Providing restriction Spaces with an ultrametric Structure

Benoît Ballenghien Benjamin Puyobro Burkhart Wolff

June 2, 2025

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›
unfolding *lipschitz-with-on-def* **by** (*metis dist-eq-0-iff zero-le-dist zero-le-mult-iff*)

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›

by (*rule lipschitz-with-onI*)

lemma *lipschitz-with-onD1* : ‹*lipschitz-with-on f α A* ⟹ *0* ≤ *α*›
unfolding *lipschitz-with-on-def* **by** *simp*

lemma *lipschitz-withD1* : ‹*lipschitz-with f α* ⟹ *0* ≤ *α*›
by (*rule lipschitz-with-onD1*)

lemma *lipschitz-with-onD2* :
‹*lipschitz-with-on f α A* ⟹ *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *dist* (*f x*) (*f y*) ≤ *α* ∗ *dist x y*›
unfolding *lipschitz-with-on-def* **by** *simp*

lemma *lipschitz-withD2* :
‹*lipschitz-with f α* ⟹ *dist* (*f x*) (*f y*) ≤ *α* ∗ *dist x y*›
unfolding *lipschitz-with-on-def* **by** *simp*

lemma *lipschitz-with-imp-lipschitz-with-on*: ‹*lipschitz-with f α* ⟹ *lipschitz-with-on f α A*›
by (*simp add*: *lipschitz-with-on-def*)

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 (*rule lipschitz-with-onI*)
show ‹*0* ≤ *β*› **by** (*meson order-trans lipschitz-with-onD1 that*)
next
fix *x y* **assume** ‹*x* ∈ *A*› **and** ‹*y* ∈ *A*›
from ‹*lipschitz-with-on f α A*›[*THEN lipschitz-with-onD2*, *OF this*]
have ‹*dist* (*f x*) (*f y*) ≤ *α* ∗ *dist x y*› **.**
from *order-trans*[*OF this*] **show** ‹*dist* (*f x*) (*f y*) ≤ *β* ∗ *dist x y*›
by (*simp add*: *mult-right-mono* ‹*α* ≤ *β*›)
qed

theorem *lipschitz-with-on-comp-lipschitz-with-on* :
‹*lipschitz-with-on* (*λx. g* (*f x*)) (*β* ∗ *α*) *A*›
if ‹*f* ‘ *A* ⊆ *B*› ‹*lipschitz-with-on g β B*› ‹*lipschitz-with-on f α A*›
proof (*rule lipschitz-with-onI*)
show ‹*0* ≤ *β* ∗ *α*› **by** (*metis lipschitz-with-onD1 mult-nonneg-nonneg that*(*2*, *3*))
next
fix *x y* **assume** ‹*x* ∈ *A*› **and** ‹*y* ∈ *A*›
with ‹*f* ‘ *A* ⊆ *B*› **have** ‹*f x* ∈ *B*› **and** ‹*f y* ∈ *B*› **by** *auto*
have ‹*dist* (*g* (*f x*)) (*g* (*f y*)) ≤ *β* ∗ *dist* (*f x*) (*f y*)›
by (*fact that*(*2*)[*THEN lipschitz-with-onD2*, *OF* ‹*f x* ∈ *B*› ‹*f y* ∈ *B*›])
also have ‹*...* ≤ *β* ∗ (*α* ∗ *dist x y*)›

by (*fact mult-left-mono*[*OF that*(3)[*THEN lipschitz-with-onD2*, *OF* ‹$x \in A$› ‹$y \in A$›]
that(2)[*THEN lipschitz-with-onD1*]])
also have ‹$\ldots = \beta * \alpha * dist\ x\ y$› **by** *simp*
finally show ‹$dist\ (g\ (f\ x))\ (g\ (f\ y)) \leq \ldots$› **.**
qed

corollary *lipschitz-with-comp-lipschitz-with* :
‹⟦*lipschitz-with g β*; *lipschitz-with f α*⟧ ⟹
lipschitz-with (λ*x*. *g* (*f x*)) (*β* ∗ *α*)›
using *lipschitz-with-on-comp-lipschitz-with-on* **by** *blast*

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*⟧ ⟹
non-expanding-on f A›
by (*simp add*: *lipschitz-with-onI non-expanding-on-def*)

lemma *non-expandingI* :
‹⟦⋀*x y*. *x* ≠ *y* ⟹ *f x* ≠ *f y* ⟹ *dist* (*f x*) (*f y*) ≤ *dist x y*⟧ ⟹ *non-expanding f*›
by (*rule non-expanding-onI*)

lemma *non-expanding-onD* :
‹*non-expanding-on f A* ⟹ *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *dist* (*f x*) (*f y*) ≤ *dist x y*›
by (*metis lipschitz-with-onD2 mult-1 non-expanding-on-def*)

lemma *non-expandingD* : ‹*non-expanding f* ⟹ *dist* (*f x*) (*f y*) ≤ *dist x y*›
by (*simp add*: *non-expanding-onD*)

lemma *non-expanding-imp-non-expanding-on*: ‹*non-expanding f* ⟹ *non-expanding-on f A*›
by (*meson non-expandingD non-expanding-onI*)

lemma *non-expanding-on-comp-non-expanding-on* :
‹⟦*f* ‘ *A* ⊆ *B*; *non-expanding-on g B*; *non-expanding-on f A*⟧ ⟹

non-expanding-on (λx. *g* (*f x*)) *A*›
unfolding *non-expanding-on-def*
by (*metis* (*no-types*) *lipschitz-with-on-comp-lipschitz-with-on mult-1*)

corollary *non-expanding-comp-non-expanding* :
‹⟦*non-expanding g*; *non-expanding f*⟧ ⟹ *non-expanding* (λx. *g* (*f x*))›
by (*blast intro*: *non-expanding-on-comp-non-expanding-on*)

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*] ⇒ *bool*›
where ‹*contraction f* ≡ *contraction-on f UNIV*›

lemma *contraction-with-onI* :
‹⟦*0* ≤ α; α < *1*; ⋀*x y*. ⟦*x* ∈ *A*; *y* ∈ *A*; *x* ≠ *y*; *f x* ≠ *f y*⟧ ⟹ *dist* (*f x*) (*f y*) ≤ α ∗ *dist x y*⟧
⟹ *contraction-with-on f* α *A*›
by (*simp add*: *contraction-with-on-def lipschitz-with-onI*)

lemma *contraction-withI* :
‹⟦*0* ≤ α; α < *1*; ⋀*x y*. *x* ≠ *y* ⟹ *f x* ≠ *f y* ⟹ *dist* (*f x*) (*f y*) ≤ α ∗ *dist x y*⟧ ⟹
contraction-with f α›
by (*rule contraction-with-onI*)

lemma *contraction-with-onD1* : ‹*contraction-with-on f* α *A* ⟹ *0* ≤ α›
by (*simp add*: *contraction-with-on-def lipschitz-with-on-def*)

lemma *contraction-withD1* : ‹*contraction-with f* α ⟹ *0* ≤ α›
by (*simp add*: *contraction-with-onD1*)

lemma *contraction-with-onD2* : ‹*contraction-with-on f* α *A* ⟹ α <

1›
by (*simp add*: *contraction-with-on-def lipschitz-with-on-def*)

lemma *contraction-withD2* : ‹*contraction-with f α* ⟹ *α* < *1*›
by (*simp add*: *contraction-with-onD2*)

lemma *contraction-with-onD3* :
‹*contraction-with-on f α A* ⟹ *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *dist* (*f x*) (*f y*) ≤ *α* ∗ *dist x y*›
by (*simp add*: *contraction-with-on-def lipschitz-with-on-def*)

lemma *contraction-withD3* : ‹*contraction-with f α* ⟹ *dist* (*f x*) (*f y*) ≤ *α* ∗ *dist x y*›
by (*simp add*: *contraction-with-on-def lipschitz-withD2*)

lemma *contraction-with-imp-contraction-with-on*:
‹*contraction-with f α* ⟹ *contraction-with-on f α A*›
by (*simp add*: *contraction-with-on-def lipschitz-with-imp-lipschitz-with-on*)

lemma *contraction-imp-contraction-on*: ‹*contraction f* ⟹ *contraction-on f A*›
using *contraction-on-def contraction-with-imp-contraction-with-on*
by *blast*

lemma *contraction-with-on-imp-contraction-on* :
‹*contraction-with-on f α A* ⟹ *contraction-on f A*›
unfolding *contraction-on-def* **by** *blast*

lemma *contraction-with-imp-contraction*: ‹*contraction-with f α* ⟹ *contraction f*›
by (*simp add*: *contraction-with-on-imp-contraction-on*)

lemma *contraction-onE*:
‹⟦*contraction-on f A*; ⋀*α*. *contraction-with-on f α A* ⟹ *thesis*⟧ ⟹ *thesis*›
unfolding *contraction-on-def* **by** *blast*

lemma *contractionE*:
‹⟦*contraction f*; ⋀*α*. *contraction-with f α* ⟹ *thesis*⟧ ⟹ *thesis*›
by (*elim contraction-onE*)

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*›
by (*simp add*: *contraction-with-on-def lipschitz-with-on-imp-lipschitz-with-on-ge*)

1.2 Properties

lemma *contraction-with-on-imp-lipschitz-with-on*[*simp*] :
‹*contraction-with-on f α A* ⟹ *lipschitz-with-on f α A*›
by (*simp add*: *contraction-with-on-def*)

lemma *non-expanding-on-imp-lipschitz-with-one-on*[*simp*] :
‹*non-expanding-on f A* ⟹ *lipschitz-with-on f 1 A*›
by (*simp add*: *non-expanding-on-def*)

lemma *contraction-on-imp-non-expanding-on*[*simp*] :
‹*contraction-on f A* ⟹ *non-expanding-on f A*›
proof (*elim contraction-onE*, *rule non-expanding-onI*)
fix *α x y* **assume** ‹*x* ∈ *A*› **and** ‹*y* ∈ *A*› **and** *contra* : ‹*contraction-with-on f α A*›
show ‹*dist* (*f x*) (*f y*) ≤ *dist x y*›
by (*rule order-trans*[*OF contra*[*THEN contraction-with-onD3*, *OF* ‹*x* ∈ *A*› ‹*y* ∈ *A*›]])
(*metis contra contraction-with-on-def mult-less-cancel-right1 nle-le order-less-le zero-le-dist*)
qed

lemma *contraction-with-on-comp-contraction-with-on* :
‹*contraction-with-on* (λ*x*. *g* (*f x*)) (*β* ∗ *α*) *A*›
if ‹*f* ‘ *A* ⊆ *B*› ‹*contraction-with-on g β B*› ‹*contraction-with-on f α A*›
proof (*unfold contraction-with-on-def*, *intro conjI*)
from *that*(*2*, *3*)[*THEN contraction-with-onD2*] *that*(*2*)[*THEN contraction-with-onD1*]
show ‹*β* ∗ *α* < *1*› **by** (*metis dual-order.strict-trans2 mult-less-cancel-left1 nle-le order-less-le*)
next
show ‹*lipschitz-with-on* (λ*x*. *g* (*f x*)) (*β* ∗ *α*) *A*›
by (*rule lipschitz-with-on-comp-lipschitz-with-on*[*OF* ‹*f* ‘ *A* ⊆ *B*›])
(*simp-all add*: *that*(*2*, *3*))
qed

corollary *contraction-with-comp-contraction-with* :
‹⟦*contraction-with g β*; *contraction-with f α*⟧ ⟹ *contraction-with* (λ*x*. *g* (*f x*)) (*β* ∗ *α*)›
by (*blast intro*: *contraction-with-on-comp-contraction-with-on*)

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 (*elim contraction-onE*)
fix *α β* **assume** ‹*f* ‘ *A* ⊆ *B*› ‹*contraction-with-on g β B*› ‹*contraction-with-on f α A*›
from *contraction-with-on-comp-contraction-with-on*[*OF this*]

show ‹*contraction-on* (λ*x*. *g* (*f x*)) *A*› **by** (*fact contraction-with-on-imp-contraction-on*)
qed

corollary *contraction-comp-contraction* :
‹⟦*contraction g*; *contraction f*⟧ ⟹ *contraction* (λ*x*. *g* (*f x*))›
by (*blast intro*: *contraction-on-comp-contraction-on*)

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 (*unfold contraction-with-on-def*, *intro conjI*)
from *that*(*2*)[*THEN contraction-with-onD2*] **show** ‹*β* < *1*› **.**
next
show ‹*lipschitz-with-on* (λ*x*. *g* (*f x*)) *β A*›
by (*rule lipschitz-with-on-comp-lipschitz-with-on*[*OF* ‹*f* ‘ *A* ⊆ *B*›,
of g β 1, *simplified*])
(*simp-all add*: *that*(*2*, *3*))
qed

corollary *contraction-with-comp-non-expanding* :
‹⟦*contraction-with g β*; *non-expanding f*⟧ ⟹ *contraction-with* (λ*x*.
g (*f x*)) *β*›
by (*blast intro*: *contraction-with-on-comp-non-expanding-on*)

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*›
by (*metis contraction-on-def contraction-with-on-comp-non-expanding-on*)

corollary *contraction-comp-non-expanding* :
‹⟦*contraction g*; *non-expanding f*⟧ ⟹ *contraction* (λ*x*. *g* (*f x*))›
by (*blast intro*: *contraction-on-comp-non-expanding-on*)

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 (*unfold contraction-with-on-def*, *intro conjI*)
from *that*(*3*)[*THEN contraction-with-onD2*] **show** ‹*α* < *1*› **.**
next
show ‹*lipschitz-with-on* (λ*x*. *g* (*f x*)) *α A*›
by (*rule lipschitz-with-on-comp-lipschitz-with-on*[*OF* ‹*f* ‘ *A* ⊆ *B*›,
of g 1, *simplified*])
(*simp-all add*: *that*(*2*, *3*))
qed

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

$g\ (f\ x))\ \alpha\rangle$
by (*blast intro*: *non-expanding-on-comp-contraction-with-on*)

corollary *non-expanding-on-comp-contraction-on* :
$\langle[\![f\ `\ A \subseteq B;\ non\text{-}expanding\text{-}on\ g\ B;\ contraction\text{-}on\ f\ A]\!] \Longrightarrow contraction\text{-}on\ (\lambda x.\ g\ (f\ x))\ A\rangle$
by (*metis contraction-on-def non-expanding-on-comp-contraction-with-on*)

corollary *non-expanding-comp-contraction* :
$\langle[\![non\text{-}expanding\ g;\ contraction\ f]\!] \Longrightarrow contraction\ (\lambda x.\ g\ (f\ x))\rangle$
by (*blast intro*: *non-expanding-on-comp-contraction-on*)

1.3 Banach's fixed-point Theorems

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

theorem *Banach-fix-type* : $\langle contraction\ f \Longrightarrow \exists!x.\ f\ x = x\rangle$
for $f :: \langle 'a :: complete\text{-}space \Rightarrow 'a\rangle$
by (*elim contractionE*)
(*metis banach-fix-type contraction-withD1 contraction-withD2 contraction-withD3*)

theorem *Banach-fix*:
$\langle contraction\text{-}on\ f\ s \Longrightarrow \exists!x.\ x \in s \wedge f\ x = x\rangle$ **if** $\langle complete\ s\rangle\ \langle s \neq \{\}\rangle\ \langle f\ `\ s \subseteq s\rangle$
proof (*elim contraction-onE*, *intro banach-fix*[*OF* $\langle complete\ s\rangle\ \langle s \neq \{\}\rangle$ - - $\langle f\ `\ s \subseteq s\rangle$] *ballI*)
show $\langle contraction\text{-}with\text{-}on\ f\ \alpha\ s \Longrightarrow 0 \leq \alpha\rangle$ **for** α **by** (*fact contraction-with-onD1*)
next
show $\langle contraction\text{-}with\text{-}on\ f\ \alpha\ s \Longrightarrow \alpha < 1\rangle$ **for** α **by** (*fact contraction-with-onD2*)
next
show $\langle contraction\text{-}with\text{-}on\ f\ \alpha\ s \Longrightarrow x \in s \Longrightarrow y \in s \Longrightarrow$
$dist\ (f\ x)\ (f\ y) \leq \alpha * dist\ x\ y\rangle$ **for** $\alpha\ x\ y$ **by** (*fact contraction-with-onD3*)
qed

2 Locales factorizing the proof Work

2.1 Preliminaries on strictly decreasing Sequences

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

lemma *strict-decseq-def* : $\langle strict\text{-}decseq\ X \longleftrightarrow (\forall\, m\ n.\ m < n \longrightarrow X$

$n < X\ m)$›
by (*fact monotone-def*)

lemma *strict-decseqI* : ‹*strict-decseq X*› **if** ‹$\bigwedge n.\ X\ (Suc\ n) < X\ n$›
by (*metis Suc-le-eq decseqD decseq-Suc-iff le-less-trans nless-le strict-decseq-def that*)

lemma *strict-decseqD* : ‹*strict-decseq* $X \Longrightarrow m < n \Longrightarrow X\ n < X\ m$›
using *strict-decseq-def* **by** *blast*

lemma *strict-decseq-def-bis* : ‹*strict-decseq* $X \longleftrightarrow (\forall\, m\ n.\ X\ n < X\ m \longleftrightarrow m < n)$›
by (*metis linorder-less-linear order-less-imp-not-less strict-decseq-def*)

lemma *strict-decseq-def-ter* : ‹*strict-decseq* $X \longleftrightarrow (\forall\, m\ n.\ X\ n \leq X\ m \longleftrightarrow m \leq n)$›
unfolding *strict-decseq-def-bis*
by (*rule iffI*, *metis le-simps*(*2*) *order-le-less*, *simp add*: *less-le-not-le*)

lemma *strict-decseq-imp-decseq* : ‹*strict-decseq* $\sigma \Longrightarrow$ *decseq* σ›
by (*simp add*: *monotone-on-def order-le-less*)

lemma *strict-decseq-SucI* : ‹$(\bigwedge n.\ X\ (Suc\ n) < X\ n) \Longrightarrow$ *strict-decseq X*›
by (*metis Suc-le-eq decseqD decseq-Suc-iff less-le-not-le order-less-le strict-decseq-def*)

lemma *strict-decseq-SucD* : ‹*strict-decseq* $A \Longrightarrow A\ (Suc\ i) < A\ i$›
by (*simp add*: *strict-decseq-def*)

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*›
by (*simp add*: *order-less-imp-le*)

lemma *restriction-σ-neq-zero* [*simp*] : ‹$\sigma_\downarrow$ *n* ≠ *0*›
by (*metis zero-less-restriction-σ order-less-irrefl*)

lemma *bounded-range-restriction-σ*: ‹*bounded* (*range* $\sigma_\downarrow$)›
by (*fact convergent-imp-bounded*[*OF restriction-σ-tendsto-zero*])

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* ≠ {}› **by** *simp*

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* $\sigma_\downarrow$›
by *blast*

lemma ‹*bdd-above* (*restriction-σ-related-set x y*)›
by (*meson bdd-above.I2 bdd-above.unfold bounded-imp-bdd-above bounded-range-restriction-σ rangeI*)

lemma ‹*bdd-above* (*restriction-σ-not-related-set x y*)›
by (*meson bdd-above.E bdd-above.I2 bounded-imp-bdd-above bounded-range-restriction-σ range-eqI*)

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

by (*meson bounded-range-restriction-σ bounded-subset image-mono top-greatest*)

lemma *bounded-restriction-σ-not-related-set*: ‹*bounded* (*restriction-σ-not-related-set x y*)›
by (*meson bounded-range-restriction-σ bounded-subset image-mono subset-UNIV*)

corollary *restriction-space-Inf-properties*:
‹$a \in$ *restriction-σ-related-set x y* $\Longrightarrow$ $dist_{\downarrow}\ x\ y \leq a$›
‹$[\![\bigwedge a.\ a \in$ *restriction-σ-related-set x y* $\Longrightarrow b \leq a]\!] \Longrightarrow b \leq dist_{\downarrow}\ x\ y$›
unfolding *dist-restriction-is*
by (*simp-all add*: *bounded-has-Inf*(*1*) *bounded-restriction-σ-related-set cInf-greatest*)

lemma *restriction-σ-related-set-alt* :
‹*restriction-σ-related-set x y* = $\{\sigma_{\downarrow}\ n|\ n.\ n \in$ *restriction-related-set x y*$\}$›
by *blast*

lemma *exists-less-restriction-σ* : ‹$\exists\, n.\ m < n \wedge \sigma_{\downarrow}\ n < \sigma_{\downarrow}\ m$›
proof (*rule ccontr*)
assume ‹$\neg\ (\exists\, n{>}m.\ \sigma_{\downarrow}\ n < \sigma_{\downarrow}\ m)$›
hence ‹$\forall\, n{\geq}m.\ \sigma_{\downarrow}\ m \leq \sigma_{\downarrow}\ n$›
by (*metis linorder-not-le nle-le*)
hence ‹$\neg\ \sigma_{\downarrow} \longrightarrow 0$›
by (*meson Lim-bounded2 linorder-not-le zero-less-restriction-σ*)
with *restriction-σ-tendsto-zero* **show** *False* **by** *simp*
qed

lemma ‹$dist_{\downarrow}\ x\ y$ = *Inf* (*restriction-σ-related-set x y*)›
by (*fact dist-restriction-is*)

lemma *not-related-imp-dist-restriction-is-some-restriction-σ* :
‹$\exists\, n.\ dist_{\downarrow}\ x\ y = \sigma_{\downarrow}\ n \wedge (\forall\, m{\leq}n.\ x \downarrow m \lessapprox y \downarrow m)$ › **if** ‹$\neg\ x \lessapprox y$›
proof −
have ‹*finite* (*restriction-related-set x y*)›
by (*simp add*: *finite-restriction-related-set-iff* ‹$\neg\ x \lessapprox y$›)
have ‹*Inf* (*restriction-σ-related-set x y*) ∈ *restriction-σ-related-set x y*›
by (*rule closed-contains-Inf*[*OF nonempty-restriction-σ-related-set*])
(*simp-all add*: ‹*finite* (*restriction-related-set x y*)› *finite-imp-closed*)
hence ‹$dist_{\downarrow}\ x\ y \in$ *restriction-σ-related-set x y*›
by (*fold dist-restriction-is*)
with *restriction-related-le* **obtain** *n*
where ‹$n \in$ *restriction-related-set x y*› ‹$dist_{\downarrow}\ x\ y = \sigma_{\downarrow}\ n$›

‹∀ $m{\leq}n$. $x \downarrow m \lessapprox y \downarrow m$› **by** *blast*
with ‹$dist_{\downarrow}\ x\ y \in$ *restriction-σ-related-set x y*› **show** *?thesis* **by** *blast*
qed

lemma *not-related-imp-dist-restriction-le-some-restriction-σ* :
‹$\neg\ x \lessapprox y \Longrightarrow \exists\, n.\ dist_{\downarrow}\ x\ y \leq \sigma_{\downarrow}\ n \wedge (\neg\ x \downarrow Suc\ n \lessapprox y \downarrow Suc\ n) \wedge (\forall\, m{\leq}n.\ x \downarrow m \lessapprox y \downarrow m)$›
by (*blast intro*: *restriction-space-Inf-properties*
dest: *ex-not-restriction-related-optimized*)

lemma *restriction-dist-eq-0-iff-related* : ‹$dist_{\downarrow}\ x\ y = 0 \longleftrightarrow x \lessapprox y$›
proof (*rule iffI*)
show ‹$dist_{\downarrow}\ x\ y = 0 \Longrightarrow x \lessapprox y$›
by (*erule contrapos-pp*)
(*auto dest*: *not-related-imp-dist-restriction-is-some-restriction-σ*)
next
show ‹$dist_{\downarrow}\ x\ y = 0$› **if** ‹$x \lessapprox y$›
proof (*rule order-antisym*)
show ‹$0 \leq dist_{\downarrow}\ x\ y$› **by** (*simp add*: *cINF-greatest dist-restriction-is*)
next
define Δ **where** ‹$\Delta\ n \equiv -\ \sigma_{\downarrow}\ n$› **for** *n*
have ∗ : ‹$\Delta\ n \leq -\ dist_{\downarrow}\ x\ y$› **for** *n*
unfolding Δ-*def* **using** ‹$x \lessapprox y$› *restriction-space-Inf-properties*(*1*)
by *simp* (*metis UNIV-restriction-related-set-iff rangeI*)
from *restriction-σ-tendsto-zero tendsto-minus-cancel-left*
have ‹$\Delta \longrightarrow 0$› **unfolding** Δ-*def* **by** *force*
from *lim-le*[*OF convergentI*[*OF* ‹$\Delta \longrightarrow 0$›] ∗] *limI*[*OF* ‹$\Delta \longrightarrow 0$›]
show ‹$dist_{\downarrow}\ x\ y \leq 0$› **by** *simp*
qed
qed

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 (*split if-split*, *intro conjI impI*)
from ‹$x \in M$› **and** ‹$y \in M$› **show** ‹$x \lessapprox y \Longrightarrow$ *restriction-dist x y* = 0›

by (*simp add*: *restriction-dist-eq-0-iff-related*)
next
show ‹$dist_{\downarrow}\ x\ y = \sigma_{\downarrow}\ (Sup\ (restriction\text{-}related\text{-}set\ x\ y))$› **if** ‹$\neg\ x \lessapprox y$›
proof (*rule order-antisym*)
show ‹$dist_{\downarrow}\ x\ y \leq \sigma_{\downarrow}\ (Sup\ (restriction\text{-}related\text{-}set\ x\ y))$›
unfolding *dist-restriction-is*
by (*rule cINF-lower*[*OF - Sup-in-restriction-related-set*[*OF* ‹$\neg\ x \lessapprox y$›]])
(*meson bdd-below.I2 zero-le-restriction-σ*)
next
show ‹$\sigma_{\downarrow}\ (Sup\ (restriction\text{-}related\text{-}set\ x\ y)) \leq dist_{\downarrow}\ x\ y$›
proof (*unfold dist-restriction-is*, *rule cINF-greatest*)
show ‹$restriction\text{-}related\text{-}set\ x\ y \neq \{\}$› **by** (*fact nonempty-restriction-related-set*)
next
fix *n* **assume** ‹$n \in restriction\text{-}related\text{-}set\ x\ y$›
hence ‹$n \leq Sup\ (restriction\text{-}related\text{-}set\ x\ y)$›
by (*metis* ‹$n \in restriction\text{-}related\text{-}set\ x\ y$› *le-cSup-finite*
finite-restriction-related-set-iff ‹$\neg\ x \lessapprox y$›)
from *decseqD*[*OF decseq-restriction-σ this*]
show ‹$restriction\text{-}σ\ (Sup\ (restriction\text{-}related\text{-}set\ x\ y)) \leq restriction\text{-}σ\ n$› **.**
qed
qed
qed

lemma *not-eq-imp-dist-restriction-is-restriction-σ-Sup-restriction-eq* :
‹$dist_{\downarrow}\ x\ y = \sigma_{\downarrow}\ (Sup\ (restriction\text{-}related\text{-}set\ x\ y))$›
‹$\forall\, m{\leq}Sup\ (restriction\text{-}related\text{-}set\ x\ y).\ x \downarrow m \lessapprox y \downarrow m$›
‹$\forall\, m{>}Sup\ (restriction\text{-}related\text{-}set\ x\ y).\ \neg\ x \downarrow m \lessapprox y \downarrow m$›
if ‹$\neg\ x \lessapprox y$› **and** ‹$x \in M$› **and** ‹$y \in M$›
subgoal by (*subst dist-restriction-is-bis*; *simp add*: ‹$\neg\ x \lessapprox y$› ‹$x \in M$› ‹$y \in M$›)
subgoal using *Sup-in-restriction-related-set*[*OF* ‹$\neg\ x \lessapprox y$›] *restriction-related-le* **by** *blast*
using *cSup-upper*[*OF - bdd-above-restriction-related-set-iff*[*THEN iffD2*, *OF* ‹$\neg\ x \lessapprox y$›],
of ‹$Suc\ (Sup\ (restriction\text{-}related\text{-}set\ x\ y))$›]
by (*metis* (*mono-tags*, *lifting*) *Suc-leI dual-order.refl mem-Collect-eq not-less-eq-eq restriction-related-le*)

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* (↓) (≲≈) *restriction-σ′ restriction-dist′*›
and ‹Σ ∈ *M*› **and** ‹*range σ* ⊆ *M*›
shows ‹(λ*n*. *dist*↓ (*σ n*) Σ) ⟶ *0* ⟷ (λ*n*. *restriction-dist′* (*σ n*) Σ) ⟶ *0*›
proof −
{ fix *restriction-σ restriction-dist restriction-σ′ restriction-dist′*
assume *a1* : ‹*DecseqRestrictionSpace* (↓) (≲≈) *restriction-σ restriction-dist* ›
and *a2* : ‹*DecseqRestrictionSpace* (↓) (≲≈) *restriction-σ′ restriction-dist′*›
and ∗ : ‹(λ*n*. *restriction-dist* (*σ n*) Σ) ⟶ *0*›

interpret *i1* : *DecseqRestrictionSpace* ‹(↓)› ‹(≲≈)› *M restriction-σ restriction-dist* **by** (*fact a1*)
interpret *i2* : *DecseqRestrictionSpace* ‹(↓)› ‹(≲≈)› *M restriction-σ′ restriction-dist′* **by** (*fact a2*)

have ‹(λ*n*. *restriction-dist′* (*σ n*) Σ) ⟶ *0*›
proof (*rule metric-LIMSEQ-I*)
fix ε :: *real* **assume** ‹*0* < ε›

from *metric-LIMSEQ-D*[*OF i2.restriction-σ-tendsto-zero* ‹*0* < ε›]
obtain *N* **where** ∗∗ : ‹*N* ≤ *n* ⟹ *restriction-σ′ n* < ε› **for** *n* **by** *auto*

fix *N′* :: *nat*

have ‹∃ *N′*. ∀ *n*≥*N′*. *N* ∈ *restriction-related-set* (*σ n*) Σ›
proof (*rule ccontr*)
assume ‹∄ *N′*. ∀ *n*≥*N′*. *N* ∈ *restriction-related-set* (*σ n*) Σ›
hence ∗∗∗ : ‹∀ *N′*. ∃ *n*≥*N′*. ¬ *σ n* ↓ *N* ≲≈ Σ ↓ *N*› **by** *simp*
have ∗∗∗∗ : ‹∀ *N′*. ∃ *n*≥*N′*. *restriction-σ N* ≤ *restriction-dist* (*σ n*) Σ›
proof (*rule allI*)
fix *N′* :: *nat*
from ∗∗∗ **obtain** *n* **where** ‹*N′* ≤ *n*› ‹¬ *σ n* ↓ *N* ≲≈ Σ ↓ *N*› **by** *blast*
hence ‹¬ *σ n* ≲≈ Σ› **using** *mono-restriction-related* **by** *blast*
have ‹*restriction-σ N* ≤ *restriction-dist* (*σ n*) Σ›
proof (*subst i1.dist-restriction-is-bis*)
show ‹*σ n* ∈ *M*› ‹Σ ∈ *M*›
by (*simp-all add*: *assms*(*2*, *3*) *range-subsetD*)
next
show ‹*restriction-σ N* ≤ (*if σ n* ≲≈ Σ *then 0*
else restriction-σ (*Sup* (*restriction-related-set* (*σ n*) Σ)))›
using ‹¬ *σ n* ≲≈ Σ› ‹¬ *σ n* ↓ *N* ≲≈ Σ ↓ *N*› *assms*(*2*, *3*)

nle-le
not-eq-imp-dist-restriction-is-restriction-σ-Sup-restriction-eq(*2*)
by (*fastforce intro*: *decseqD*[*OF i1.decseq-restriction-σ*])
qed
with ‹$N' \leq n$› **show** ‹$\exists\, n{\geq}N'.$ *restriction-σ* $N \leq$ *restriction-dist* $(\sigma\ n)\ \Sigma$› **by** *blast*
qed
from *metric-LIMSEQ-D*[*OF* $*$, *of* ‹*restriction-σ* N›]
have ‹$\exists\, N''.\ \forall\, n{\geq}N''.$ *restriction-dist* $(\sigma\ n)\ \Sigma <$ *restriction-σ* N›
by (*metis abs-of-nonneg i1.zero-le-restriction-σ i1.zero-less-restriction-σ norm-conv-dist order-trans real-norm-def*
verit-comp-simplify1(*3*))
with $****$ **show** *False* **by** *fastforce*
qed

then obtain N' **where** $***$: ‹$N' \leq n \Longrightarrow N \in$ *restriction-related-set* $(\sigma\ n)\ \Sigma$› **for** n **by** *blast*
have ‹*restriction-dist'* $(\sigma\ n)\ \Sigma < \varepsilon$› **if** ‹$N' \leq n$› **for** n
proof (*rule le-less-trans*[*OF i2.restriction-space-Inf-properties*(*1*), *of* ‹*restriction-σ'* N›])
from ‹$N' \leq n$› $***$
show ‹*restriction-σ'* $N \in$ *i2.restriction-σ-related-set* $(\sigma\ n)\ \Sigma$› **by** *blast*
next
show ‹*restriction-σ'* $N < \varepsilon$› **by** (*simp add*: $**$)
qed
thus ‹$\exists\, N.\ \forall\, n{\geq}N.$ *dist* (*restriction-dist'* $(\sigma\ n)\ \Sigma)\ 0 < \varepsilon$›
by (*metis abs-of-nonneg dist-0-norm dist-commute i2.not-related-imp-dist-restriction-is-some-restric*
i2.restriction-dist-eq-0-iff-related i2.zero-less-restriction-σ order-less-imp-not-less real-norm-def
verit-comp-simplify1(*3*))
qed }
note $* =$ *this*

show ‹$(\lambda n.$ *dist*$_{\downarrow}$ $(\sigma\ n)\ \Sigma) \longrightarrow 0 = (\lambda n.$ *restriction-dist'* $(\sigma\ n)\ \Sigma) \longrightarrow 0$›
using $*$ *DecseqRestrictionSpace-axioms assms*(*1*) **by** *blast*
qed

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* : ‹$X \longrightarrow L \longleftrightarrow (\forall r{>}0.\ \exists no.\ \forall n{\geq}no.\ dist\ (X\ n)\ L < r)$›
  unfolding *tendsto-iff eventually-sequentially* **..**

lemma *LIMSEQ-iff-nz*: ‹$X \longrightarrow L \longleftrightarrow (\forall r{>}0.\ \exists no{>}0.\ \forall n{\geq}no.\ dist\ (X\ n)\ L < r)$›
  by (*meson Suc-leD LIMSEQ-def zero-less-Suc*)

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

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

lemma *LIMSEQ-dist-iff*:
  ‹$f \longrightarrow l \longleftrightarrow (\lambda x.\ dist\ (f\ x)\ l) \longrightarrow 0$›
proof (*unfold LIMSEQ-def*, *rule iffI*)
  show ‹$(\lambda n.\ dist\ (f\ n)\ l) \longrightarrow 0 \Longrightarrow (\forall r{>}0.\ \exists no.\ \forall n{\geq}no.\ dist\ (f\ n)\ l < r)$›
   by (*metis* (*mono-tags*, *lifting*) *eventually-at-top-linorder order-tendstoD*(*2*))
next
  show ‹$\forall r{>}0.\ \exists no.\ \forall n{\geq}no.\ dist\ (f\ n)\ l < r \Longrightarrow (\lambda n.\ dist\ (f\ n)\ l) \longrightarrow 0$›
    by (*simp add*: *metric-space-class.LIMSEQ-def*)
qed

lemma *Cauchy-converges-subseq*:
  fixes *u*::*nat* ⇒ *'a*
  assumes *Cauchy u*
    strict-mono r
    $(u \circ r) \longrightarrow l$
  shows $u \longrightarrow l$
proof −
  have ∗: *eventually* $(\lambda n.\ dist\ (u\ n)\ l < e)$ *sequentially* **if** $e > 0$ **for** *e*
  proof −
    have $e/2 > 0$ **using** *that* **by** *auto*
    then obtain *N1* **where** *N1*: $\bigwedge m\ n.\ m \geq N1 \Longrightarrow n \geq N1 \Longrightarrow dist\ (u\ m)\ (u\ n) < e/2$
      using ‹*Cauchy u*› **unfolding** *Cauchy-def* **by** *blast*
    obtain *N2* **where** *N2*: $\bigwedge n.\ n \geq N2 \Longrightarrow dist\ ((u \circ r)\ n)\ l < e\ /\ 2$
      using *order-tendstoD*(*2*)[*OF iffD1*[*OF LIMSEQ-dist-iff* ‹$(u \circ r) \longrightarrow l$›] ‹$e/2 > 0$›]

unfolding *eventually-sequentially* **by** *auto*
have *dist* (*u n*) *l* < *e* **if** *n* ≥ *max N1 N2* **for** *n*
proof −
have *dist* (*u n*) *l* ≤ *dist* (*u n*) ((*u* ∘ *r*) *n*) + *dist* ((*u* ∘ *r*) *n*) *l*
by (*rule dist-triangle*)
also have ... < *e*/2 + *e*/2
proof (*intro add-strict-mono*)
show *dist* (*u n*) ((*u* ∘ *r*) *n*) < *e* / 2
using *N1*[*of n r n*] *N2*[*of n*] *that* **unfolding** *comp-def*
by (*meson assms*(2) *le-trans max.bounded-iff strict-mono-imp-increasing*)
show *dist* ((*u* ∘ *r*) *n*) *l* < *e* / 2
using *N2 that* **by** *auto*
qed
finally show *?thesis* **by** *simp*
qed
then show *?thesis* **unfolding** *eventually-sequentially* **by** *blast*
qed
have (λ*n*. *dist* (*u n*) *l*) ⟶ *0*
by (*simp add*: *less-le-trans* ∗ *order-tendstoI*)
then show *?thesis* **using** *LIMSEQ-dist-iff* **by** *auto*
qed

end

3.1 The Construction with Classes

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

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

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*) ⟶ *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*
by *unfold-locales* (*simp-all add*: *restriction-σ-tendsto-zero' dist-restriction-is'*)

end

setup ‹*Sign.add-const-constraint* (***const-name*** ‹*dist*›, *SOME* ***typ*** ‹*'a* :: *metric-space* ⇒ *'a* ⇒ *real*›)›
— Only allow *dist* in class *metric-space* (back to normal).

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 *unfold-locales*
show ‹*dist x y* = *0* ⟷ *x* = *y*› **for** *x y*
by (*simp add*: *restriction-dist-eq-0-iff-related*)

have *dist-commute* : ‹*dist x y* = *dist y x*› **for** *x y*
by (*simp add*: *dist-restriction-is*) *metis*

show ‹*dist x y* ≤ *max* (*dist x z*) (*dist y z*)› **for** *x y z*
proof −
consider ‹*x* ≠ *y*› **and** ‹*y* ≠ *z*› **and** ‹*x* ≠ *z*› | ‹*x* = *y* ∨ *y* = *z* ∨ *x* = *z*› **by** *blast*

thus ‹*dist x y* ≤ *max* (*dist x z*) (*dist y z*)›
proof *cases*
assume ‹*x* ≠ *y*› **and** ‹*y* ≠ *z*› **and** ‹*x* ≠ *z*›
from *this*(*1*)[*THEN not-related-imp-dist-restriction-le-some-restriction-σ*] *this*(*2*, *3*)[*THEN not-related-imp-dist-restriction-is-some-restriction-σ*]
obtain *l m n*
where ∗ : ‹*dist x y* ≤ $\sigma_\downarrow$ *TYPE*('*a*) *l*› ‹*x* ↓ *Suc l* ≠ *y* ↓ *Suc l*›
and ∗∗ : ‹*dist y z* = $\sigma_\downarrow$ *TYPE*('*a*) *m*› ‹∀ *k*≤*m*. *y* ↓ *k* = *z* ↓ *k*›
and ∗∗∗ : ‹*dist x z* = $\sigma_\downarrow$ *TYPE*('*a*) *n*› ‹∀ *k*≤*n*. *x* ↓ *k* = *z* ↓ *k*›
by *blast*
have ‹*min n m* ≤ *l*›
proof (*rule ccontr*)
assume ‹¬ *min n m* ≤ *l*›
hence ‹*Suc l* ≤ *n*› **and** ‹*Suc l* ≤ *m*› **by** *simp-all*
with ∗∗(*2*) ∗∗∗(*2*) *Suc-le-lessD*
have ‹*x* ↓ *Suc l* = *z* ↓ *Suc l*› ‹*y* ↓ *Suc l* = *z* ↓ *Suc l*› **by** *simp-all*
hence ‹*x* ↓ *Suc l* = *y* ↓ *Suc l*› **by** *simp*
with ‹*x* ↓ *Suc l* ≠ *y* ↓ *Suc l*› **show** *False* **..**
qed

have ‹*dist x y* ≤ $\sigma_\downarrow$ *TYPE*('*a*) (*min n m*)›
by (*rule restriction-space-Inf-properties*(*1*))

```
        (simp add: **(2) ***(2))
    also have ‹... ≤ max (dist x z) (dist y z)›
      unfolding **(1) ***(1) by linarith
    finally show ‹dist x y ≤ max (dist x z) (dist y z)› .
  next
    show ‹x = y ∨ y = z ∨ x = z ⟹ dist x y ≤ max (dist x z) (dist
y z)›
      by (elim disjE, simp-all add: dist-commute)
        (metis not-related-imp-dist-restriction-is-some-restriction-σ
      restriction-dist-eq-0-iff-related zero-le-restriction-σ dual-order.refl)
  qed
 qed
qed

end

context non-decseq-restriction-space begin


lemma restriction-tendsto-self: ‹(λn. x ↓ n) ⟶ x›
proof −
 have ‹(λn. dist (x ↓ n) x) ⟶ 0›
 proof (rule real-tendsto-sandwich[OF - ])
   show ‹∀⇩F n in sequentially. 0 ≤ dist (x ↓ n) x› by simp
 next
   show ‹∀⇩F n in sequentially. dist (x ↓ n) x ≤ σ↓ TYPE('a) n›
   by (auto intro: eventually-sequentiallyI[OF restriction-space-Inf-properties(1)])
 next
   show ‹(λn. 0) ⟶ 0› by simp
 next
  show ‹σ↓ TYPE('a) ⟶ 0› by (simp add: restriction-σ-tendsto-zero)
 qed
 thus ‹(λn. x ↓ n) ⟶ x›
  by (subst tendsto-iff[of ‹(λn. x ↓ n)›], subst eventually-sequentially)
    (simp add: LIMSEQ-iff)
qed

end




class decseq-restriction-space = non-decseq-restriction-space +
 assumes decseq-restriction-σ' : ‹decseq (σ↓ TYPE('a))›
begin

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

by *unfold-locales* (*simp add*: *decseq-restriction-σ'*)

— Removing $x \in M$ and $y \in M$.
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 *decseq* ($\sigma_{\downarrow}$ *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*
by *unfold-locales*
(*fact strict-decseq-imp-decseq*[*OF strict-decseq-restriction-σ*])

end

Generic Properties

lemma (**in** *metric-space*) *dist-sequences-tendsto-zero-imp-tendsto-iff* :
‹(λ*n*. *dist* (*σ n*) (*ψ n*)) ⟶ *0* ⟹ *σ* ⟶ Σ ⟷ *ψ* ⟶ Σ›
proof (*rule iffI*)
show ‹*ψ* ⟶ Σ› **if** ‹*σ* ⟶ Σ› **and** ‹(λ*n*. *dist* (*σ n*) (*ψ n*)) ⟶ *0*› **for** *σ ψ*
proof −
from *that* **have** ∗ : ‹(λ*n*. *dist* (*σ n*) (*ψ n*) + *dist* (*σ n*) Σ) ⟶ *0*›
unfolding *LIMSEQ-dist-iff* **using** *tendsto-add-zero* **by** *blast*
have ‹(λ*n*. *dist* (*ψ n*) Σ) ⟶ *0*›
by (*rule real-tendsto-sandwich*
[*of* ‹λ*n*. *0*› ‹λ*n*. *dist* (*ψ n*) Σ› - ‹λ*n*. *dist* (*σ n*) (*ψ n*) + *dist* (*σ n*) Σ›])
(*simp-all add*: ∗ *dist-triangle3*)
with *LIMSEQ-dist-iff* **show** ‹*ψ* ⟶ Σ› **by** *blast*
qed
thus ‹(λ*n*. *dist* (*σ n*) (*ψ n*)) ⟶ *0* ⟹ *ψ* ⟶ Σ ⟹ *σ* ⟶ Σ›
by (*simp add*: *dist-commute*)
qed

lemma (**in** *non-decseq-restriction-space*) *restricted-sequence-tendsto-iff*
:
‹$(\lambda n.\ \sigma\ n \downarrow n) \longrightarrow \Sigma \longleftrightarrow \sigma \longrightarrow \Sigma$›
proof −
have ‹$(\lambda n.\ dist\ (\sigma\ n \downarrow n)\ (\sigma\ n)) \longrightarrow 0$›
proof (*unfold metric-space-class.LIMSEQ-def*, *intro allI impI*)
fix ε :: *real* **assume** ‹$0 < \varepsilon$›
from *restriction-σ-tendsto-zero*[*unfolded metric-space-class.LIMSEQ-def*, *rule-format*, *OF* ‹$0 < \varepsilon$›]
obtain *no* **where** ‹$\forall\, n{\geq}no.\ \sigma_{\downarrow}\ TYPE('a)\ n < \varepsilon$›
by (*auto simp add*: *dist-real-def*)
have ‹$no \leq n \Longrightarrow dist\ (dist\ (\sigma\ n \downarrow n)\ (\sigma\ n))\ 0 < \varepsilon$› **for** *n*
by (*simp*, *rule le-less-trans*[*OF restriction-space-Inf-properties*(*1*)[*of* ‹$\sigma_{\downarrow}\ TYPE('a)\ n$›]])
(*simp*, *simp add*: ‹$\forall\, n{\geq}no.\ \sigma_{\downarrow}\ TYPE('a)\ n < \varepsilon$›)
thus ‹$\exists\, no.\ \forall\, n{\geq}no.\ dist\ (dist\ (\sigma\ n \downarrow n)\ (\sigma\ n))\ 0 < \varepsilon$› **by** *blast*
qed

thus ‹$(\lambda n.\ \sigma\ n \downarrow n) \longrightarrow \Sigma \longleftrightarrow \sigma \longrightarrow \Sigma$›
by (*simp add*: *dist-sequences-tendsto-zero-imp-tendsto-iff*)
qed

lemma (**in** *non-decseq-restriction-space*) *Cauchy-restriction-chain* :
‹*Cauchy* σ› **if** ‹$chain_{\downarrow}\ \sigma$›
proof (*rule metric-CauchyI*)
fix ε :: *real* **assume** ‹$0 < \varepsilon$›
from *LIMSEQ-D*[*OF restriction-σ-tendsto-zero* ‹$0 < \varepsilon$›, *simplified*]
obtain *M* **where** ‹$\sigma_{\downarrow}\ TYPE('a)\ M < \varepsilon$› **by** *blast*
moreover have ‹$M \leq m \Longrightarrow M \leq n \Longrightarrow dist\ (\sigma\ m)\ (\sigma\ n) \leq \sigma_{\downarrow}\ TYPE('a)\ M$› **for** *m n*
by (*rule restriction-space-Inf-properties*(*1*), *simp add*: *image-iff*)
(*metis restriction-chain-def-ter* ‹$chain_{\downarrow}\ \sigma$›)
ultimately have ‹$\forall\, m{\geq}M.\ \forall\, n{\geq}M.\ dist\ (\sigma\ m)\ (\sigma\ n) < \varepsilon$›
by (*meson dual-order.strict-trans2*)
thus ‹$\exists\, M.\ \forall\, m{\geq}M.\ \forall\, n{\geq}M.\ dist\ (\sigma\ m)\ (\sigma\ n) < \varepsilon$› **..**
qed

lemma (**in** *non-decseq-restriction-space*) *restriction-tendsto-imp-tendsto*
:
‹$\sigma \longrightarrow \Sigma$› **if** ‹$\sigma\ {-}{\downarrow}{\rightarrow}\ \Sigma$›
proof (*rule metric-LIMSEQ-I*)
fix ε :: *real* **assume** ‹$0 < \varepsilon$›
with *restriction-σ-tendsto-zero*[*THEN LIMSEQ-D*]
obtain *n0* **where** ‹$\forall\, k{\geq}n0.\ \sigma_{\downarrow}\ TYPE('a)\ k < \varepsilon$› **by** *auto*

hence ‹$\sigma_\downarrow$ *TYPE*('*a*) *n0* < ε› **by** *simp*
from *restriction-tendstoD*[*OF* ‹σ −↓→ Σ›]
obtain *n1* **where** ‹$\forall$ *k*≥*n1*. Σ ↓ *n0* = σ *k* ↓ *n0*› **..**
hence ‹$\forall$ *k*≥*n1*. *dist* (σ *k*) Σ ≤ $\sigma_\downarrow$ *TYPE*('*a*) *n0*›
by (*simp add*: *restriction-space-Inf-properties*(*1*))
with ‹$\sigma_\downarrow$ *TYPE*('*a*) *n0* < ε›
have ‹$\forall$ *k*≥*n1*. *dist* (σ *k*) Σ < ε› **by** (*meson order-le-less-trans*)
thus ‹$\exists$ *n1*. $\forall$ *k*≥*n1*. *dist* (σ *k*) Σ < ε› **..**
qed

In Decseq Restriction Space

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

lemma *le-dist-to-restriction-eqE* :
obtains *k* **where** ‹*n* ≤ *k*› ‹$\bigwedge$*x y* :: '*a*. *dist x y* ≤ $\sigma_\downarrow$ *TYPE*('*a*) *k* ⟹ *x* ↓ *n* = *y* ↓ *n*›
proof −
have ‹$\exists$ *k*≥*n*. $\forall$ *x y* :: '*a*. *dist x y* ≤ $\sigma_\downarrow$ *TYPE*('*a*) *k* ⟶ *x* ↓ *n* = *y* ↓ *n*›
proof (*rule ccontr*)
assume ‹¬ ($\exists$ *k*≥*n*. $\forall$ *x y* :: '*a*. *dist x y* ≤ $\sigma_\downarrow$ *TYPE*('*a*) *k* ⟶ *x* ↓ *n* = *y* ↓ *n*)›
hence ‹$\forall$ *k*≥*n*. $\exists$ *x y* :: '*a*. *dist x y* ≤ $\sigma_\downarrow$ *TYPE*('*a*) *k* ∧ *x* ↓ *n* ≠ *y* ↓ *n*› **by** *simp*
then obtain *X Y* :: ‹*nat* ⇒ '*a*›
where ∗ : ‹$\forall$ *k*≥*n*. *dist* (*X k*) (*Y k*) ≤ $\sigma_\downarrow$ *TYPE*('*a*) *k*›
‹$\forall$ *k*≥*n*. *X k* ↓ *n* ≠ *Y k* ↓ *n*› **by** *metis*
moreover obtain *n0* **where** ‹$\forall$ *k*≥*n0*. $\sigma_\downarrow$ *TYPE*('*a*) *k* < $\sigma_\downarrow$ *TYPE*('*a*) *n*›
by (*metis LIMSEQ-D zero-less-restriction-σ abs-of-nonneg diff-zero restriction-σ-tendsto-zero real-norm-def zero-le-restriction-σ*)
ultimately have ‹$\forall$ *k*≥*n*+*n0*. *dist* (*X k*) (*Y k*) < $\sigma_\downarrow$ *TYPE*('*a*) *n*›
by (*metis dual-order.strict-trans2 add-leE*)
moreover from ∗(*2*) **have** ‹$\forall$ *k*≥*n*. $\sigma_\downarrow$ *TYPE*('*a*) *n* ≤ *dist* (*X k*) (*Y k*)›
by (*simp add*: *dist-restriction-is-bis*)
(*metis* (*full-types*) *decseq-restriction-σ*[*THEN decseqD*] *linorder-linear not-eq-imp-dist-restriction-is-restriction-σ-Sup-restriction-eq-simplified*(*2*))
ultimately show *False* **by** (*metis le-add1 linorder-not-le order-refl*)
qed
thus ‹(⋀*k*. ⟦*n* ≤ *k*; ⋀*x y* :: '*a*. *dist x y* ≤ $\sigma_\downarrow$ *TYPE*('*a*) *k* ⟹ *x* ↓ *n* = *y* ↓ *n*⟧ ⟹ *thesis*) ⟹ *thesis*› **by** *blast*
qed

theorem *tendsto-iff-restriction-tendsto* : ‹σ ⟶⟶ Σ ⟷ σ −↓→ Σ›
proof (*rule iffI*)

show ‹σ $-\downarrow\rightarrow$ $\Sigma \Longrightarrow \sigma \longrightarrow \Sigma$› **by** (*fact restriction-tendsto-imp-tendsto*)
next
show ‹σ $-\downarrow\rightarrow$ Σ› **if** ‹$\sigma \longrightarrow \Sigma$›
proof (*rule restriction-tendstoI*)
fix n
obtain *n0* **where** ‹*dist* x $y \leq \sigma_{\downarrow}$ $TYPE('a)$ *n0* $\Longrightarrow$ $x \downarrow n = y \downarrow n$› **for** x y :: $'a$
by (*metis le-dist-to-restriction-eqE*)
moreover from *metric-LIMSEQ-D*[*OF* ‹$\sigma \longrightarrow \Sigma$› *zero-less-restriction-σ*]
obtain *n1* **where** ‹$\forall\, k{\geq}n1.$ *dist* (σ k) $\Sigma < \sigma_{\downarrow}$ $TYPE('a)$ *n0*› **..**
ultimately have ‹$\forall\, k{\geq}n1.\ \Sigma \downarrow n = \sigma\ k \downarrow n$› **by** (*metis dual-order.order-iff-strict*)
thus ‹$\exists\, n1.\ \forall\, k{\geq}n1.\ \Sigma \downarrow n = \sigma\ k \downarrow n$› **..**
qed
qed

corollary *convergent-iff-restriction-convergent* : ‹*convergent* σ $\longleftrightarrow$ *convergent*$_{\downarrow}$ σ›
by (*simp add*: *convergent-def restriction-convergent-def tendsto-iff-restriction-tendsto*)

theorem *complete-iff-restriction-complete* :
‹($\forall\, \sigma.$ *Cauchy* $\sigma \longrightarrow$ *convergent* σ) $\longleftrightarrow$ ($\forall\, \sigma.$ *chain*$_{\downarrow}$ $\sigma \longrightarrow$ *convergent*$_{\downarrow}$ σ)›
— The following result shows that we have not lost anything with our definitions of convergence, completeness, etc. specific to restriction spaces.
proof (*intro iffI impI allI*)
fix σ **assume** *hyp* : ‹$\forall\, \sigma.$ *Cauchy* $\sigma \longrightarrow$ *convergent* σ› **and** ‹*chain*$_{\downarrow}$ σ›
from *Cauchy-restriction-chain* ‹*chain*$_{\downarrow}$ σ› **have** ‹*Cauchy* σ› **by** *blast*
hence ‹*convergent* σ› **by** (*simp add*: *hyp*)
thus ‹*convergent*$_{\downarrow}$ σ› **by** (*simp add*: *convergent-iff-restriction-convergent*)
next
fix σ **assume** *hyp* : ‹$\forall\, \sigma.$ *chain*$_{\downarrow}$ $\sigma \longrightarrow$ *convergent*$_{\downarrow}$ σ› **and** ‹*Cauchy* σ›
from ‹*Cauchy* σ› **have** $*$: ‹$\forall\, n.\ \exists\, k.\ \forall\, l{\geq}k.\ \sigma\ l \downarrow n = \sigma\ k \downarrow n$›
by (*metis* (*mono-tags*, *opaque-lifting*) *Cauchy-altdef2 dual-order.order-iff-strict le-dist-to-restriction-eqE zero-less-restriction-σ*)
define f **where** ‹$f \equiv$ *rec-nat*
(*LEAST k.* $\forall\, l{\geq}k.\ \sigma\ l \downarrow 0 = \sigma\ k \downarrow 0$)
($\lambda n\ k.$ *LEAST l.* $k < l \wedge (\forall\, m{\geq}l.\ \sigma\ m \downarrow$ *Suc* $n = \sigma\ l \downarrow$ *Suc* n))›
have *f-Suc-def* : ‹f (*Suc* n) = (*LEAST l.* $f\ n < l \wedge (\forall\, m{\geq}l.\ \sigma\ m \downarrow$ *Suc* $n = \sigma\ l \downarrow$ *Suc* n))›
(**is** ‹f (*Suc* n) = *Least* (*?f-Suc* n)›) **for** n **by** (*simp add*: *f-def*)
from $*$ **have** $**$: ‹$\exists\, k{>}f\ n.\ \forall\, m{\geq}k.\ \sigma\ m \downarrow$ *Suc* $n = \sigma\ k \downarrow$ *Suc* n› **for** n
by (*metis dual-order.trans lessI linorder-not-le order-le-less*)

```
  have ‹strict-mono f›
  proof (unfold strict-mono-Suc-iff, rule allI)
    show ‹f n < f (Suc n)› for n
      by (fact LeastI-ex[of ‹?f-Suc n›, folded f-Suc-def, OF **, THEN
conjunct1])
  qed

  have ‹chain↓ (λn. (σ ∘ f) n ↓ n)› — key point
  proof (rule restriction-chainI)
    fix n
    have ‹σ (f (Suc n)) ↓ n = σ (f n) ↓ n›
    proof (cases n)
      show ‹n = 0 ⟹ σ (f (Suc n)) ↓ n = σ (f n) ↓ n› by simp
    next
      show ‹n = Suc nat ⟹ σ (f (Suc n)) ↓ n = σ (f n) ↓ n› for nat
      proof (clarify, intro LeastI-ex[of ‹?f-Suc nat›,
            folded f-Suc-def, THEN conjunct2, rule-format, OF **])
        show ‹n = Suc nat ⟹ f (Suc nat) ≤ f (Suc (Suc nat))›
          by (simp add: ‹strict-mono f› strict-mono-less-eq)
      qed
    qed
    thus ‹(σ ∘ f) (Suc n) ↓ Suc n ↓ n = (σ ∘ f) n ↓ n› by simp
  qed
  with hyp have ‹convergent↓ (λn. (σ ∘ f) n ↓ n)› by simp
  hence ‹convergent↓ (λn. (σ ∘ f) n)›
   by (simp add: restriction-convergent-restricted-iff-restriction-convergent)
  hence ‹convergent (λn. (σ ∘ f) n)›
    by (simp add: convergent-iff-restriction-convergent)
  with Cauchy-converges-subseq[OF ‹Cauchy σ› ‹strict-mono f›]
  show ‹convergent σ› unfolding convergent-def by blast
qed

end
```

The following classes will be useful later.

```
class complete-decseq-restriction-space = decseq-restriction-space +
  assumes restriction-chain-imp-restriction-convergent' : ‹chain↓ σ
⟹ convergent↓ σ›
begin

subclass complete-restriction-space
  by unfold-locales (fact restriction-chain-imp-restriction-convergent')

subclass complete-ultrametric-space
proof (unfold-locales)
 from complete-iff-restriction-complete restriction-chain-imp-restriction-convergent
  show ‹Cauchy σ ⟹ convergent σ› for σ by blast
qed
```

end

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

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*
 by (*unfold-locales*) (*fact restriction-chain-imp-restriction-convergent'*)

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*] : ‹$0 \leq \delta_{\downarrow}\ TYPE('a)\ \widehat{}\ n$›
  by (*simp-all add*: *order-less-imp-le*)

lemma *pow-restriction-δ-le-one* [*simp*] : ‹$\delta_{\downarrow}\ TYPE('a)\ \widehat{}\ n \leq 1$›
  by (*simp add*: *power-le-one*)

lemma *pow-restriction-δ-less-one* [*simp*] : ‹$n \neq 0 \Longrightarrow \delta_{\downarrow}\ TYPE('a)\ \widehat{}\ n < 1$›
  by (*metis restriction-δ-less-one zero-less-restriction-δ not-gr-zero power-0 power-strict-decreasing*)

end

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

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'* :
‹$dist\ x\ y = (INF\ n \in \{n.\ x \downarrow n = y \downarrow n\}.\ \sigma_\downarrow\ TYPE('a)\ n)$›

setup ‹*Sign.add-const-constraint* (***const-name***‹*dist*›, *SOME* ***typ***‹$'a$ :: *metric-space* ⇒ $'a$ ⇒ *real*›)›
— Only allow *dist* in class *metric-space* (back to normal).

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

lemma *restriction-σ-le-restriction-σ-times-pow-restriction-δ* :
‹$\sigma_\downarrow\ TYPE('a)\ (n + k) \leq \sigma_\downarrow\ TYPE('a)\ n * \delta_\downarrow\ TYPE('a)\ \hat{}\ k$›
by (*induct k*, *simp-all*)
(*metis dual-order.trans restriction-σ-le zero-less-restriction-δ*
mult.left-commute mult-le-cancel-left-pos)

lemma *restriction-σ-le-pow-restriction-δ* :
‹$\sigma_\downarrow\ TYPE('a)\ n \leq \sigma_\downarrow\ TYPE('a)\ 0 * \delta_\downarrow\ TYPE('a)\ \hat{}\ n$›
by (*metis add-0 restriction-σ-le-restriction-σ-times-pow-restriction-δ*)

subclass *strict-decseq-restriction-space*
proof *unfold-locales*
have ‹$\forall_F\ n$ *in sequentially*. $0 \leq \sigma_\downarrow\ TYPE('a)\ n$›
by (*simp add*: *zero-less-restriction-σ' order-less-imp-le*)
moreover have ‹$\forall_F\ n$ *in sequentially*. $\sigma_\downarrow\ TYPE('a)\ n \leq \sigma_\downarrow\ TYPE('a)\ 0 * \delta_\downarrow\ TYPE('a)\ \hat{}\ n$›
by (*simp add*: *restriction-σ-le-pow-restriction-δ*)
moreover have ‹$(\lambda n.\ 0) \longrightarrow 0$› **by** *simp*
moreover have ‹$(\lambda n.\ \sigma_\downarrow\ TYPE('a)\ 0 * \delta_\downarrow\ TYPE('a)\ \hat{}\ n) \longrightarrow 0$›
by (*simp add*: *LIMSEQ-abs-realpow-zero2 abs-of-pos tendsto-mult-right-zero*)
ultimately show ‹$\sigma_\downarrow\ TYPE('a) \longrightarrow 0$› **by** (*fact real-tendsto-sandwich*)
next
show ‹$0 < \sigma_\downarrow\ TYPE('a)\ n$› **for** *n*
by (*fact zero-less-restriction-σ'*)
next
show ‹$dist\ x\ y = Inf\ (\sigma_\downarrow\ TYPE('a)\ `\ \{n.\ x \downarrow n = y \downarrow n\})$› **for** *x y*
by (*fact dist-restriction-is'*)
next
show ‹*strict-decseq* ($\sigma_\downarrow\ TYPE('a)$)›
by (*rule strict-decseqI*, *rule le-less-trans*[*OF restriction-σ-le*])
(*simp add*: *zero-less-restriction-σ'*)
qed

lemma ‹$0 < \delta_\downarrow\ TYPE('a) \ \widehat{}\ n$› **by** *simp*

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′* : ‹$chain_\downarrow\ \sigma \Longrightarrow convergent_\downarrow\ \sigma$›
begin

subclass *complete-strict-decseq-restriction-space*
  by *unfold-locales* (*fact restriction-chain-imp-restriction-convergent′*)

end

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

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

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

class *geometric-restriction-space = uniformity-dist + open-uniformity* + *restriction-δ* +
  assumes *restriction-σ-is* : ‹$\sigma_\downarrow\ TYPE('a)\ n = \delta_\downarrow\ TYPE('a) \ \widehat{}\ n$›
    and *dist-restriction-is′* : ‹$dist\ x\ y = (INF\ n \in \{n.\ x \downarrow n = y \downarrow n\}.\ \sigma_\downarrow\ TYPE('a)\ n)$›
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 *unfold-locales*
  show ‹$0 < \sigma_\downarrow\ TYPE('a)\ n$› **for** *n* **by** (*simp add*: *restriction-σ-is*)
next

show ‹$\sigma_\downarrow$ *TYPE*('*a*) (*Suc n*) $\leq$ $\delta_\downarrow$ *TYPE*('*a*) $*$ $\sigma_\downarrow$ *TYPE*('*a*) *n*› **for** *n*
by (*simp add*: *restriction-σ-is*)
next
show ‹*dist x y* = *Inf* ($\sigma_\downarrow$ *TYPE*('*a*) ' {*n*. *x* $\downarrow$ *n* = *y* $\downarrow$ *n*})› **for** *x y*
by (*fact dist-restriction-is'*)
qed

lemma ‹0 < $\delta_\downarrow$ *TYPE*('*a*) ^ *n*› **by** *simp*

end

setup ‹*Sign.add-const-constraint* (***const-name***‹*dist*›, *SOME* ***typ***‹'*a* :: *metric-space* ⇒ '*a* ⇒ *real*›)›
— Only allow *dist* in class *metric-space* (back to normal).

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*$_\downarrow$ *σ* ⟹ *convergent*$_\downarrow$ *σ*›
begin

subclass *complete-at-least-geometric-restriction-space*
by (*unfold-locales*) (*fact restriction-chain-imp-restriction-convergent'*)

end

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

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

theorem *geometric-restriction-space-completeI* : ‹*convergent σ*›
if ‹⋀*σ* :: *nat* ⇒ '*a*. *restriction-chain σ* ⟹ ∃ Σ. ∀ *n*. Σ $\downarrow$ *n* = *σ n*›
and ‹*Cauchy σ*› **for** *σ* :: ‹*nat* ⇒ '*a* :: *geometric-restriction-space*›
by (*metis complete-iff-restriction-complete convergent-def*
convergent-iff-restriction-convergent ext restriction-tendsto-self that)

— Because *cball* is not defined in *metric-space*.
lemma (**in** *non-decseq-restriction-space*) *restriction-cball-subset-cball* :

‹$\sigma_\downarrow$ *TYPE*('*a*) $n \leq r \Longrightarrow \mathcal{B}_\downarrow(\Sigma,\, n) \subseteq \{x.\ dist\ \Sigma\ x \leq r\}$›
by (*simp add*: *subset-iff restriction-cball-mem-iff*)
(*simp add*: *dual-order.trans restriction-space-Inf-properties*(*1*))

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

lemma (**in** *non-decseq-restriction-space*) *restriction-ball-subset-ball* :
‹$\sigma_\downarrow$ *TYPE*('*a*) $n < r \Longrightarrow$ *restriction-ball* $\Sigma\ n \subseteq \{x.\ dist\ \Sigma\ x < r\}$›
by (*simp add*: *subset-iff restriction-cball-mem-iff*)
(*metis* (*mono-tags*, *lifting*) *image-eqI mem-Collect-eq order-le-less-trans*
restriction-related-pred restriction-space-Inf-properties(*1*))

corollary *restriction-ball-subset-ball-bis* :
‹$\sigma_\downarrow$ *TYPE*('*a*) $n < r \Longrightarrow$ *restriction-ball* $\Sigma\ n \subseteq$ *ball* $\Sigma\ r$›
for Σ :: ‹'*a* :: *non-decseq-restriction-space*›
unfolding *ball-def* **by** (*fact restriction-ball-subset-ball*)

lemma (**in** *strict-decseq-restriction-space*)
restriction-cball-is-cball : ‹$\mathcal{B}_\downarrow(\Sigma,\, n) = \{x.\ dist\ \Sigma\ x \leq \sigma_\downarrow$ *TYPE*('*a*) *n*}›
proof (*intro subset-antisym subsetI*)
from *restriction-cball-subset-cball*
show ‹$x \in \mathcal{B}_\downarrow(\Sigma,\, n) \Longrightarrow x \in \{x.\ dist\ \Sigma\ x \leq \sigma_\downarrow$ *TYPE*('*a*) *n*}› **for** *x* **by** *blast*
next
show ‹$x \in \mathcal{B}_\downarrow(\Sigma,\, n)$› **if** ‹$x \in \{x.\ dist\ \Sigma\ x \leq \sigma_\downarrow$ *TYPE*('*a*) *n*}› **for** *x*
proof (*cases* ‹$\Sigma = x$›)
show ‹$\Sigma = x \Longrightarrow x \in \mathcal{B}_\downarrow(\Sigma,\, n)$› **by** *simp*
next
assume ‹$\Sigma \neq x$›
with *not-related-imp-dist-restriction-is-some-restriction-*σ **obtain** *m*
where ‹*dist* $\Sigma\ x = \sigma_\downarrow$ *TYPE*('*a*) *m*› ‹$\forall\, k{\leq}m.\ \Sigma \downarrow k = x \downarrow k$› **by** *blast*
from ‹$x \in \{x.\ dist\ \Sigma\ x \leq \sigma_\downarrow$ *TYPE*('*a*) *n*}›
have ‹*dist* $\Sigma\ x \leq \sigma_\downarrow$ *TYPE*('*a*) *n*› **by** *simp*
with ‹*dist* $\Sigma\ x = \sigma_\downarrow$ *TYPE*('*a*) *m*› **have** ‹$n \leq m$›
by (*metis linorder-not-less strict-decseq-restriction-*σ *strict-decseq-def-bis*)
with ‹$\forall\, k{\leq}m.\ \Sigma \downarrow k = x \downarrow k$› **have** ‹$\Sigma \downarrow n = x \downarrow n$› **by** *simp*
thus ‹$x \in \mathcal{B}_\downarrow(\Sigma,\, n)$› **by** (*simp add*: *restriction-cball-mem-iff*)
qed
qed

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

lemma (**in** *strict-decseq-restriction-space*)
restriction-ball-is-ball : ‹*restriction-ball* Σ n = $\{x.\ dist\ \Sigma\ x < \sigma_{\downarrow}\ TYPE('a)\ n\}$›
proof (*intro subset-antisym subsetI*)
show ‹$x \in$ *restriction-ball* Σ $n \Longrightarrow x \in \{x.\ dist\ \Sigma\ x < \sigma_{\downarrow}\ TYPE('a)\ n\}$› **for** x
by (*simp add*: *subset-iff*)
(*metis lessI local.restriction-cball-is-cball strict-decseq-restriction-σ mem-Collect-eq order-le-less-trans strict-decseq-def-bis*)
next
show ‹$x \in$ *restriction-ball* Σ n› **if** ‹$x \in \{x.\ dist\ \Sigma\ x < \sigma_{\downarrow}\ TYPE('a)\ n\}$› **for** x
proof (*cases* ‹$\Sigma = x$›)
show ‹$\Sigma = x \Longrightarrow x \in$ *restriction-ball* Σ n› **by** *simp*
next
assume ‹$\Sigma \neq x$›
with *not-related-imp-dist-restriction-is-some-restriction-σ* **obtain** m
where ‹*dist* Σ $x = \sigma_{\downarrow}\ TYPE('a)\ m$› ‹$\forall k{\leq}m.\ \Sigma \downarrow k = x \downarrow k$› **by** *blast*
from ‹$x \in \{x.\ dist\ \Sigma\ x < \sigma_{\downarrow}\ TYPE('a)\ n\}$›
have ‹*dist* Σ $x < \sigma_{\downarrow}\ TYPE('a)\ n$› **by** *simp*
with ‹*dist* Σ $x = \sigma_{\downarrow}\ TYPE('a)\ m$› **have** ‹$n < m$›
using *strict-decseq-restriction-σ strict-decseq-def-bis* **by** *auto*
with ‹$\forall k{\leq}m.\ \Sigma \downarrow k = x \downarrow k$› **have** ‹$\Sigma \downarrow Suc\ n = x \downarrow Suc\ n$› **by** *simp*
thus ‹$x \in$ *restriction-ball* Σ n› **by** (*simp add*: *restriction-cball-mem-iff*)
qed
qed

lemma *restriction-ball-is-ball-bis* : ‹*restriction-ball* Σ n = *ball* Σ $(\sigma_{\downarrow}\ TYPE('a)\ n)$›
for Σ :: ‹$'a$ :: *strict-decseq-restriction-space*›
by (*simp add*: *ball-def restriction-ball-is-ball*)

lemma *isCont-iff-restriction-cont-at* : ‹*isCont* f $\Sigma \longleftrightarrow$ *restriction-cont-at* f Σ›
for f :: ‹$'a$ :: *decseq-restriction-space* ⇒ $'b$ :: *decseq-restriction-space*›
by (*unfold restriction-cont-at-def continuous-at-sequentially comp-def*,

fold tendsto-iff-restriction-tendsto) *simp*

lemma (**in** *strict-decseq-restriction-space*)
open-iff-restriction-open : ‹*open* $U \longleftrightarrow$ *open*$_\downarrow$ U›
proof (*unfold open-dist restriction-open-iff-restriction-cball-characterization*, *intro iffI ballI*)
fix Σ **assume** ‹$\forall\, x{\in}U.\ \exists\, e{>}0.\ \forall\, y.$ *dist* $y\ x < e \longrightarrow y \in U$› **and** ‹$\Sigma \in U$›
then obtain e **where** ‹$0 < e$› ‹$\forall\, y.$ *dist* $y\ \Sigma < e \longrightarrow y \in U$› **by** *blast*
from ‹$0 < e$› **obtain** n **where** ‹$\sigma_\downarrow$ *TYPE*($'a$) $n < e$›
by (*metis eventually-at-top-linorder le-refl restriction-σ-tendsto-zero order-tendstoD*(*2*))
with ‹$\forall\, y.$ *dist* $y\ \Sigma < e \longrightarrow y \in U$› *dist-commute restriction-cball-is-cball*
have ‹$\mathcal{B}_\downarrow(\Sigma,\ n) \subseteq U$› **by** *auto*
thus ‹$\exists\, n.\ \mathcal{B}_\downarrow(\Sigma,\ n) \subseteq U$› **..**
next
fix x **assume** ‹$\forall\, \Sigma{\in}U.\ \exists\, n.\ \mathcal{B}_\downarrow(\Sigma,\ n) \subseteq U$› ‹$x \in U$›
then obtain n **where** ‹$\mathcal{B}_\downarrow(x,\ n) \subseteq U$› **by** *blast*
hence ‹$\forall\, y.$ *dist* $y\ x < \sigma_\downarrow$ *TYPE*($'a$) $n \longrightarrow y \in U$›
by (*simp add*: *dist-commute restriction-cball-is-cball subset-iff*)
thus ‹$\exists\, e{>}0.\ \forall\, y.$ *dist* $y\ x < e \longrightarrow y \in U$›
using *zero-less-restriction-σ′* **by** *blast*
qed

lemma (**in** *strict-decseq-restriction-space*)
closed-iff-restriction-closed : ‹*closed* $U \longleftrightarrow$ *closed*$_\downarrow$ U›
by (*simp add*: *closed-open open-iff-restriction-open restriction-open-Compl-iff*)

lemma *continuous-on-iff-restriction-cont-on* :
‹*open* $U \Longrightarrow$ *continuous-on* $U\ f \longleftrightarrow$ *restriction-cont-on* $f\ U$›
for f :: ‹$'a$ :: *decseq-restriction-space* $\Rightarrow$ $'b$ :: *decseq-restriction-space*›
by (*simp add*: *restriction-cont-on-def continuous-on-eq-continuous-at flip*: *isCont-iff-restriction-cont-at*)

3.2 Equivalence between Lipschitz Map and Restriction shift Map

For a function $f{::}'a \Rightarrow 'b$, 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 (*rule lipschitz-with-onI*)
show ‹0 ≤ *restriction-δ TYPE*('*b*) ^ *k*› **by** *simp*
next
fix *x y* **assume** ‹*x* ∈ *A*› ‹*y* ∈ *A*› ‹*x* ≠ *y*› ‹*f x* ≠ *f y*›
from ‹*restriction-shift-on f k A*›[*THEN restriction-shift-onD*, *OF* ‹*x* ∈ *A*› ‹*y* ∈ *A*›]
have ‹*i* ∈ *restriction-related-set x y* ⟹ *i* + *k* ∈ *restriction-related-set* (*f x*) (*f y*)› **for** *i*
by (*simp add*: *nat-int-add*)
hence ‹*Sup* (*restriction-related-set x y*) + *k* ∈ *restriction-related-set* (*f x*) (*f y*)›
using ‹*x* ≠ *y*› *Sup-in-restriction-related-set* **by** *blast*
hence ‹*Sup* (*restriction-related-set x y*) + *k* ≤ *Sup* (*restriction-related-set* (*f x*) (*f y*))›
by (*simp add*: ‹*f x* ≠ *f y*› *bdd-above-restriction-related-set-iff cSup-upper*)
moreover have ‹*dist* (*f x*) (*f y*) = *restriction-σ TYPE*('*b*) (*Sup* (*restriction-related-set* (*f x*) (*f y*)))›
by (*simp add*: ‹*f x* ≠ *f y*› *dist-restriction-is-bis-simplified*)
ultimately have ‹*dist* (*f x*) (*f y*) ≤ *restriction-σ TYPE*('*b*) (*Sup* (*restriction-related-set x y*) + *k*)›
by (*simp add*: *decseqD decseq-restriction-space-class.decseq-restriction-σ*)
hence ‹*dist* (*f x*) (*f y*) ≤ *restriction-σ TYPE*('*b*) (*Sup* (*restriction-related-set x y*)) ∗ *restriction-δ TYPE*('*b*) ^ *k*›
by (*meson order.trans restriction-σ-le-restriction-σ-times-pow-restriction-δ*)
also have ‹... ≤ *restriction-σ TYPE*('*a*) (*Sup* (*restriction-related-set x y*)) ∗ *restriction-δ TYPE*('*b*) ^ *k*›
by (*simp add*: *le-restriction-σ*)
finally show ‹*dist* (*f x*) (*f y*) ≤ *restriction-δ TYPE*('*b*) ^ *k* ∗ *dist x y*›
by (*simp add*: *dist-restriction-is-bis*[*of x y*] ‹*x* ≠ *y*› *mult.commute*)
qed

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*›
using *restriction-shift-def restriction-shift-nonneg-on-imp-lipschitz-with-on*
by *blast*

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 (*rule restriction-shift-onI*, *goal-cases*)
fix *x y n* **assume** ‹*x* ∈ *A*› ‹*y* ∈ *A*› ‹*f x* ≠ *f y*› ‹*x* ↓ *n* = *y* ↓ *n*›
from ‹*f x* ≠ *f y*› **have** ‹*x* ≠ *y*› **by** *blast*
from ‹*lipschitz-with-on f* (*restriction-δ TYPE*('*b*) *powi* − *int k*) *A*›
[*THEN lipschitz-with-onD2*, *OF* ‹*x* ∈ *A*› ‹*y* ∈ *A*›]
have ‹*dist* (*f x*) (*f y*) ≤ *restriction-δ TYPE*('*b*) *powi* − *int k* ∗ *dist x y*› **.**
hence ‹*dist* (*f x*) (*f y*) ∗ *restriction-δ TYPE*('*b*) ^ *k* ≤ *dist x y*›
by (*subst* (*asm*) *mult.commute*)
(*drule mult-imp-div-pos-le*[*rotated*]; *simp add*: *power-int-minus-divide*)
hence ‹*restriction-σ TYPE*('*b*) (*Sup* (*restriction-related-set* (*f x*) (*f y*))) ∗ *restriction-δ TYPE*('*b*) ^ *k*
≤ *restriction-σ TYPE*('*b*) (*Sup* (*restriction-related-set x y*))›
by (*simp add*: *dist-restriction-is-bis* ‹*x* ≠ *y*› ‹*f x* ≠ *f y*›)
(*drule order-trans*[*OF - le-restriction-σ*], *simp*)
from *order-trans*[*OF restriction-σ-le-restriction-σ-times-pow-restriction-δ*
[*of* ‹*Sup* (*restriction-related-set* (*f x*) (*f y*))› *k*] *this*]
have ‹*restriction-σ TYPE*('*b*) (*Sup* (*restriction-related-set* (*f x*) (*f y*)) + *k*)
≤ *restriction-σ TYPE*('*b*) (*Sup* (*restriction-related-set x y*))› **.**
hence ‹*Sup* (*restriction-related-set x y*) ≤ *Sup* (*restriction-related-set* (*f x*) (*f y*)) + *k*›
using *strict-decseq-def-ter strict-decseq-restriction-σ* **by** *blast*
from *order-trans*[*OF cSup-upper this*] **have** ‹*n* ≤ *Sup* (*restriction-related-set* (*f x*) (*f y*)) + *k*›
by (*simp add*: ‹*x* ↓ *n* = *y* ↓ *n*› ‹*x* ≠ *y*› *bdd-above-restriction-related-set-iff*)
hence ‹*nat* (*int n* + − *int k*) ≤ *Sup* (*restriction-related-set* (*f x*) (*f y*))› **by** *linarith*
thus ‹*f x* ↓ *nat* (*int n* + − *int k*) = *f y* ↓ *nat* (*int n* + − *int k*)›
by (*metis not-eq-imp-dist-restriction-is-restriction-σ-Sup-restriction-eq-simplified*(*2*))
qed

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*›
using *lipschitz-with-on-imp-restriction-shift-neg-on restriction-shift-def*
by *blast*

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 (*rule iffI*)
— We could do a case on *k*, but both cases are actually handled by the proof required after applying ⟦*lipschitz-with-on ?f* (*δ*↓ *TYPE*(*?'b*) *powi* − *int ?k*) *?A*; ⋀*n*. *σ*↓ *TYPE*(*?'a*) *n* ≤ *σ*↓ *TYPE*(*?'b*) *n*⟧ ⟹ *restriction-shift-on ?f* (− *int ?k*) *?A*.
show ‹*restriction-shift-on f k A*› **if** ‹*lipschitz-with-on f* (*restriction-δ TYPE*('*b*) *powi k*) *A*›
proof (*rule restriction-shift-onI*)
fix *x y n* **assume** ‹*x* ∈ *A*› ‹*y* ∈ *A*› ‹*f x* ≠ *f y*› ‹*x* ↓ *n* = *y* ↓ *n*›
from ‹*f x* ≠ *f y*› **have** ‹*x* ≠ *y*› **by** *blast*
from ‹*lipschitz-with-on f* (*restriction-δ TYPE*('*b*) *powi k*) *A*›
[*THEN lipschitz-with-onD2*, *OF* ‹*x* ∈ *A*› ‹*y* ∈ *A*›]
have ‹*dist* (*f x*) (*f y*) ≤ *restriction-δ TYPE*('*b*) *powi k* ∗ *dist x y*› **.**
also have ‹*dist* (*f x*) (*f y*) = *restriction-δ TYPE*('*b*) ^ *Sup* (*restriction-related-set* (*f x*) (*f y*))›
by (*simp add*: *dist-restriction-is-bis* ‹*f x* ≠ *f y*› *restriction-σ-is*)
also have ‹*dist x y* = *restriction-δ TYPE*('*b*) ^*Sup* (*restriction-related-set x y*)›
by (*simp add*: *dist-restriction-is-bis* ‹*x* ≠ *y*› *restriction-σ-is same-restriction-σ*[*symmetric*])
finally have ‹*restriction-δ TYPE*('*b*) ^ *Sup* (*restriction-related-set* (*f x*) (*f y*))
≤ *restriction-δ TYPE*('*b*) *powi k* ∗ *restriction-δ TYPE*('*b*) ^ *Sup* (*restriction-related-set x y*)› **.**
also have ‹. . . = *restriction-δ TYPE*('*b*) *powi* (*k* + *Sup* (*restriction-related-set x y*))›
by (*subst power-int-add*, *metis order-less-irrefl zero-less-restriction-δ*) *simp*
finally have ‹*restriction-δ TYPE*('*b*) ^ *Sup* (*restriction-related-set* (*f x*) (*f y*)) ≤ . . .› **.**
moreover have ‹*restriction-δ TYPE*('*b*) *powi m* ≤ *restriction-δ TYPE*('*b*) *powi m'* ⟹ *m'* ≤ *m*› **for** *m m'*
by (*rule ccontr*, *unfold not-le*,
drule power-int-strict-decreasing[*OF - zero-less-restriction-δ restriction-δ-less-one*])
(*fold not-le*, *blast*)
ultimately have ‹*k* + *Sup* (*restriction-related-set x y*) ≤ *Sup* (*restriction-related-set* (*f x*) (*f y*))› **by** *simp*
hence ‹*Sup* (*restriction-related-set x y*) ≤ *Sup* (*restriction-related-set* (*f x*) (*f y*)) − *k*› **by** *simp*
with ‹*x* ↓ *n* = *y* ↓ *n*› ‹*x* ≠ *y*› *linorder-not-le*
not-eq-imp-dist-restriction-is-restriction-σ-Sup-restriction-eq-simplified(*3*)
have ‹*n* ≤ *Sup* (*restriction-related-set* (*f x*) (*f y*)) − *k*› **by** *fastforce*

hence ‹*nat* (*int n* + *k*) ≤ *Sup* (*restriction-related-set* (*f x*) (*f y*))›
by *simp*
thus ‹*f x* ↓ *nat* (*int n* + *k*) = *f y* ↓ *nat* (*int n* + *k*)›
by (*metis not-eq-imp-dist-restriction-is-restriction-σ-Sup-restriction-eq-simplified*(*2*))
qed
next
show ‹*lipschitz-with-on f* (*restriction-δ TYPE*('*b*) *powi k*) *A*› **if** ‹*restriction-shift-on f k A*›
proof (*cases k*)
show ‹*k* = *int k'* ⟹ *lipschitz-with-on f* (*restriction-δ TYPE*('*b*) *powi k*) *A*› **for** *k'*
by (*simp*, *rule restriction-shift-nonneg-on-imp-lipschitz-with-on*)
(*use* ‹*restriction-shift-on f k A*› *same-restriction-σ* **in** *simp-all*)
next
fix *k'* **assume** ‹*k* = − *int* (*Suc k'*)›
show ‹*lipschitz-with-on f* (*restriction-δ TYPE*('*b*) *powi k*) *A*›
proof (*rule lipschitz-with-onI*)
show ‹*0* ≤ *restriction-δ TYPE*('*b*) *powi k*› **by** *simp*
next
fix *x y* **assume** ‹*x* ∈ *A*› ‹*y* ∈ *A*› ‹*x* ≠ *y*› ‹*f x* ≠ *f y*›
have ‹*i* ∈ *restriction-related-set x y* ⟹ *i* − *Suc k'* ∈ *restriction-related-set* (*f x*) (*f y*)› **for** *i*
using ‹*restriction-shift-on f k A*›[*THEN restriction-shift-onD*, *OF* ‹*x* ∈ *A*› ‹*y* ∈ *A*›, *of i*]
by (*unfold* ‹*k* = − *int* (*Suc k'*)›, *clarify*) (*metis diff-conv-add-uminus nat-minus-as-int*)
hence ‹*Sup* (*restriction-related-set x y*) − *Suc k'* ∈ *restriction-related-set* (*f x*) (*f y*)›
using ‹*x* ≠ *y*› *not-eq-imp-dist-restriction-is-restriction-σ-Sup-restriction-eq-simplified*
by *blast*
hence ‹*Sup* (*restriction-related-set x y*) − *Suc k'* ≤ *Sup* (*restriction-related-set* (*f x*) (*f y*))›
by (*simp add*: ‹*f x* ≠ *f y*› *bdd-above-restriction-related-set-iff cSup-upper*)
hence ∗ : ‹*Sup* (*restriction-related-set x y*) + *k* ≤ *Sup* (*restriction-related-set* (*f x*) (*f y*))›
by (*simp add*: ‹*k* = − *int* (*Suc k'*)›)
have ‹*dist* (*f x*) (*f y*) = *restriction-δ TYPE*('*b*) *powi Sup* (*restriction-related-set* (*f x*) (*f y*))›
by (*simp add*: ‹*f x* ≠ *f y*› *dist-restriction-is-bis-simplified restriction-σ-is*)
also have ‹... ≤ *restriction-δ TYPE*('*b*) *powi* (*Sup* (*restriction-related-set x y*) + *k*)›
by (*rule power-int-decreasing*[*OF* ∗]; *simp*)
(*metis order-less-irrefl zero-less-restriction-δ*)
also have ‹... = *restriction-δ TYPE*('*b*) *powi k* ∗ *restriction-δ TYPE*('*b*) *powi* (*Sup* (*restriction-related-set x y*))›
by (*subst power-int-add*, *metis order-less-irrefl zero-less-restriction-δ*) *simp*

also have ‹*restriction-δ TYPE*('*b*) *powi* (*Sup* (*restriction-related-set x y*)) = *dist x y*›
by (*simp add*: ‹*x* ≠ *y*› *dist-restriction-is-bis-simplified restriction-σ-is same-restriction-σ*[*symmetric*])
finally show ‹*dist* (*f x*) (*f y*) ≤ *restriction-δ TYPE*('*b*) *powi k* ∗ *dist x y*› **.**
qed
qed
qed

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*›
by (*simp add*: *lipschitz-with-on-iff-restriction-shift-on restriction-shift-def*)

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 by *intro-classes*

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 by *intro-classes*
(*simp-all add*: *restriction-σ-fun-def dist-fun-def open-fun-def*
uniformity-fun-def restriction-σ-tendsto-zero)

end

instance ‹*fun*› :: (*type*, *decseq-restriction-space*) *decseq-restriction-space*
by *intro-classes* (*simp add*: *restriction-σ-fun-def decseq-restriction-σ*)

instance ‹*fun*› :: (*type*, *strict-decseq-restriction-space*) *strict-decseq-restriction-space*
by *intro-classes*
(*simp add*: *restriction-σ-fun-def strict-decseq-restriction-σ*)

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 by *intro-classes* (*simp-all add*: *restriction-δ-fun-def*)

end

instance ‹*fun*› :: (*type*, *at-least-geometric-restriction-space*) *at-least-geometric-restriction-space*
proof *intro-classes*
show ‹*0* < *restriction-σ TYPE*('*a* ⇒ '*b*) *n*› **for** *n*
by (*simp add*: *restriction-σ-fun-def*)
next
show ‹*restriction-σ TYPE*('*a* ⇒ '*b*) (*Suc n*)
≤ *restriction-δ TYPE*('*a* ⇒ '*b*) ∗ *restriction-σ TYPE*('*a* ⇒ '*b*)
n› **for** *n*
by (*simp add*: *restriction-σ-le restriction-σ-fun-def restriction-δ-fun-def*)
next
show ‹*dist f g* = *Inf* (*restriction-σ-related-set f g*)› **for** *f g* :: ‹'*a* ⇒
'*b*›
by (*simp add*: *dist-fun-def*)
qed

instance ‹*fun*› :: (*type*, *geometric-restriction-space*) *geometric-restriction-space*
proof *intro-classes*
show ‹*restriction-σ TYPE*('*a* ⇒ '*b*) *n* = *restriction-δ TYPE*('*a* ⇒
'*b*) ^ *n*› **for** *n*
by (*simp add*: *restriction-σ-fun-def restriction-σ-is restriction-δ-fun-def*)
next
show ‹*dist f g* = *Inf* (*restriction-σ-related-set f g*)› **for** *f g* :: ‹'*a* ⇒
'*b*›

by (*simp add*: *dist-fun-def*)
qed

lemma *dist-image-le-dist-fun* : ‹*dist* (*f x*) (*g x*) ≤ *dist f g*›
for *f g* :: ‹'*a* ⇒ '*b* :: *non-decseq-restriction-space*›
proof (*unfold dist-restriction-is*, *rule cInf-superset-mono*)
show ‹*restriction-σ-related-set f g* ≠ {}› **by** *simp*
next
show ‹*bdd-below* (*restriction-σ-related-set* (*f x*) (*g x*))›
by (*simp add*: *bounded-imp-bdd-below bounded-restriction-σ-related-set*)
next
show ‹*restriction-σ-related-set f g* ⊆ *restriction-σ-related-set* (*f x*) (*g x*)›
unfolding *restriction-fun-def restriction-σ-fun-def*
by (*simp add*: *image-def subset-iff*) *metis*
qed

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*›
by (*simp add*: *dist-image-le-dist-fun cSUP-least*)
— The other inequality will require the additional assumption *decseq*.

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

lemma *reached-dist-fun* : ‹∃ *x*. *dist f g* = *dist* (*f x*) (*g x*)›
proof (*cases* ‹*f* = *g*›)
show ‹*f* = *g* ⟹ ∃ *x*. *dist f g* = *dist* (*f x*) (*g x*)› **by** *simp*
next
assume ‹*f* ≠ *g*›
let *?n* = ‹*Sup* (*restriction-related-set f g*)›
from *not-eq-imp-dist-restriction-is-restriction-σ-Sup-restriction-eq-simplified*(*2*, *3*)[*OF* ‹*f* ≠ *g*›]
obtain *x* **where** ‹∀ *m*≤*?n*. *f x* ↓ *m* = *g x* ↓ *m*› ‹*f x* ↓ *Suc ?n* ≠ *g x* ↓ *Suc ?n*›
unfolding *restriction-fun-def* **by** (*meson lessI*)
hence ‹*dist* (*f x*) (*g x*) = *restriction-σ TYPE*('*b*) *?n*›
by (*metis* (*no-types*, *lifting*) *le-neq-implies-less not-less-eq-eq*
not-eq-imp-dist-restriction-is-restriction-σ-Sup-restriction-eq-simplified)
with *not-eq-imp-dist-restriction-is-restriction-σ-Sup-restriction-eq-simplified*(*1*)
[*OF* ‹*f* ≠ *g*›, *unfolded restriction-σ-fun-def*]
have ‹*dist f g* = *dist* (*f x*) (*g x*)› **by** *simp*
thus ‹∃ *x*. *dist f g* = *dist* (*f x*) (*g x*)› **..**
qed

lemma *dist-fun-eq-Sup-dist-image* : ‹*dist f g* = (*SUP x*. *dist* (*f x*) (*g x*))›
proof (*rule order-antisym*)
 show ‹(*SUP x*. *dist* (*f x*) (*g x*)) ≤ *dist f g*› **by** (*fact Sup-dist-image-le-dist-fun*)
next
 from *reached-dist-fun* **obtain** *x* **where** ‹*dist f g* = *dist* (*f x*) (*g x*)›
..
 thus ‹*dist f g* ≤ (*SUP x*. *dist* (*f x*) (*g x*))›
 proof (*rule ord-eq-le-trans*)
  show ‹*dist* (*f x*) (*g x*) ≤ (*SUP x*. *dist* (*f x*) (*g x*))›
  proof (*rule cSup-upper*)
   show ‹*dist* (*f x*) (*g x*) ∈ *range* (λ*x*. *dist* (*f x*) (*g x*))› **by** *simp*
  next
   show ‹*bdd-above* (*range* (λ*x*. *dist* (*f x*) (*g x*)))›
   by (*rule bdd-aboveI*[*of* - ‹*dist f g*›]) (*auto intro*: *dist-image-le-dist-fun*)
  qed
 qed
qed

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*›
 by (*use reached-dist-fun* **in** ‹*auto simp add*: *dist-image-le-dist-fun*›)

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 $'a \Rightarrow 'b$ 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*
 by *intro-classes* (*fact restriction-chain-imp-restriction-convergent*)

instance ‹*fun*› :: (*type*, *complete-strict-decseq-restriction-space*) *complete-strict-decseq-restriction-space*
 by *intro-classes* (*fact restriction-chain-imp-restriction-convergent*)

instance ‹*fun*› :: (*type*, *complete-at-least-geometric-restriction-space*) *complete-at-least-geometric-restriction-space*
by *intro-classes* (*fact restriction-chain-imp-restriction-convergent*)

instance ‹*fun*› :: (*type*, *complete-geometric-restriction-space*) *complete-geometric-restriction-space*
by *intro-classes* (*fact restriction-chain-imp-restriction-convergent*)

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 (*intro iffI allI*)
fix *y* **assume** *assm* : ‹*lipschitz-with f α*›
show ‹*lipschitz-with* (λ*x*. *f x y*) *α*›
proof (*rule lipschitz-withI*)
from *assm*[*THEN lipschitz-withD1*] **show** ‹*0* ≤ *α*› **.**
next
show ‹*dist* (*f x1 y*) (*f x2 y*) ≤ *α* ∗ *dist x1 x2*› **for** *x1 x2*
by (*rule order-trans*[*OF - assm*[*THEN lipschitz-withD2*]])
(*simp add*: *dist-image-le-dist-fun*)
qed
next
assume *assm* : ‹∀ *y*. *lipschitz-with* (λ*x*. *f x y*) *α*›
show ‹*lipschitz-with f α*›
proof (*rule lipschitz-withI*)
from *assm*[*rule-format*, *THEN lipschitz-withD1*] **show** ‹*0* ≤ *α*› **.**
next
fix *x1 x2*
obtain *y* **where** ‹*dist* (*f x1*) (*f x2*) = *dist* (*f x1 y*) (*f x2 y*)›
by (*meson reached-dist-fun*)
also have ‹... ≤ *α* ∗ *dist x1 x2*› **by** (*rule assm*[*rule-format*, *THEN lipschitz-withD2*])
finally show ‹*dist* (*f x1*) (*f x2*) ≤ *α* ∗ *dist x1 x2*› **.**
qed
qed

lemma *non-expanding-simplification* :
‹*non-expanding f* ⟷ (∀ *y*. *non-expanding* (λ*x*. *f x y*))›
by (*metis lipschitz-with-simplification non-expanding-on-def*)

lemma *contraction-with-simplification*:
‹*contraction-with f α* ⟷ (∀ *y*. *contraction-with* (λ*x*. *f x y*) *α*)›
by (*metis contraction-with-on-def lipschitz-with-simplification*)

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-prod*$_{max}$ *to-prod*$_{max}$ **by** *simp*

— Simplifications because the **typedef** is trivial.

declare *from-prod*$_{max}$*-inject* [*simp*]
from-prod$_{max}$*-inverse* [*simp*]

lemmas *to-prod*$_{max}$*-inject-simplified* [*simp*] = *to-prod*$_{max}$*-inject* [*simplified*]
and *to-prod*$_{max}$*-inverse-simplified*[*simp*] = *to-prod*$_{max}$*-inverse*[*simplified*]

lemmas *to-prod*$_{max}$*-induct-simplified* = *to-prod*$_{max}$*-induct*[*simplified*]
and *to-prod*$_{max}$*-cases-simplified* = *to-prod*$_{max}$*-cases* [*simplified*]
and *from-prod*$_{max}$*-induct-simplified* = *from-prod*$_{max}$*-induct*[*simplified*]
and *from-prod*$_{max}$*-cases-simplified* = *from-prod*$_{max}$*-cases* [*simplified*]

setup-lifting *type-definition-prod*$_{max}$

lift-definition *Pair*$_{max}$ :: ‹$'a$ ⇒ $'b$ ⇒ $'a$ ×$_{max}$ $'b$› **is** *Pair* **.**

free-constructors *case-prod*$_{max}$ **for** *Pair*$_{max}$ *fst*$_{max}$ *snd*$_{max}$
by (*metis Pair*$_{max}$*.abs-eq from-prod*$_{max}$*-inverse surjective-pairing*)
(*metis Pair*$_{max}$*.rep-eq prod.inject*)

lemma *fst*$_{max}$*-def* : ‹*fst*$_{max}$ ≡ *map-fun from-prod*$_{max}$ *id fst*›
by (*intro eq-reflection ext*, *simp add*: *fst*$_{max}$*-def*,

metis $Pair_{max}$*.rep-eq from-prod*$_{max}$*-inverse* fst_{max}*-def prod.collapse* $prod_{max}$*.sel(1)*)

lemma fst_{max}*-rep-eq* : ‹fst_{max} *x* = *fst* (*from-prod*$_{max}$ *x*)›
by (*metis* $Pair_{max}$*.rep-eq fst-conv* $prod_{max}$*.collapse*)

lemma fst_{max}*-abs-eq* [*simp*] : ‹fst_{max} (*to-prod*$_{max}$ *y*) = *fst y*›
by (*metis* $Pair_{max}$*.abs-eq prod.exhaust-sel* $prod_{max}$*.sel(1)*)

lemma fst_{max}*-transfer* [*transfer-rule*]: ‹*rel-fun* (*pcr-prod*$_{max}$ (=) (=)) (=) *fst* fst_{max}›
by (*metis* (*mono-tags*) $Pair_{max}$*.rep-eq cr-prod*$_{max}$*-def fst-conv* $prod_{max}$*.collapse* $prod_{max}$*.pcr-cr-eq rel-funI*)

lemma snd_{max}*-def* : ‹snd_{max} ≡ *map-fun from-prod*$_{max}$ *id snd*›
by (*intro eq-reflection ext*, *simp add*: snd_{max}*-def*,
metis $Pair_{max}$*.rep-eq from-prod*$_{max}$*-inverse prod.collapse* $prod_{max}$*.case*)

lemma snd_{max}*-rep-eq* : ‹snd_{max} *x* = *snd* (*from-prod*$_{max}$ *x*)›
by (*metis* $Pair_{max}$*.rep-eq* $prod_{max}$*.collapse snd-conv*)

lemma snd_{max}*-abs-eq* [*simp*] : ‹snd_{max} (*to-prod*$_{max}$ *y*) = *snd y*›
by (*metis* $Pair_{max}$*.abs-eq prod.exhaust-sel* $prod_{max}$*.sel(2)*)

lemma snd_{max}*-transfer* [*transfer-rule*] : ‹*rel-fun* (*pcr-prod*$_{max}$ (=) (=)) (=) *snd* snd_{max}›
by (*metis* (*mono-tags*, *lifting*) $Pair_{max}$*.rep-eq cr-prod*$_{max}$*-def* $prod_{max}$*.collapse* $prod_{max}$*.pcr-cr-eq rel-funI snd-conv*)

lemma *case-prod*$_{max}$*-def* : ‹*case-prod*$_{max}$ ≡ *map-fun id* (*map-fun from-prod*$_{max}$ *id*) *case-prod*›
by (*intro eq-reflection ext*, *simp add*: $prod_{max}$*.case-eq-if* fst_{max}*-rep-eq* snd_{max}*-rep-eq split-beta*)

lemma *case-prod*$_{max}$*-rep-eq* : ‹*case-prod*$_{max}$ *f p* = (*case from-prod*$_{max}$ *p of* (*x*, *y*) ⇒ *f x y*)›
by (*simp add*: fst_{max}*-rep-eq* $prod_{max}$*.case-eq-if* snd_{max}*-rep-eq split-beta*)

lemma *case-prod*$_{max}$*-abs-eq* [*simp*] : ‹*case-prod*$_{max}$ *f* (*to-prod*$_{max}$ *q*) = (*case q of* (*x*, *y*) ⇒ *f x y*)›
by (*simp add*: $prod_{max}$*.case-eq-if split-beta*)

lemma *case-prod*$_{max}$*-transfer* [*transfer-rule*] : ‹*rel-fun* (=) (*rel-fun* (*pcr-prod*$_{max}$ (=) (=)) (=)) *case-prod case-prod*$_{max}$›
by (*simp add*: *cr-prod*$_{max}$*-def* fst_{max}*-rep-eq* $prod_{max}$*.case-eq-if* $prod_{max}$*.pcr-cr-eq rel-fun-def* snd_{max}*-rep-eq split-beta*)

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** $\langle(\downarrow)\rangle$ **.**

lemma $restriction\text{-}prod_{max}\text{-}def'$: $\langle p \downarrow n = \{\!|fst_{max}\ p \downarrow n,\ snd_{max}\ p \downarrow n|\!\}\rangle$
by *transfer* (*simp add*: *restriction-prod-def*)

instance by (*intro-classes*, *transfer*, *simp*)

end

instance $prod_{max}$ :: (*restriction-space*, *restriction-space*) *restriction-space*
by (*intro-classes*; *transfer*) (*simp-all add*: *ex-not-restriction-related*)

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 by *intro-classes*
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 *intro-classes*
show ‹*restriction-σ TYPE*('*a* ×$_{max}$ '*b*) ⟶ *0*›
by (*rule real-tendsto-sandwich*
[*of* ‹λ*n*. *0*› - - ‹λ*n*. *restriction-σ TYPE*('*a*) *n* + *restriction-σ TYPE*('*b*) *n*›])
(*simp-all add*: *order-less-imp-le restriction-σ-prod*$_{max}$*-def max-def restriction-σ-tendsto-zero tendsto-add-zero*)
qed (*simp-all add*: *uniformity-prod*$_{max}$*-def open-prod*$_{max}$*-def restriction-σ-prod*$_{max}$*-def max-def dist-prod*$_{max}$*-def*)

end

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

show ‹*decseq* (*restriction-σ* *TYPE*('*a* ×$_{max}$ '*b*))›
proof (*intro decseq-SucI*)
show ‹*restriction-σ* *TYPE*('*a* ×$_{max}$ '*b*) (*Suc n*) ≤ *restriction-σ* *TYPE*('*a* ×$_{max}$ '*b*) *n*› **for** *n*
using *decseq-SucD*[*of* ‹*restriction-σ* *TYPE*('*a*)› *n*]
decseq-SucD[*of* ‹*restriction-σ* *TYPE*('*b*)› *n*]
by (*auto simp add*: *restriction-σ-prod*$_{max}$*-def decseq-restriction-σ*)
qed
qed

instance *prod*$_{max}$ :: (*strict-decseq-restriction-space*, *strict-decseq-restriction-space*) *strict-decseq-restriction-space*
proof *intro-classes*
show ‹*strict-decseq* (*restriction-σ* *TYPE*('*a* ×$_{max}$ '*b*))›
proof (*intro strict-decseq-SucI*)
show ‹*restriction-σ* *TYPE*('*a* ×$_{max}$ '*b*) (*Suc n*) < *restriction-σ* *TYPE*('*a* ×$_{max}$ '*b*) *n*› **for** *n*
using *strict-decseq-SucD*[*of* ‹*restriction-σ* *TYPE*('*a*)› *n*]
strict-decseq-SucD[*of* ‹*restriction-σ* *TYPE*('*b*)› *n*]
by (*auto simp add*: *restriction-σ-prod*$_{max}$*-def strict-decseq-restriction-σ*)
qed
qed

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 by *intro-classes* (*simp-all add*: *restriction-δ-prod*$_{max}$*-def max-def*)

end

instance *prod*$_{max}$ :: (*at-least-geometric-restriction-space*, *at-least-geometric-restriction-space*) *at-least-geometric-restriction-space*
proof *intro-classes*
show ‹*0* < *restriction-σ* *TYPE*('*a* ×$_{max}$ '*b*) *n*› **for** *n* **by** *simp*
next
show ‹*restriction-σ* *TYPE*('*a* ×$_{max}$ '*b*) (*Suc n*)
≤ *restriction-δ* *TYPE*('*a* ×$_{max}$ '*b*) ∗ *restriction-σ* *TYPE*('*a* ×$_{max}$ '*b*) *n*› **for** *n*
by (*auto intro*: *order-trans*[*OF restriction-σ-le*]
simp add: *restriction-δ-prod*$_{max}$*-def mult-mono*′ *restriction-σ-prod*$_{max}$*-def*)
next
show ‹*dist p1 p2* = *Inf* (*restriction-σ-related-set p1 p2*)› **for** *p1 p2* :: ‹'*a* ×$_{max}$ '*b*›
by (*simp add*: *dist-prod*$_{max}$*-def*)

qed

instance $prod_{max}$ :: (*geometric-restriction-space*, *geometric-restriction-space*) *geometric-restriction-space*
proof *intro-classes*
show ‹*restriction-σ TYPE*('*a* ×$_{max}$ '*b*) *n* = *restriction-δ TYPE*('*a* ×$_{max}$ '*b*) ^ *n*› **for** *n*
by (*simp add*: *restriction-σ-prod*$_{max}$*-def restriction-σ-is restriction-δ-prod*$_{max}$*-def max-def*)
(*meson nle-le power-mono zero-le-restriction-δ*)
next
show ‹*dist p1 p2* = *Inf* (*restriction-σ-related-set p1 p2*)› **for** *p1 p2* :: ‹'*a* ×$_{max}$ '*b*›
by (*simp add*: *dist-prod*$_{max}$*-def*)
qed

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 (*unfold dist-restriction-is max-def*, *split if-split*, *intro conjI impI*)
show ‹*Inf* (*restriction-σ-related-set* (*snd*$_{max}$ *p1*) (*snd*$_{max}$ *p2*)) ≤ *Inf* (*restriction-σ-related-set p1 p2*)›
proof (*rule cINF-superset-mono*[*OF nonempty-restriction-related-set*])
show ‹*bdd-below* (*restriction-σ-related-set* (*snd*$_{max}$ *p1*) (*snd*$_{max}$ *p2*))›
by (*meson bdd-belowI2 zero-le-restriction-σ*)
qed (*simp-all add*: *subset-iff add*: *restriction-prod*$_{max}$*-def*' *restriction-σ-prod*$_{max}$*-def*)
next
show ‹*Inf* (*restriction-σ-related-set* (*fst*$_{max}$ *p1*) (*fst*$_{max}$ *p2*)) ≤ *Inf* (*restriction-σ-related-set p1 p2*)›
proof (*rule cINF-superset-mono*[*OF nonempty-restriction-related-set*])
show ‹*bdd-below* (*restriction-σ-related-set* (*fst*$_{max}$ *p1*) (*fst*$_{max}$ *p2*))›
by (*meson bdd-belowI2 zero-le-restriction-σ*)
qed (*simp-all add*: *subset-iff add*: *restriction-prod*$_{max}$*-def*' *restriction-σ-prod*$_{max}$*-def*)
qed

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* ×$_{max}$

$'b)$›
unfolding *restriction-σ-prod$_{max}$-def* **by** (*rule ext*) *simp*

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*)›
unfolding *dist-left-prod$_{max}$-def dist-right-prod$_{max}$-def*
by (*subst restriction-σ-prod$_{max}$-commute*) *simp*

lemma *dist-le-dist-left-prod$_{max}$* : ‹*dist x y* ≤ *dist-left-prod$_{max}$ TYPE*(*'a* ×$_{max}$ *'b*) *x y*›
proof (*unfold dist-left-prod$_{max}$-def dist-restriction-is*,
rule cINF-mono[*OF nonempty-restriction-related-set*[*of x y*]])
show ‹*bdd-below* (*restriction-σ-related-set x y*)›
by (*meson bdd-belowI2 zero-le-restriction-σ*)
next
show ‹*m* ∈ *restriction-related-set x y* ⟹
∃ *n*∈*restriction-related-set x y. σ*$_{\downarrow}$ *TYPE*(*'a*) *n* ≤ *σ*$_{\downarrow}$ *TYPE*(*'a* ×$_{max}$ *'b*) *m*› **for** *m*
by (*metis max.cobounded1 restriction-σ-prod$_{max}$-def*)
qed

lemma *dist-le-dist-right-prod$_{max}$* : ‹*dist x y* ≤ *dist-right-prod$_{max}$ TYPE*(*'b* ×$_{max}$ *'a*) *x y*›
by (*simp add*: *dist-le-dist-left-prod$_{max}$ dist-right-prod$_{max}$-is-dist-left-prod$_{max}$*)

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 −
interpret *left* : *DecseqRestrictionSpace* ‹(↓)› ‹(=)› ‹*UNIV*›
‹*restriction-σ TYPE*(*'a* ×$_{max}$ *'b*)› ‹*dist-left-prod$_{max}$ TYPE*(*'a* ×$_{max}$ *'b*)›

by *unfold-locales*
(*simp-all add*: *restriction-σ-tendsto-zero dist-left-prod$_{max}$-def decseq-restriction-σ*)

interpret *right* : *DecseqRestrictionSpace* ‹(↓)› ‹(=)› ‹*UNIV* :: *'b set*› ‹*restriction-σ TYPE('a ×$_{max}$ 'b)*› ‹*dist-right-prod$_{max}$ TYPE('a ×$_{max}$ 'b)*›
by *unfold-locales*
(*simp-all add*: *restriction-σ-tendsto-zero dist-right-prod$_{max}$-def decseq-restriction-σ*)

show ‹*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)*)›
by (*auto simp add*: *dist-restriction-is-bis left.dist-restriction-is-bis right.dist-restriction-is-bis prod$_{max}$.expand restriction-prod$_{max}$-def'*)
(*smt* (*verit*, *best*) *Collect-cong nle-le restriction-related-le*)
qed

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*))›
by (*simp add*: *restriction-chain-def*, *transfer*)
(*metis fst-conv prod.collapse restriction-prod-def snd-conv*)

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›
by (*simp add*: *restriction-tendsto-def*, *transfer*, *simp add*: *restriction-prod-def*)
(*meson nle-le order.trans*)

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*))›
by (*simp add*: *restriction-convergent-def restriction-tendsto-prod$_{max}$-iff*)
(*metis prod$_{max}$.sel*)

instance $prod_{max}$ :: (*complete-decseq-restriction-space*, *complete-decseq-restriction-space*)
complete-decseq-restriction-space
proof (*intro-classes*, *transfer*)
fix σ :: ‹*nat* ⇒ $'a \times_{max} 'b$› **assume** ‹$chain_{\downarrow}$ σ›
hence ‹$chain_{\downarrow}$ ($\lambda n.$ fst_{max} (σ *n*))› ‹$chain_{\downarrow}$ ($\lambda n.$ snd_{max} (σ *n*))›
by (*simp-all add*: *restriction-chain-prod*$_{max}$*-iff*)
hence ‹$convergent_{\downarrow}$ ($\lambda n.$ fst_{max} (σ *n*))› ‹$convergent_{\downarrow}$ ($\lambda n.$ snd_{max} (σ *n*))›
by (*simp-all add*: *restriction-chain-imp-restriction-convergent*)
thus ‹$convergent_{\downarrow}$ σ›
by (*simp add*: *restriction-convergent-prod*$_{max}$*-iff*)
qed

instance $prod_{max}$ :: (*complete-strict-decseq-restriction-space*, *complete-strict-decseq-restriction-space*)
complete-strict-decseq-restriction-space
by *intro-classes* (*simp add*: *restriction-chain-imp-restriction-convergent*)

instance $prod_{max}$ :: (*complete-at-least-geometric-restriction-space*, *complete-at-least-geometric-restriction-space*)
complete-at-least-geometric-restriction-space
by *intro-classes* (*simp add*: *restriction-chain-imp-restriction-convergent*)

instance $prod_{max}$ :: (*complete-geometric-restriction-space*, *complete-geometric-restriction-space*)
complete-geometric-restriction-space
by *intro-classes* (*simp add*: *restriction-chain-imp-restriction-convergent*)

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($'a \times_{max} 'b$) = *restriction-σ TYPE*($'a$)›
unfolding *restriction-σ-prod*$_{max}$*-def* **by** *simp*

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* $\times_{max}$ $'b$ :: *decseq-restriction-space*›
proof (*rule order-antisym*)
have ‹*dist-left-prod*$_{max}$ *TYPE*($'a \times_{max} 'b$) (fst_{max} *p1*) (fst_{max} *p2*) = *dist* (fst_{max} *p1*) (fst_{max} *p2*)›
by (*simp add*: *dist-left-prod*$_{max}$*-def dist-restriction-is*)
moreover have ‹*dist-right-prod*$_{max}$ *TYPE*($'a \times_{max} 'b$) (snd_{max} *p1*) (snd_{max} *p2*) = *dist* (snd_{max} *p1*) (snd_{max} *p2*)›

by (*simp add*: *dist-right-prod*$_{max}$*-def dist-restriction-is*)
ultimately show ‹*dist p1 p2* ≤ *max* (*dist* (*fst*$_{max}$ *p1*) (*fst*$_{max}$ *p2*)) (*dist* (*snd*$_{max}$ *p1*) (*snd*$_{max}$ *p2*))›
by (*metis dist-prod*$_{max}$*-le-max-dist-left-prod*$_{max}$*-dist-right-prod*$_{max}$)
next
show ‹*max* (*dist* (*fst*$_{max}$ *p1*) (*fst*$_{max}$ *p2*)) (*dist* (*snd*$_{max}$ *p1*) (*snd*$_{max}$ *p2*)) ≤ *dist p1 p2*›
by (*fact max-dist-projections-le-dist-prod*$_{max}$)
qed

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

We could of course imagine more support for $'a \times_{max} 'b$ 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