPE('a)\ n * \delta_\downarrow\ TYPE('a)\ \hat{}\ k$›

⟨*proof*⟩

lemma *restriction-σ-le-pow-restriction-δ* :
‹$\sigma_\downarrow$ *TYPE*('*a*) *n* ≤ $\sigma_\downarrow$ *TYPE*('*a*) *0* ∗ $\delta_\downarrow$ *TYPE*('*a*) ^ *n*›
⟨*proof*⟩

subclass *strict-decseq-restriction-space*
⟨*proof*⟩

lemma ‹*0* < $\delta_\downarrow$ *TYPE*('*a*) ^ *n*› ⟨*proof*⟩

end

We hide duplicated facts *0* < $\sigma_\downarrow$ *TYPE*(*?'a*) *?n*
dist ?x ?y = *Inf* (*restriction-σ-related-set ?x ?y*).

hide-fact *zero-less-restriction-σ' dist-restriction-is'*

class *complete-at-least-geometric-restriction-space* = *at-least-geometric-restriction-space*
+
assumes *restriction-chain-imp-restriction-convergent'* : ‹*chain*$_\downarrow$ *σ*
⟹ *convergent*$_\downarrow$ *σ*›
begin

subclass *complete-strict-decseq-restriction-space*
⟨*proof*⟩

end

We hide duplicated fact *chain*$_\downarrow$ *?σ* ⟹ *convergent*$_\downarrow$ *?σ*.

hide-fact *restriction-chain-imp-restriction-convergent'*

⟨*ML*⟩

class *geometric-restriction-space* = *uniformity-dist* + *open-uniformity*
+ *restriction-δ* +
assumes *restriction-σ-is* : ‹$\sigma_\downarrow$ *TYPE*('*a*) *n* = $\delta_\downarrow$ *TYPE*('*a*) ^ *n*›
and *dist-restriction-is'* : ‹*dist x y* = (*INF n* ∈ {*n*. *x* ↓ *n* = *y* ↓ *n*}.
$\sigma_\downarrow$ *TYPE*('*a*) *n*)›

begin

This is what "restriction space" usually mean in the literature. The previous classes are generalizations of this concept (even this one is a generalization, since we usually have $\delta_\downarrow\ TYPE('a) = 1\ /\ 2$).

subclass *at-least-geometric-restriction-space*
⟨*proof*⟩

lemma ‹$0 < \delta_\downarrow\ TYPE('a)\ \widehat{}\ n$› ⟨*proof*⟩

end

⟨*ML*⟩

We hide duplicated fact *dist ?x ?y = Inf* (*restriction-σ-related-set ?x ?y*).

hide-fact *dist-restriction-is'*

class *complete-geometric-restriction-space* = *geometric-restriction-space* +
  assumes *restriction-chain-imp-restriction-convergent'* : ‹$chain_\downarrow\ \sigma \Longrightarrow convergent_\downarrow\ \sigma$›
begin

subclass *complete-at-least-geometric-restriction-space*
  ⟨*proof*⟩

end

We hide duplicated fact $chain_\downarrow\ ?\sigma \Longrightarrow convergent_\downarrow\ ?\sigma$.

hide-fact *restriction-chain-imp-restriction-convergent'*

theorem *geometric-restriction-space-completeI* : ‹*convergent σ*›
  if ‹$\bigwedge\sigma :: nat \Rightarrow 'a.$ *restriction-chain σ* $\Longrightarrow \exists\,\Sigma.\ \forall\, n.\ \Sigma \downarrow n = \sigma\ n$›
    and ‹*Cauchy σ*› **for** $\sigma :: ‹nat \Rightarrow 'a ::$ *geometric-restriction-space*›
  ⟨*proof*⟩
lemma (**in** *non-decseq-restriction-space*) *restriction-cball-subset-cball* :
  ‹$\sigma_\downarrow\ TYPE('a)\ n \leq r \Longrightarrow \mathcal{B}_\downarrow(\Sigma,\ n) \subseteq \{x.\ dist\ \Sigma\ x \leq r\}$›
  ⟨*proof*⟩

corollary *restriction-cball-subset-cball-bis* :
  ‹$\sigma_\downarrow\ TYPE('a)\ n \leq r \Longrightarrow \mathcal{B}_\downarrow(\Sigma,\ n) \subseteq cball\ \Sigma\ r$›
  for $\Sigma :: ‹'a ::$ *non-decseq-restriction-space*›

⟨*proof*⟩

lemma (**in** *non-decseq-restriction-space*) *restriction-ball-subset-ball* :
‹$\sigma_\downarrow$ *TYPE*('*a*) *n* < *r* ⟹ *restriction-ball* Σ *n* ⊆ {*x*. *dist* Σ *x* < *r*}›
⟨*proof*⟩

corollary *restriction-ball-subset-ball-bis* :
‹$\sigma_\downarrow$ *TYPE*('*a*) *n* < *r* ⟹ *restriction-ball* Σ *n* ⊆ *ball* Σ *r*›
for Σ :: ‹'*a* :: *non-decseq-restriction-space*›
⟨*proof*⟩

lemma (**in** *strict-decseq-restriction-space*)
restriction-cball-is-cball : ‹$\mathcal{B}_\downarrow$(Σ, *n*) = {*x*. *dist* Σ *x* ≤ $\sigma_\downarrow$ *TYPE*('*a*) *n*}›
⟨*proof*⟩

lemma *restriction-cball-is-cball-bis* : ‹$\mathcal{B}_\downarrow$(Σ, *n*) = *cball* Σ ($\sigma_\downarrow$ *TYPE*('*a*) *n*)›
for Σ :: ‹'*a* :: *strict-decseq-restriction-space*›
⟨*proof*⟩

lemma (**in** *strict-decseq-restriction-space*)
restriction-ball-is-ball : ‹*restriction-ball* Σ *n* = {*x*. *dist* Σ *x* < $\sigma_\downarrow$ *TYPE*('*a*) *n*}›
⟨*proof*⟩

lemma *restriction-ball-is-ball-bis* : ‹*restriction-ball* Σ *n* = *ball* Σ ($\sigma_\downarrow$ *TYPE*('*a*) *n*)›
for Σ :: ‹'*a* :: *strict-decseq-restriction-space*›
⟨*proof*⟩

lemma *isCont-iff-restriction-cont-at* : ‹*isCont f* Σ ⟷ *restriction-cont-at f* Σ›
for *f* :: ‹'*a* :: *decseq-restriction-space* ⇒ '*b* :: *decseq-restriction-space*›
⟨*proof*⟩

lemma (**in** *strict-decseq-restriction-space*)
open-iff-restriction-open : ‹*open U* ⟷ $open_\downarrow$ *U*›
⟨*proof*⟩

lemma (**in** *strict-decseq-restriction-space*)
closed-iff-restriction-closed : ‹*closed U* ⟷ *closed*$_\downarrow$ *U*›
⟨*proof*⟩

lemma *continuous-on-iff-restriction-cont-on* :
‹*open U* ⟹ *continuous-on U f* ⟷ *restriction-cont-on f U*›
for *f* :: ‹'*a* :: *decseq-restriction-space* ⇒ '*b* :: *decseq-restriction-space*›
⟨*proof*⟩

3.2 Equivalence between Lipschitz Map and Restriction shift Map

For a function *f*::'*a* ⇒ '*b*, it is equivalent to have *restriction-shift-on f k A* and *lipschitz-with-on f* $(\delta_\downarrow\ TYPE('b))^k$ *A* when '*a* is of sort *geometric-restriction-space* and $\sigma_\downarrow\ TYPE('b) = \sigma_\downarrow\ TYPE('a)$ ('*b* is then necessarily also of sort *geometric-restriction-space*).

Weaker versions of this result wan be established with weaker assumptions on the sort, this is what we do below.

lemma *restriction-shift-nonneg-on-imp-lipschitz-with-on* :
‹*lipschitz-with-on f* (*restriction-δ TYPE*('*b*) ^ *k*) *A*› **if** ‹*restriction-shift-on f* (*int k*) *A*›
and *le-restriction-σ* : ‹⋀*n*. *restriction-σ TYPE*('*b*) *n* ≤ *restriction-σ TYPE*('*a*) *n*›
for *f* :: ‹'*a* :: *decseq-restriction-space* ⇒ '*b* :: *at-least-geometric-restriction-space*›
⟨*proof*⟩

corollary *restriction-shift-nonneg-imp-lipschitz-with* :
‹⟦*restriction-shift f* (*int k*); ⋀*n*. *restriction-σ TYPE*('*b*) *n* ≤ *restriction-σ TYPE*('*a*) *n*⟧
⟹ *lipschitz-with f* (*restriction-δ TYPE*('*b*) ^ *k*)›
for *f* :: ‹'*a* :: *decseq-restriction-space* ⇒ '*b* :: *at-least-geometric-restriction-space*›
⟨*proof*⟩

lemma *lipschitz-with-on-imp-restriction-shift-neg-on* :
‹*restriction-shift-on f* (− *int k*) *A*› **if** ‹*lipschitz-with-on f* (*restriction-δ TYPE*('*b*) *powi* − *int k*) *A*›
and *le-restriction-σ* : ‹⋀*n*. *restriction-σ TYPE*('*a*) *n* ≤ *restriction-σ TYPE*('*b*) *n*›
for *f* :: ‹'*a* :: *decseq-restriction-space* ⇒ '*b* :: *at-least-geometric-restriction-space*›
⟨*proof*⟩

corollary *lipschitz-with-imp-restriction-shift-neg* :

‹⟦*lipschitz-with f* (*restriction-δ TYPE*('*b*) *powi* − *int k*);
⋀*n*. *restriction-σ TYPE*('*a*) *n* ≤ *restriction-σ TYPE*('*b*) *n*⟧
⟹ *restriction-shift f* (− *int k*)›
for *f* :: ‹'*a* :: *decseq-restriction-space* ⇒ '*b* :: *at-least-geometric-restriction-space*›
⟨*proof*⟩

We obtained that *restriction-shift* implies *lipschitz-with* when *0* ≤ *k* and that *lipschitz-with* implies *restriction-shift* when *k* ≤ *0*. To cover the remaining cases, we give move from *at-least-geometric-restriction-space* to *geometric-restriction-space*.

theorem *lipschitz-with-on-iff-restriction-shift-on* :
‹*lipschitz-with-on f* (*restriction-δ TYPE*('*b*) *powi k*) *A* ⟷ *restriction-shift-on f k A*›
if *same-restriction-σ* : ‹*restriction-σ TYPE*('*b*) = *restriction-σ TYPE*('*a*)›
for *f* :: ‹'*a* :: *decseq-restriction-space* ⇒ '*b* :: *geometric-restriction-space*›
⟨*proof*⟩

corollary *lipschitz-with-iff-restriction-shift* :
‹*restriction-σ TYPE*('*b*) = *restriction-σ TYPE*('*a*) ⟹
lipschitz-with f (*restriction-δ TYPE*('*b*) *powi k*) ⟷ *restriction-shift f k*›
for *f* :: ‹'*a* :: *decseq-restriction-space* ⇒ '*b* :: *geometric-restriction-space*›
⟨*proof*⟩

4 Functions

4.1 Restriction Space

instantiation ‹*fun*› :: (*type*, *restriction-σ*) *restriction-σ*
begin

definition *restriction-σ-fun* :: ‹('*a* ⇒ '*b*) *itself* ⇒ *nat* ⇒ *real*›
where ‹*restriction-σ-fun* - ≡ *restriction-σ TYPE*('*b*)›

instance ⟨*proof*⟩

end

instantiation ‹*fun*› :: (*type*, *non-decseq-restriction-space*) *non-decseq-restriction-space*
begin

definition *dist-fun* :: ‹['*a* ⇒ '*b*, '*a* ⇒ '*b*] ⇒ *real*›
where ‹*dist-fun f g* ≡ *INF n* ∈ *restriction-related-set f g*. *restriction-σ TYPE*('*a* ⇒ '*b*) *n*›

definition *uniformity-fun* :: ‹(('*a* ⇒ '*b*) × ('*a* ⇒ '*b*)) *filter*›
where ‹*uniformity-fun* ≡ *INF e*∈{*0* <..}. *principal* {(*x*, *y*). *dist x y* < *e*}›

definition *open-fun* :: ‹('a ⇒ 'b) set ⇒ bool›
  where ‹*open-fun U* ≡ ∀ *x*∈*U*. *eventually* (λ(*x'*, *y*). *x'* = *x* ⟶ *y* ∈ *U*) *uniformity*›

instance ⟨*proof*⟩

end

instance ‹*fun*› :: (*type*, *decseq-restriction-space*) *decseq-restriction-space*
  ⟨*proof*⟩

instance ‹*fun*› :: (*type*, *strict-decseq-restriction-space*) *strict-decseq-restriction-space*
  ⟨*proof*⟩

instantiation ‹*fun*› :: (*type*, *restriction-δ*) *restriction-δ*
begin

definition *restriction-δ-fun* :: ‹('a ⇒ 'b) *itself* ⇒ *real*›
  where ‹*restriction-δ-fun* - ≡ *restriction-δ* *TYPE*('b)›

instance ⟨*proof*⟩

end

instance ‹*fun*› :: (*type*, *at-least-geometric-restriction-space*) *at-least-geometric-restriction-space*
⟨*proof*⟩

instance ‹*fun*› :: (*type*, *geometric-restriction-space*) *geometric-restriction-space*
⟨*proof*⟩

lemma *dist-image-le-dist-fun* : ‹*dist* (*f x*) (*g x*) ≤ *dist f g*›
  for *f g* :: ‹'a ⇒ 'b :: *non-decseq-restriction-space*›
⟨*proof*⟩

lemma *Sup-dist-image-le-dist-fun* : ‹(*SUP x*. *dist* (*f x*) (*g x*)) ≤ *dist f g*›
  for *f g* :: ‹'a ⇒ 'b :: *non-decseq-restriction-space*›
  ⟨*proof*⟩

context fixes *f g* :: ‹'a ⇒ 'b :: *decseq-restriction-space*› **begin**

lemma *reached-dist-fun* : ‹∃ *x*. *dist f g* = *dist* (*f x*) (*g x*)›

⟨*proof*⟩

lemma *dist-fun-eq-Sup-dist-image* : ‹*dist f g* = (*SUP x*. *dist* (*f x*) (*g x*))›
⟨*proof*⟩

lemma *fun-restriction-space-Sup-properties* :
‹*dist* (*f x*) (*g x*) ≤ *dist f g*›
‹(⋀*x*. *dist* (*f x*) (*g x*) ≤ *b*) ⟹ *dist f g* ≤ *b*›
⟨*proof*⟩

end

4.2 Completeness

Actually we can obtain even better: when the instance *'b* of *decseq-restriction-space* is also an instance of *complete-space*, the type *'a* ⇒ *'b* is an instance of *complete-space*.

This is because when *'b* is an instance of *decseq-restriction-space* (and not only *non-decseq-restriction-space*) the distance between two functions is reached (see ∃ *x*. *dist ?f ?g* = *dist* (*?f x*) (*?g x*)).

The only remaining thing is to prove that completeness is preserved on higher-order.

instance ‹*fun*› :: (*type*, *complete-decseq-restriction-space*) *complete-decseq-restriction-space*
⟨*proof*⟩

instance ‹*fun*› :: (*type*, *complete-strict-decseq-restriction-space*) *complete-strict-decseq-restriction-space*
⟨*proof*⟩

instance ‹*fun*› :: (*type*, *complete-at-least-geometric-restriction-space*) *complete-at-least-geometric-restriction-space*
⟨*proof*⟩

instance ‹*fun*› :: (*type*, *complete-geometric-restriction-space*) *complete-geometric-restriction-space*
⟨*proof*⟩

4.3 Kind of Extensionality

context fixes *f* :: ‹['*a* :: *metric-space*, *'b* :: *type*] ⇒ *'c* :: *decseq-restriction-space*› **begin**

lemma *lipschitz-with-simplification*:
‹*lipschitz-with f α* ⟷ (∀ *y*. *lipschitz-with* (λ*x*. *f x y*) *α*)›

⟨*proof*⟩

lemma *non-expanding-simplification* :
‹*non-expanding f* ⟷ (∀ *y*. *non-expanding* (λ*x*. *f x y*))›
⟨*proof*⟩

lemma *contraction-with-simplification*:
‹*contraction-with f α* ⟷ (∀ *y*. *contraction-with* (λ*x*. *f x y*) *α*)›
⟨*proof*⟩

end

5 Product

The product type $'a \times 'b$ of to metric spaces is already instantiated as a metric space by setting *dist x y* = *sqrt* ((*dist* (*fst x*) (*fst y*))^2 + (*dist* (*snd x*) (*snd y*))^2). Unfortunately, this definition is not compatible with the distance required by the *non-decseq-restriction-space*.. We first have to define a new product type with a trivial **typedef**.

5.1 Isomorphic Product Construction

5.1.1 Definition and First Properties

typedef ('*a*, '*b*) $prod_{max}$ (‹(- ×$_{max}$/ -)› [*21*, *20*] *20*) = ‹*UNIV* :: ('*a* × '*b*) *set*›
morphisms $from\text{-}prod_{max}$ $to\text{-}prod_{max}$ ⟨*proof*⟩

declare $from\text{-}prod_{max}\text{-}inject$ [*simp*]
$from\text{-}prod_{max}\text{-}inverse$ [*simp*]

lemmas $to\text{-}prod_{max}\text{-}inject\text{-}simplified$ [*simp*] = $to\text{-}prod_{max}\text{-}inject$ [*simplified*]
and $to\text{-}prod_{max}\text{-}inverse\text{-}simplified$[*simp*] = $to\text{-}prod_{max}\text{-}inverse$[*simplified*]

lemmas $to\text{-}prod_{max}\text{-}induct\text{-}simplified$ = $to\text{-}prod_{max}\text{-}induct$[*simplified*]
and $to\text{-}prod_{max}\text{-}cases\text{-}simplified$ = $to\text{-}prod_{max}\text{-}cases$ [*simplified*]
and $from\text{-}prod_{max}\text{-}induct\text{-}simplified$ = $from\text{-}prod_{max}\text{-}induct$[*simplified*]
and $from\text{-}prod_{max}\text{-}cases\text{-}simplified$ = $from\text{-}prod_{max}\text{-}cases$ [*simplified*]

setup-lifting $type\text{-}definition\text{-}prod_{max}$

lift-definition $Pair_{max} :: \langle 'a \Rightarrow 'b \Rightarrow 'a \times_{max} 'b \rangle$ **is** $Pair$ ⟨*proof*⟩

free-constructors $case\text{-}prod_{max}$ **for** $Pair_{max}$ fst_{max} snd_{max}
⟨*proof*⟩

lemma $fst_{max}\text{-}def$: $\langle fst_{max} \equiv map\text{-}fun\ from\text{-}prod_{max}\ id\ fst \rangle$
⟨*proof*⟩
lemma $fst_{max}\text{-}rep\text{-}eq$: $\langle fst_{max}\ x = fst\ (from\text{-}prod_{max}\ x) \rangle$
⟨*proof*⟩

lemma $fst_{max}\text{-}abs\text{-}eq$ [*simp*] : $\langle fst_{max}\ (to\text{-}prod_{max}\ y) = fst\ y \rangle$
⟨*proof*⟩

lemma $fst_{max}\text{-}transfer$ [*transfer-rule*]: $\langle rel\text{-}fun\ (pcr\text{-}prod_{max}\ (=)\ (=))\ (=)\ fst\ fst_{max} \rangle$
⟨*proof*⟩

lemma $snd_{max}\text{-}def$: $\langle snd_{max} \equiv map\text{-}fun\ from\text{-}prod_{max}\ id\ snd \rangle$
⟨*proof*⟩

lemma $snd_{max}\text{-}rep\text{-}eq$: $\langle snd_{max}\ x = snd\ (from\text{-}prod_{max}\ x) \rangle$
⟨*proof*⟩

lemma $snd_{max}\text{-}abs\text{-}eq$ [*simp*] : $\langle snd_{max}\ (to\text{-}prod_{max}\ y) = snd\ y \rangle$
⟨*proof*⟩

lemma $snd_{max}\text{-}transfer$ [*transfer-rule*] : $\langle rel\text{-}fun\ (pcr\text{-}prod_{max}\ (=)\ (=))\ (=)\ snd\ snd_{max} \rangle$
⟨*proof*⟩

lemma $case\text{-}prod_{max}\text{-}def$: $\langle case\text{-}prod_{max} \equiv map\text{-}fun\ id\ (map\text{-}fun\ from\text{-}prod_{max}\ id)\ case\text{-}prod \rangle$
⟨*proof*⟩

lemma $case\text{-}prod_{max}\text{-}rep\text{-}eq$: $\langle case\text{-}prod_{max}\ f\ p = (case\ from\text{-}prod_{max}\ p\ of\ (x,\ y) \Rightarrow f\ x\ y) \rangle$
⟨*proof*⟩

lemma $case\text{-}prod_{max}\text{-}abs\text{-}eq$ [*simp*] : $\langle case\text{-}prod_{max}\ f\ (to\text{-}prod_{max}\ q) = (case\ q\ of\ (x,\ y) \Rightarrow f\ x\ y) \rangle$
⟨*proof*⟩

lemma $case\text{-}prod_{max}\text{-}transfer$ [*transfer-rule*] : $\langle rel\text{-}fun\ (=)\ (rel\text{-}fun\ (pcr\text{-}prod_{max}\ (=)\ (=))\ (=))\ case\text{-}prod\ case\text{-}prod_{max} \rangle$
⟨*proof*⟩

5.2 Syntactic Sugar

The following syntactic sugar is of course recovered from the theory *HOL.Product-Type*.

nonterminal $tuple\text{-}args_{max}$ **and** $patterns_{max}$
syntax
$\text{-}tuple_{max}$ :: $'a \Rightarrow tuple\text{-}args_{max} \Rightarrow 'a \times_{max} 'b$ $(\langle(1\{\!|\text{-},/\ \text{-}|\!\})\rangle)$
$\text{-}tuple\text{-}arg_{max}$ :: $'a \Rightarrow tuple\text{-}args_{max}$ $(\langle\text{-}\rangle)$
$\text{-}tuple\text{-}args_{max}$ :: $'a \Rightarrow tuple\text{-}args_{max} \Rightarrow tuple\text{-}args_{max}$ $(\langle\text{-},/\ \text{-}\rangle)$
$\text{-}pattern_{max}$ :: $pttrn \Rightarrow patterns_{max} \Rightarrow pttrn$ $(\langle\{\!|\text{-},/\ \text{-}|\!\}\rangle)$
:: $pttrn \Rightarrow patterns_{max}$ $(\langle\text{-}\rangle)$
$\text{-}patterns_{max}$ :: $pttrn \Rightarrow patterns_{max} \Rightarrow patterns_{max}$ $(\langle\text{-},/\ \text{-}\rangle)$
translations
$\{\!|x,\ y|\!\} \rightleftharpoons CONST\ Pair_{max}\ x\ y$
$\text{-}pattern_{max}\ x\ y \rightleftharpoons CONST\ Pair_{max}\ x\ y$
$\text{-}patterns_{max}\ x\ y \rightleftharpoons CONST\ Pair_{max}\ x\ y$
$\text{-}tuple_{max}\ x\ (\text{-}tuple\text{-}args_{max}\ y\ z) \rightleftharpoons \text{-}tuple_{max}\ x\ (\text{-}tuple\text{-}arg_{max}\ (\text{-}tuple_{max}\ y\ z))$
$\lambda\{\!|x,\ y,\ zs|\!\}.\ b \rightleftharpoons CONST\ case\text{-}prod_{max}\ (\lambda x\ \{\!|y,\ zs|\!\}.\ b)$
$\lambda\{\!|x,\ y|\!\}.\ b \rightleftharpoons CONST\ case\text{-}prod_{max}\ (\lambda x\ y.\ b)$
$\text{-}abs\ (CONST\ Pair_{max}\ x\ y)\ t \rightharpoonup \lambda\{\!|x,\ y|\!\}.\ t$
— This rule accommodates tuples in *case* $C\ \ldots\ \{\!|x,\ y|\!\}\ \ldots \Rightarrow \ldots$: The $\{\!|x,\ y|\!\}$ is parsed as $Pair_{max}\ x\ y$ because it is *logic*, not *pttrn*.

With this syntactic sugar, one can write *case* a *of* $\{\!|b,\ c,\ d,\ e|\!\} \Rightarrow \{\!|c,\ d|\!\}, \lambda\{\!|y,\ u|\!\}.\ a, \lambda\{\!|a,\ b|\!\}.\ \{\!|a,\ b,\ c,\ d,\ e|\!\}, \lambda\{\!|a,\ b,\ c|\!\}.\ a, \ldots$ as for the type $'a \times 'b$.

lemmas $to\text{-}prod_{max}\text{-}tuple$ $[simp] = Pair_{max}.abs\text{-}eq[symmetric]$
and $from\text{-}prod_{max}\text{-}tuple_{max}$ $[simp] = Pair_{max}.rep\text{-}eq$

5.3 Product

We first redo the work of *Restriction-Spaces.Restriction-Spaces-Prod*.

instantiation $prod_{max}$ :: (*restriction*, *restriction*) *restriction*
begin

lift-definition $restriction\text{-}prod_{max}$ :: $\langle 'a \times_{max} 'b \Rightarrow nat \Rightarrow 'a \times_{max} 'b\rangle$ **is** $\langle(\downarrow)\rangle$ ⟨*proof*⟩

lemma $restriction\text{-}prod_{max}\text{-}def'$: $\langle p \downarrow n = \{\!|fst_{max}\ p \downarrow n,\ snd_{max}\ p \downarrow n|\!\}\rangle$
⟨*proof*⟩

instance ⟨*proof*⟩

end

instance $prod_{max}$ :: (*restriction-space*, *restriction-space*) *restriction-space*
⟨*proof*⟩

instantiation $prod_{max}$ :: (*restriction-σ*, *restriction-σ*) *restriction-σ*
begin

definition *restriction-σ-prod*$_{max}$ :: ‹('*a* ×$_{max}$ '*b*) *itself* ⇒ *nat* ⇒ *real*›
where ‹*restriction-σ-prod*$_{max}$ - *n* ≡
max (*restriction-σ TYPE*('*a*) *n*) (*restriction-σ TYPE*('*b*) *n*)›

instance ⟨*proof*⟩
end

instantiation $prod_{max}$ :: (*non-decseq-restriction-space*, *non-decseq-restriction-space*)
non-decseq-restriction-space
begin

definition *dist-prod*$_{max}$ :: ‹['*a* ×$_{max}$ '*b*, '*a* ×$_{max}$ '*b*] ⇒ *real*›
where ‹*dist-prod*$_{max}$ *f g* ≡ *INF n* ∈ *restriction-related-set f g*. *restriction-σ TYPE*('*a* ×$_{max}$ '*b*) *n*›

definition *uniformity-prod*$_{max}$ :: ‹(('*a* ×$_{max}$ '*b*) × '*a* ×$_{max}$ '*b*) *filter*›
where ‹*uniformity-prod*$_{max}$ ≡ *INF e*∈{*0* <..}. *principal* {(*x*, *y*). *dist x y* < *e*}›

definition *open-prod*$_{max}$ :: ‹('*a* ×$_{max}$ '*b*) *set* ⇒ *bool*›
where ‹*open-prod*$_{max}$ *U* ≡ ∀ *x*∈*U*. *eventually* (λ(*x'*, *y*). *x'* = *x* ⟶ *y* ∈ *U*) *uniformity*›

instance
⟨*proof*⟩

end

instance $prod_{max}$ :: (*decseq-restriction-space*, *decseq-restriction-space*)
decseq-restriction-space
⟨*proof*⟩

instance $prod_{max}$ :: (*strict-decseq-restriction-space*, *strict-decseq-restriction-space*)
strict-decseq-restriction-space
⟨*proof*⟩

instantiation $prod_{max}$ :: (*restriction-δ*, *restriction-δ*) *restriction-δ*
begin

definition *restriction-δ-prod*$_{max}$ :: ‹(*'a* ×$_{max}$ *'b*) *itself* ⇒ *real*›
where ‹*restriction-δ-prod*$_{max}$ - ≡ *max* (*restriction-δ TYPE*(*'a*)) (*restriction-δ TYPE*(*'b*))›

instance ⟨*proof*⟩

end

instance *prod*$_{max}$ :: (*at-least-geometric-restriction-space*, *at-least-geometric-restriction-space*) *at-least-geometric-restriction-space*
⟨*proof*⟩

instance *prod*$_{max}$ :: (*geometric-restriction-space*, *geometric-restriction-space*) *geometric-restriction-space*
⟨*proof*⟩

lemma *max-dist-projections-le-dist-prod*$_{max}$:
‹*max* (*dist* (*fst*$_{max}$ *p1*) (*fst*$_{max}$ *p2*)) (*dist* (*snd*$_{max}$ *p1*) (*snd*$_{max}$ *p2*)) ≤ *dist p1 p2*›
⟨*proof*⟩

5.4 Completeness

5.4.1 Preliminaries

default-sort *non-decseq-restriction-space* — Otherwise we should always specify.

lemma *restriction-σ-prod*$_{max}$*-commute* :
‹*restriction-σ TYPE*(*'b* ×$_{max}$ *'a*) = *restriction-σ TYPE*(*'a* ×$_{max}$ *'b*)›
⟨*proof*⟩

definition *dist-left-prod*$_{max}$ :: ‹(*'a* ×$_{max}$ *'b*) *itself* ⇒ *'a* ⇒ *'a* ⇒ *real*›
where ‹*dist-left-prod*$_{max}$ - *x y* ≡ *INF n* ∈ *restriction-related-set x y*. *restriction-σ TYPE*(*'a* ×$_{max}$ *'b*) *n*›

definition *dist-right-prod*$_{max}$ :: ‹(*'a* ×$_{max}$ *'b*) *itself* ⇒ *'b* ⇒ *'b* ⇒ *real*›
where ‹*dist-right-prod*$_{max}$ - *x y* ≡ *INF n* ∈ *restriction-related-set x y*. *restriction-σ TYPE*(*'a* ×$_{max}$ *'b*) *n*›

lemma *dist-right-prod*$_{max}$*-is-dist-left-prod*$_{max}$:
‹*dist-right-prod*$_{max}$ *TYPE*(*'b* ×$_{max}$ *'a*) = *dist-left-prod*$_{max}$ *TYPE*(*'a* ×$_{max}$ *'b*)›
⟨*proof*⟩

lemma *dist-le-dist-left-prod*$_{max}$: ‹*dist x y* ≤ *dist-left-prod*$_{max}$ *TYPE*('*a* ×$_{max}$ '*b*) *x y*›
⟨*proof*⟩

lemma *dist-le-dist-right-prod*$_{max}$: ‹*dist x y* ≤ *dist-right-prod*$_{max}$ *TYPE*('*b* ×$_{max}$ '*a*) *x y*›
⟨*proof*⟩

lemma
fixes *p1 p2* :: ‹'*a* :: *decseq-restriction-space* ×$_{max}$ '*b* :: *decseq-restriction-space*›
shows *dist-prod*$_{max}$*-le-max-dist-left-prod*$_{max}$*-dist-right-prod*$_{max}$:
‹*dist p1 p2* ≤ *max* (*dist-left-prod*$_{max}$ *TYPE*('*a* ×$_{max}$ '*b*) (*fst*$_{max}$ *p1*) (*fst*$_{max}$ *p2*))
(*dist-right-prod*$_{max}$ *TYPE*('*a* ×$_{max}$ '*b*) (*snd*$_{max}$ *p1*) (*snd*$_{max}$ *p2*))›
⟨*proof*⟩

default-sort *type* — Back to normal.

5.4.2 Complete Restriction Space

When the instances '*a* and '*b* of *decseq-restriction-space* are also instances of *complete-space*, the type '*a* ×$_{max}$ '*b* is an instance of *complete-space*.

lemma *restriction-chain-prod*$_{max}$*-iff* :
‹*restriction-chain* σ ⟷ *restriction-chain* (λ*n*. *fst*$_{max}$ (σ *n*)) ∧
restriction-chain (λ*n*. *snd*$_{max}$ (σ *n*))›
⟨*proof*⟩

lemma *restriction-tendsto-prod*$_{max}$*-iff* :
‹σ −↓→ Σ ⟷ (λ*n*. *fst*$_{max}$ (σ *n*)) −↓→ *fst*$_{max}$ Σ ∧ (λ*n*. *snd*$_{max}$ (σ *n*)) −↓→ *snd*$_{max}$ Σ›
⟨*proof*⟩

lemma *restriction-convergent-prod*$_{max}$*-iff* :
‹*restriction-convergent* σ ⟷ *restriction-convergent* (λ*n*. *fst*$_{max}$ (σ *n*)) ∧
restriction-convergent (λ*n*. *snd*$_{max}$ (σ *n*))›
⟨*proof*⟩

instance *prod*$_{max}$ :: (*complete-decseq-restriction-space*, *complete-decseq-restriction-space*)
complete-decseq-restriction-space
⟨*proof*⟩

instance $prod_{max}$ :: (*complete-strict-decseq-restriction-space*, *complete-strict-decseq-restriction-space*)
complete-strict-decseq-restriction-space
⟨*proof*⟩

instance $prod_{max}$ :: (*complete-at-least-geometric-restriction-space*, *complete-at-least-geometric-restriction-space*)
complete-at-least-geometric-restriction-space
⟨*proof*⟩

instance $prod_{max}$ :: (*complete-geometric-restriction-space*, *complete-geometric-restriction-space*)
complete-geometric-restriction-space
⟨*proof*⟩

When the types $'a$ and $'b$ share the same restriction sequence, we have the following equality.

lemma *same-restriction-σ-imp-restriction-σ-*$prod_{max}$*-is* [*simp*] :
‹*restriction-σ TYPE*($'b$ :: *non-decseq-restriction-space*) =
restriction-σ TYPE($'a$ :: *non-decseq-restriction-space*) ⟹
restriction-σ TYPE($'a$ ×$_{max}$ $'b$) = *restriction-σ TYPE*($'a$)›
⟨*proof*⟩

lemma *same-restriction-σ-imp-dist-*$prod_{max}$*-eq-max-dist-projections* :
‹*dist p1 p2* = *max* (*dist* (fst_{max} *p1*) (fst_{max} *p2*)) (*dist* (snd_{max} *p1*) (snd_{max} *p2*))›
if *same-restriction-σ* [*simp*] : ‹*restriction-σ TYPE*($'b$) = *restriction-σ TYPE*($'a$)›
for *p1 p2* :: ‹$'a$ :: *decseq-restriction-space* ×$_{max}$ $'b$ :: *decseq-restriction-space*›
⟨*proof*⟩

Now, one can write things like $\upsilon\ \langle\!|x,\ y|\!\rangle.\ f\ \langle\!|x,\ y|\!\rangle$.

We could of course imagine more support for $'a$ ×$_{max}$ $'b$ type, but as restriction spaces are intended to be used without recourse to metric spaces, we have not undertaken this task for the time being.

6 Main entry Point